

KANDIDATAFHANDLING
Cand.merc.(mat.)

STATISTISK PRÆDIKTION AF KRYPTOVALUTAEN BITCOIN

En statistisk baseret handelsstrategi

Vejleder:
Professor Søren Feodor Nielsen, Center for Statistik

Forfattere:
Reidar Luid (46104)
Niklas Hansson (8589)

Afleveret:
14. maj 2018

Antal sider:
101 (185.000)

Abstract

The purpose of this master thesis was to examine whether it's possible to predict the future *USD*-returns of the cryptocurrency *Bitcoin*.

This was conducted through the use of the statistical models *Elastic Net*, *Random Forest*, *Gradient Boosting Machine* and the *Artificial Neural Network*, where technical indicators were used as features – all within a walk-forward-optimization approach.

The predictions from each of these models was to be used as inputs in their own respective trading strategy. The *walk-forward-optimization* approach allowed both the models' hyperparameters and their corresponding trading strategy's trading signals to adapt to potential changes in the *BTC/USD* market. Each trading strategy's total return – conditional on a few market assumptions – was to be compared to a *buy-and-hold* strategy's total return over the same period.

Furthermore, the use of *stacking* was explored, in an attempt, to improve both the statistical results and the trading results. This implied constructing 4 new models, labeled *meta-learners*, based on the same algorithm-class as the above-mentioned models, labeled *base-learners*, but where the feature-space of each *meta-learner* contained the *predictions* of all four *base-learners*.

The findings of this thesis suggest, that it's somewhat possible to solve this prediction-problem, as all 4 *base-learners* achieved a lower *mean-absolute-error* than the 2 benchmark-models, *mean-* and *zero-prediction*.

Additionally, the total return of each *base-learner*'s corresponding trading strategy produced a higher total return than the buy-and-hold strategy.

It was also found, that the *stacking* approach was able to improve upon the statistical results, as out of all 8 models, 3 of the best 4 models were *meta-learners* and the best model was a *meta-learner*. Finally, the use of *stacking* was able to improve the trading results as well, as 3 of the best trading strategies, conditional on the given market assumptions, contained predictions from *meta-learners*.

Forord

I forbindelse med udarbejdelsen af denne kandidatafhandling vil vi gerne sige stor tak til vores vejleder professor Søren Feodor Nielsen.

Han har igennem hele forløbet guidet os i den rigtige retning samt bidraget med stor faglig viden og besvarelse af alverdens spørgsmål. Han har været en inspiration for os igennem vores tid på CBS.

Frederiksberg, maj 2018

Reidar Luid

Niklas Hansson

Indhold

1	Indledning	1
1.1	Motivation og metode	2
1.2	Problemformulering.....	4
1.3	Afgrænsning	4
1.4	Afhandlingens struktur	4
2	Kryptovaluta markedet	6
2.1	Traditionelle markeder ifht. kryptovaluta markedet.....	6
2.2	Markedets mikrostruktur.....	7
2.3	Teknisk analyse af prisbevægelser.....	9
2.4	Et eksempel med teknisk analyse	10
3	Data.....	14
3.1	Tidsintervaller	15
3.2	Kort om API'er.....	16
3.3	Scraping.....	16
3.4	Responsen.....	18
3.5	Features	20
4	Teori og metode.....	25
4.1	Evaluering.....	25
4.1.1	MAE-fejlmålet	25
4.1.2	Benchmark modeller.....	26
4.1.3	Risiko	27
4.2	Modeller.....	28
4.2.1	Elastic Net	28
4.2.2	Træ-baserede Ensembles.....	29
4.2.3	<i>Artificial Neural Network</i>	34
4.3	Stacking	37
4.4	<i>Walk-forward-optimization</i> metoden.....	38
5	Feature extraction og feature engineering.....	40
6	Modellering.....	43
6.1	Base-learners	43
6.1.1	Elastic Net	44
6.1.2	Random Forest.....	47

6.1.3	Gradient Boosting Machine	50
6.1.4	Artificial Neural Network	53
6.2	<i>Grid-search</i>	56
6.2.1	Elastic Net (<i>Cartesian</i> -) <i>grid-search</i>	58
6.2.2	Random Forest (<i>random</i> -) <i>grid-search</i>	61
6.2.3	Opsummering.....	68
6.3	Meta-learners	69
6.3.1	Opsummering.....	71
7	Walk-forward-optimization	73
7.1	Redegørelse af vores <i>trading</i> strategi.....	75
7.2	Resultater fra <i>walk-forward-optimization</i>	82
7.3	Detaljer og overvejelser	89
8	Diskussion og perspektivering	96
9	Konklusion.....	100
	Bilag.....	102
	Oversigt af filer i det digitale bilag	102
	Kryptovalutamarkedet	104
	Modellering.....	105
	GLM	105
	GRID SEARCH.....	106
	STACKING	107
	Walk-forward-optimization	109
	Resultater fra walk-forward-optimization	109

1 Indledning

I oktober 2008, inde på en obskur email-liste dedikeret til kryptografikyndige, offentliggjorde den stadig ukendte Satoshi Nakamoto et *whitepaper* (i.e. en forklarende artikel) med titlen "Bitcoin: A Peer-to-Peer Electronic Cash System". Ved at løse det kryptografiske *double-spend* problem, hvilket består i, at det er muligt at genbruge den samme enkelte digitale mønt, uden at dens involvering i tidligere transaktioner registreres (i.e. forfalskning), kunne han foreslå et revolutionerende, digitalt betalingssystem, der i teorien kun involverer betaler og modtager, og dermed ikke afhænger af en faciliterende *centraliseret* tredjepart (e.g. en bank), der bl.a. har til opgave at integritets sikre selve pengeopbevarelsen for de to parter, samt verificere hver enkelt transaktion, sådan at *double-spending* ikke forekommer.

I Bitcoin's *whitepaper* argumenterer Satoshi for, at denne nye form for et betalingssystem er nødvendig, da enhver centraliseret tredjepart med ovenstående ansvar, er udsat for partiskhed og er relativ nem at overrumple eller angribe, da den på mange måder kan anses for at befinde sig på et centralt punkt – f.eks. i form af en central bestyrelse med indflydelse på afgørende beslutninger eller en central server, der er modtagelige overfor ondsindede aktører. At sætte sin tillid til en sådan enhed, som forbruger, er derfor ikke uden risici, fortsætter han. I et forsøg på at rette op på dette, definerer han konceptet *blockchain*, hvor han i sin teori istedet fordeler verifikationsprocessen ud til et netværk af *nodes*, der matematisk så skal komme til en relativ enighed om gyldigheden af hver enkelt enkrypteret transaktion, i en blok af transaktioner, der derefter kronologisk knyttes til en allerede verificeret og fordelt, og dermed irreversibel, kæde af blokke.

Da Satoshi Nakamoto i januar 2009 møntede de første *Bitcoins* ved at verificere den allerførste blok (den såkaldte *genesis block*) i en kæde der nu snart er 10 år gammel, blev det startskuddet til en af de allerbedste investeringsmuligheder nogensinde. På det tidspunkt var det for det meste kun relevante fagfolk, diverse idealister og tech-entusiaster der havde læst *Bitcoins whitepaper*, og var interesserede i at hverve sig *Bitcoins* – enten ved at tilbyde computerkraft som *node* i opretholdelsen af netværket, og derved modtage *Bitcoins* som belønning, eller ved at veksle fiat (i.e. traditionelle penge) om til *Bitcoins* på diverse kryptobørser på internettet, der var hurtige at springe op.

Siden har idéen tiltrukket flere og flere, hvilket også reflekteres i *Bitcoin* prisens voldsomme stigning. Med en startpris på omtrentligt *0.05 USD* i midten af 2010, til dens højeste pris på lige under *20.000 USD* den 17. december 2017 – altså en værdi på ca. 400.000 gange initialbeløbet for de allerførste købere.

Siden *Bitcoin's* fødsel er antallet af forskellige kryptovalutaer steget voldsomt til flere end 1500 i skrivende stund. Hvor *Bitcoins* oprindelige formål var at blive en form for *digital cash*, så er der andre kryptovalutaer der bruger *blockchain* teknologien til andre formål. Tag f.eks. den næststørste i samlet markedsværdi, *Ethereum*, der stiler efter at være en platform for programmer kørende på en blockchain, og som nogle ser som det bedste bud på verdens første *decentraliserede* supercomputer, og dermed også som en investeringsmulighed i noget der – af dets tilhængere – bliver omtalt som det næste internet, i.e. *Web 3.0*.

Generelt set, fremstiller udviklerne af de allerfleste kryptovalutaer deres produkt som et fantastisk svar på diverse digitale problemer; disse omfatter typisk problemer omkring tillid, endegyldighed af transaktioner, ekstra omkostninger vedr. for mange mellemlid eller det meget aktuelle spørgsmål omkring digital privathed, for at nævne nogle få.

Med *use-cases* som disse, der i teorien muliggøres af den underliggende *blockchain* teknologi, er det ikke så mærkeligt, at kryptovalutamarkedet har tiltrukket så mange nye aktører, der jagter den næste 400.000x-investeringsmulighed.

Disse aktører kommer bl.a. i form af såkaldte *holders*, der køber en eller flere af de forskellige kryptovalutaer, og sælger kun efter længere tid, i håbet om at deres investering er blevet multipliceret mange gange.

En anden type er en *trader*, der forsøger sig med at analysere prisbevægelser vha. en række tekniske prisindikatorer, typisk i håbet om, at disse har en forudsigelseskraft, som *traderen* kan handle kortsigtet på.

I hvor høj grad ovenstående måde at handle ud fra på kryptovalutamarkedet, kan være successfuld eller ej i det lange løb, er et godt spørgsmål. Lige siden *Fama* og *Blume* i deres efterhånden gamle artikel fra 1965, "*Filter Rules and Stock-Market Trading*", fandt *holderens* strategi til at være mere fornuftig på de mere traditionelle valuta- og aktiemarkeder, har mange været skeptiske overfor tilgangen med teknisk analyse – også i kryptovalutamarkedet.

Derimod er det tænkeligt, at givet hvor spekulativt dette marked er, så har aktørerne der aktivt handler her, ikke så meget andet at basere deres forventninger til prisen på, end de tekniske prisindikatorer, da f.eks. *Bitcoin* ikke er et land eller en virksomhed, og dermed f.eks. ikke udgiver kvartalsmæssige eller årlige tal, en aktør kan udføre en fundamental analyse af.

Takket være den imponerende udvikling indenfor statistikkens verden igennem årene, samt den store computerkraft den sædvanlige person har til rådighed via *cloud*-baserede servere, er det måske muligt at komme med et svar til ovenstående spørgsmål.

1.1 Motivation og metode

I denne afhandling ønsker vi at give læseren indblik i, hvordan et prædiktionsproblem af det fremtidige afkast for kryptovalutaen, *Bitcoin*, kan opstilles vha, statistiske modeller og hvorvidt

resultater fra dette endvidere kan benyttes i en profitabel *trading* strategi, givet nogle specifikke antagelser.

Som vi rørte på i indledningen, så er der uenighed omkring hvorvidt brugen af teknisk analyse på de finansielle markeder, er et nyttigt redskab i forudsigelsen af fremtidige prisbevægelser. Givet kryptovalutamarkedets høje spekulationsniveau, er vi nysgerrige efter at se, hvorvidt en tilgang baseret på teknisk analyse, kan vise sig at være fordelagtig på dette marked. Vi er derfor interesserede i at benytte tekniske prisindikatorer som forklarende variable i vores statistiske modeller.

Givet det høje antal af forklarende variable, samt den store mængde 1-minut observationer, vi kommer til at arbejde med, er vi bevidste om at anvendelsen af de typer modeller der har vist sig at være i stand til at løse nogle formidable prædiktionsproblemer, som f.eks. en *Random Forest* eller et *Artificial Neural Network*, kan føre til bedre prædiktioner under disse forhold, end f.eks. en *Ordinary-Least-Squares*.

Idet vi primært er interesserede i at opnå så gode prædiktioner som muligt, har vi valgt at udforske en metode der bygger på prædiktioner fra andre modeller – nemlig *stacked generalization*, ofte forkortet til *stacking*.

Kort sagt, så drejer det sig om, først at opbygge en række statistiske modeller med forklarende variable (hyppigt kaldet *features* i denne afhandling) man mener kan have indflydelse på sin respons, og herefter benytte prædiktioner fra disse som forklarende variable i en efterfølgende statistisk model, fittet til samme respons.

Vi vil gerne se hvorvidt prædiktionsresultater for disse initiale modeller – som vi kalder for *base-learners* – kan forbedres ved at benytte de samme typer statistiske modeller, men hvor prædiktioner fra samtlige af vores *base-learners* istedet benyttes som forklarende variable. Disse udvidede modeller, kalder vi for *meta-learners*.

Prædiktionskraften for både *base-* og *meta-learners* vælger vi at evaluere imod to forskellige *benchmark* modeller.

Da vi i alle vores statistiske modeller kommer til at arbejde med en høj mængde data, med mange forklarende variable, ønsker vi at benytte *H2O* pakken i programmeringsproget *R*, der udnytter paralleliseringmuligheder hvor disse findes.

Da det er fremtidigt afkast for *Bitcoin* vi kommer til at prædiktere, er det oplagt at se hvorvidt vores *base-* og *meta-learners* kan benyttes i hver sin adaptive *trading* strategi.

Under en række antagelser, vil vi gerne gøre brug af *walk-forward-optimization* metoden, der tager højde for, at markedstendenser muligvis kan ændre sig over tid, og at ens strategi derfor også skal kunne have denne mulighed. Da strategierne kommer til at indeholde hver sin *learner*, samt en række *trading* signaler, tillader vi derfor, at hver *learners* hyperparametre, samt *trading* signaler, får muligheden for at tilpasse sig over den samlede *trading* periode, uden vores indblanding. Vi har tænkt os at sammenholde resultaterne fra de forskellige strategier med hvordan en *holder* ville klare sig over samme periode.

1.2 Problemformulering

I gennem *walk-forward-optimization* metoden vil vi forsøge at svare på følgende:

1. Kan man vha. statistiske modeller, med tekniske prisindikatorer som *features*, prædiktere fremtidigt afkast for kryptovalutaen *Bitcoin*?
2. Kan en *trading* strategi baseret på disse prædiktioner slå en køb-og-hold strategi?
3. Formår en *stacking* tilgang af de statistiske modeller at forbedre disse to resultater?

1.3 Afgrænsning

Vi valgte at begrænse os til kun at gøre brug af tekniske prisindikatorer som forklarende variable. Der er allerhelst mange andre forklarende variable vi kan inddrage end disse indikatorer, som f.eks. *social-media* data, men samtidigt er vi også nysgerrige efter at vide, om det er muligt at prædiktere fremtidige prisbevægelser *udelukkende* fra historiske prisbevægelser, samt den tilhørende volumen, da tekniske indikatorer ikke er andet end netop det.

Vores oprindelige idé med denne afhandling, var at benytte en af de statistisk baserede strategier vi ville udlede, i et *real-time trading* system, der aktivt handlede på en kryptovalutabørs. Her skulle nye observationer af *Bitcoin* prisen indsamles hvert minut, hvorefter der skulle prædiktes på dette og derefter handles ud fra prædiktionerne. Dette viste sig at til at ligge en smule udenfor de rammer vi satte omkring afhandlingen. Vi har derfor valgt at begrænse vores egentlige *Bitcoin trading*, til at blive eksekveret ud fra det historiske data vi har indsamlet, betinget af en række *trading* antagelser. I den forstand, kan det ses som et udkast til et eventuelt *real-time trading* system.

1.4 Afhandlingens struktur

I kapitel 2 ser vi lidt nærmere på hvordan kryptovalutamarkedet ser ud i en trading sammenhæng, samt hvordan to entusiastiske *traders* kan finde på at benytte teknisk analyse her.

Kapitel 3 beskriver vores data nærmere. Her defineres responsen, samt de typer tekniske indikatorer vi har valgt at benytte som vores *features*.

I kapitel 4 går vi igennem vores teori og metode. Her redegør vi for de 4 typer statistiske modeller vi kommer til at benytte, hvordan *stacking* tilgangen ser ud. Sidst i kapitlet beskrives *walk-forward-optimization* metoden.

Kapitel 5 beskriver hvordan vi konstruerer yderligere *features* ud fra de tekniske indikatorer, beskrevet i det tredje kapitel. Der redegøres endvidere for hvordan vi tager hånd om nogle af disse, for at gøre dem mere passende for vores modellering.

I kapitel 6 går vi igang med vores modellering, hvis endelige formål er at finde frem til passende hyperparameterintervaller for vores 4 *base-learners* samt 4 *meta-learners*.

I kapitel 7 eksekveres vores *walk-forward-optimization* metode, hvor vores 8 forskellige modeller, med hver sine passende hyperparameterintervaller, evalueres på *out-of-sample* data. Denne evaluering sker på et statistisk grundlag, men også op et afkastmæssigt grundlag.

I kapitel 8 diskuteres vores statistiske og afkastmæssige resultater, samt vores fremgangsmåde. Endvidere perspektiveres der på disse.

I kapitel 9, der også er det sidste, konkluderer vi på vores problemformulering.

2 Kryptovaluta markedet

I dette afsnit vil vi give en lille introduktion til kryptovalutamarkedet.

Vi ser først på, hvad nogle af de væsentligeste forskelle er mellem de mere traditionelle markeder og kryptovalutamarkedet, og hvordan manipulation muligvis spiller en rolle på markedet.

Ved at se på en forholdsvis meget anvendt hjemmeside, går vi igennem de nok mest basale elementer man skal være opmærksom på, når der handles på kryptovalutamarkedet.

Efterfølgende overvejer vi hvor relevant teknisk analyse er på kryptovalutamarkedet, og slutteligt giver vi et eksempel på hvordan denne tilgang kan blive brugt af *traders* på dette marked.

2.1 Traditionelle markeder ifht. kryptovaluta markedet

Ifht. de regulerede, traditionelle markeder, hvor myndigheder sætter dagsordenen og spillereglerne, bliver kryptovalutamarkedet ofte betegnet som det vilde vesten, der er åbent 24/7. Myndighederne i mange lande har nemlig stadig svært ved at klassificere kryptovaluta i en bestemt klasse (f.eks. aktiv eller valuta).

Som konsekvens, findes der et utal af *scammere*, der nogenlunde uhæmmet, forsøger at lokke nye, ivrige aktører der har anskaffet sig kryptovaluta, til at, i en eller anden form, give afkald på deres penge.

F.eks. ved at på Twitter, give sig ud til at være en velkendt krypto-personlighed, som f.eks. skaberen af *Ethereum*, Vitalik Buterin, der pludselig er i gavehumør. Det eneste du skal gøre, er nemlig at sende ham så og så mange *bitcoins*, *ether*, eller en anden kryptovaluta, til *wallet*-adressen han også har lagt ud, så sender Vitalik dig beløbet 5-10 gange igen. Dem som kun holder øje med det navn en bruger kan tilpasse – og som *scammeren* har ændet til noget der umiddelbart ligner Vitalik Buterins twitter-navn i dette tilfælde – istedet for profilens virkelige, uændrbare navn, er skæbnet til en brat opvågning, når der ikke sendes noget retur. Dette er muligt, idet man med kryptovaluta ”er sin egen bank”, og der dermed ikke er nogen at verificere om *wallet*-adressen i den anden ende virkelig tilhører Vitalik.

Det faktum at, *scammere* har succes med eksempler som ovenstående (dette kan verificeres, idet enhver altid kan tjekke, om der rent faktisk kommer kryptovaluta ind til den pågældende *wallet*-adresse) siger nok noget om, hvor ivrige efter gevinst, mange af aktørerne på markedet er.

En anden konsekvens af den manglende regulering, er pris manipulation.

Det er ikke helt til at vide hvor stor manipulationen er på kryptovalutamarkedet, men der er næppe tvivl om, at den eksisterer, taget i betragtning de profitmuligheder det kan medføre for

den potentielle manipulator.

F.eks. anklages endda børserne for at manipulere markedet. Det siges, at de løbende udfører såkaldt *wash-trading*, hvor børserne f.eks. sætter en stor mængde salgsordrer i ordrebogen, som de derefter selv køber, for at puste den samlede handelsvolume op, i håbet om at tiltrække flere aktører (<https://medium.com/@bitfinexed/wash-trading-bitcoin-how-bitfinex-benefits-from-fraudulent-trading-8bd66be73215>).

Et andet eksempel, er når en eller flere *whales* (i.e. rigtig store spillere) opstiller *buy-* eller *sell-walls*. Med f.eks. en *buy-wall* sætter en *whale* en købsordre under markedsprisen, der er så stor, at det kræver millioner af f.eks. dollars, at bryde den ned. De lille spillere ser denne *buy-wall*, og instinktivt tænker de, at prisen i den nærmeste fremtid, ikke falder under denne *buy-wall* pris. Det øger deres købsvillighed, og prisen stiger som konsekvens. Men hvad de ikke ved, er at den samme *whale* har sat en stor mængde lille salgs ordrer for en pris højere, men tæt på sin *buy-wall*. Så snart han har fået solgt den mængde han ønsker til denne lidt højere pris, end hvis der ikke var en *buy-wall*, så fjerner han sin store købsordre, der efter kort tid og alt andet lige, ofte får prisen til at falde under den pris som han satte for sin *buy-wall*.

Senere i dette kapitel, kommer vi til at redegøre for, hvordan teknisk analyse, der baseres på historiske prisbevægelser typisk benyttes.

Det er muligt, at disse *whales* også manipulerer med prisen, med hensigt på, at få de diverse tekniske indikatorer til at have nogle bestemte værdier, sådan at de aktører på markedet, der er tilhængere af tekniske analyse og handler ud fra disse indikatorer, er med til at skubbe prisen i en retning der er favorabel for den store spiller.

Ligesom de menneskelige aktører kan være udsatte overfor mulig prismanipulation, som de sidste to eksempler ovenover, så kan det også antages, at dette kan gælde for en adaptiv *trading* strategi, der benytter indikatorer baseret på prisen som forklarende variable.

2.2 Markedets mikrostruktur

For at give et indtryk af hvordan handel med kryptovaluta ser ud for en menneskelig aktør, ser vi på en meget anvendt *trading GUI* (*Graphical User Interface*), www.cryptowat.ch. Siden trækker, i real-time, på prisinformation fra de fleste store kryptobørser, og udregner og fremviser tekniske prisindikatorer som brugerne kan vælge at handle ud fra.

Vi ser på et af de mest indflydelsesrige kryptomarkeder, nemlig *Bitcoin* imod dollaren på børsen *Bitfinex* (se nedenstående udklip, se bilag for fuld figur). Fordi den daglige volume af handlet *Bitcoin* på dette marked i årevis har været høj ifht. andre børser, bevæger kursen sig ofte først her, og derefter følger markederne for *Bitcoin* imod dollaren på de andre børser efter.



Tickeren for markedet, *BTC/USD*, siger noget om hvor mange enheder af *quote currency (USD)* markedets *base currency (BTC)* er værd. Den nuværende pris, i.e. markedsprisen, som *tickeren* repræsenterer, baseres kun på prisen i den sidst eksekverede handel. Til højre i billedet er markedsprisen altså givet ved $BTC/USD = 7491.70$.

Ovenover denne *ticker* på billedet, er salgsordrer placeret til en pris der altid er højere end samtlige købsordrer, der vises nedenunder *tickeren*, og med tilhørende volumen for samtlige ordrer. Denne sektion kaldes *orderbook*, og til venstre for den illustreres dens kumulerede salgs- og købsvolumen, med hhv. den røde og grønne figur.

På børser med forholdsvis høj volumen, som *Bitfinex*, er *tickeren* en størrelse som skifter værdi mange gange i minuttet, da der næsten hele tiden udføres *ordermatching* imellem køberes og sælgeres ordrer. Et *match* indebærer, for eksempel, at en aktør har placeret en *limit sell order* (en salgsordre placeret til en pris højere end den højeste købsordre) og en anden aktør har placeret en købsordre til en pris lig med sælgers fastsatte pris. Betinget af, at købers ordre indeholder tilstrækkelig stor volumen (antal *BTC* han vil købe), så vil hans ordre kumulativt "æde" salgsordrer i *orderbook* – begyndende fra den første salgsordre, der er højere end den højeste købsordre, helt op til den omtalte sælgers ordre, hvilket medfører en *ny* markedspris, der er højere end tidligere.

Alle de ordrer, som køber "æder" på denne måde, medfører omkostning i form af *taker fee*, idet køber tager (i.e. fjerner) likviditet fra markedet. Den andel af købsordren, som bruges til at æde opad på denne måde, betegnes typisk som en *market buy order*.

Er der nogen resterende volumen bag købsordren efter den har matchet med ovennævnte salgsordre, så placeres ordren i *orderbook*, og kan nu defineres som den højeste *limit buy order*, der er lavere end den laveste salgsordre. Hvis nogen vælger at sælge ned i denne ordre, vil køber betale en anden type omkostning i form af *maker fee*, idet han nu tilføjer likviditet til markedet med den resterende volumen i sin købs ordre.

Maker fee er typisk lavere end *taker fee*, idet en børs gerne vil incentivere aktører til at tilføje

likviditet til markedet.

Forskellen på disse to typer af omkostning kan være meget vigtig at udnytte for en *trader*, alt efter om han handler på kort eller langt sigt og hvor volatilt markedet er.

Alle gennemførte handler registreres i *order history*, sektionen nederst i højre hjørne.

Øverst til venstre i figuren ses udviklingen i prisen over tid. Hver lodret streg hos grafen kaldes et *candle*, og er i dette tilfælde valgt til at representere 15 minutter (på *cryptowat.ch* kan hvert *candle* vælges til at repræsentere alt fra 1 minut, til 1 uge). Disse *candles* opbygges ud fra prisinformationerne lagret i *order history* for det tilsvarende tidsinterval; hvad prisen var ved tidsintervallets begyndelse, dens højeste og laveste værdi i intervallet og hvad prisen slutteligt var. Den samlede volume af handlet *BTC* for det tilsvarende tidsinterval skitseres nedunder grafen for prisudviklingen.

Formatet for disse 5 størrelser i prisinformationerne kaldes i samlet forstand *OHLCV* (*Open, High, Low, Close, Volume*), og kan benyttes til at konstruere diverse tekniske prisindikatorer, som *traders* kan handle ud fra.

Vi går mere i dybden omkring dette i kapitel 3.

2.3 Teknisk analyse af prisbevægelser

Det antages ofte, at der på et marked er to forskellige måder at handle ud fra – fundamental analyse og teknisk analyse. Imellem disse to, er der forskel på fremgangsmåden for analysen, samt over hvor lang tid en aktør typisk handler ud fra disse to.

Fundamental analyse går simpelthen ud på at analysere fundamentet for det aktiv eller den valuta, man er interesseret i at anskaffe sig. Er det f.eks. en virksomheds aktie man ser på, så går man typisk igennem virksomhedens regnskaber de seneste år, og inddrager f.eks. også hvordan udsigterne for virksomhedens markedssektor (f.eks. grøn energi) ser ud de kommende måneder eller år. Ud fra en analyse af dette, skabes der derefter en forventning om den fremtidige pris.

Ifht. teknisk analyse, er denne strategi typisk mere langsigtet.

I kryptovalutamarkedet kan en *holder*, som vi nævnte hvad er i indledningen, typisk klassificeres i denne gruppe. Han har dog ikke mulighed for analysere ting som regnskaber, men baserer i højere grad sin forventning til prisen på, hvad han mener den langsigtede fremtidige efterspørgsel af kryptovalutaen kommer til at være. For de helt store tilhængere af f.eks. *Bitcoin*, er der tale om en valuta, der kommer til at være en global valuta, som dollaren, indenfor en ikke alt for lang årrække.

Teknisk analyse er derimod langt mere kortsigtet.

Der ses udelukkende på indikatorer baseret på den historiske pris og volumen, og en aktiv

trader benytter typisk disse til at skabe sig en forventning om hvilken retning prisen kommer til at have i den nærmeste fremtid. Hans tidshorisont strækker sig ofte over nogle dage eller timer, men han kan også finde på at handle på minut-basis.

Som sagt tidligere, så kritiseres teknisk analyse ofte af mange. Det bliver nemlig ofte anset som noget meget uvidenskabeligt. Kan man virkelige sige noget om hvad prisen kommer til at være, udelukkende ud fra hvad prisen har været i fortiden, (husk, at tekniske indikatorer jo kun baseres på den historiske pris og dens volumen) eller er det meget ubrugbart?

Givet det første tilfælde og metoden virker, vil nogle mene, at dette kun gælder fordi det er en selvopfyldende profeti: hvis et stort nok antal aktører tror på, at disse indikatorer har en forklarende effekt, så vil prisen per automatik følge den retning, som de tekniske prisindikatorer forudsiger.

Som vi har nævnt tidligere, så kan man forestille sig at tilgangen med teknisk analyse benyttes meget på kryptovalutamarkedet, da aktørerne har så få solide fundamentaler at basere deres forventning til prisen på.

Hvis der findes tilstrækkeligt mange der handler ud fra tekniske indikatorer, kan det være at det bliver en selvopfyldende profeti, og at de derfor har en vis forklaringsgrad på prisen.

Om det hænger sådan sammen, bliver vi nok en smule klogere på i løbet af denne afhandling.

2.4 Et eksempel med teknisk analyse

For at få en smule intuition bag hvordan en kortsigtet *trader* kan finde på at handle ud fra tekniske indikatorer, går vi igennem et eksempel.

Vi ser på *BTC/USD* på *Bitfinex*, med 1 minut *candles*, og inkluderer 3 populære tekniske indikatorer. Se figuren på næste side.

Udover grafen for prisudviklingen og den tilhørende volumen, har vi også

- 3 *Exponential Moving Average* serier oven på pris grafen
 - Ses som udglatninger af prisen, med hhv. *lag* på (standard *Bitfinex* værdier) 10, 21 og 100.
- *MACD (Moving Average Convergence/Divergence)* under volumen
 - Opfattes traditionelt som en indikator der identificerer trenden for prisudviklingen, samt momentum bag trenden. Indeholder 2 *SMA*s der kan benyttes som signaler for gode købs- eller salgstidspunkter.
- samt *RSI (Relative Strength Index)* under *MACD*
 - En anden momentum indikator, der bruges til at identificere om aktivet er overkøbt eller oversolgt.

For at give et tilfredsstillende perspektiv på hvordan disse 3 indikatorer kan fortolkes, vil vi i det følgende klæde vi os om til et par entusiastiske *traders*, der i høj grad er tilhængere af teknisk analyse.

Den typisk overordnede strategi for en *trader* er at købe lavt og sælge højt, uden at bruge lånte penge. Vi ser derfor bort fra *gearede* handler, samt *shorting*.



Vi er 39 sekunder inde i det nuværende *candle*, og kl. er 13:37. Markedsprisen er 8864.2 USD. På omkring 120 minutter er den steget fra det lokale minimum på 8665 – svarende til en øgning på ca. 2.3%.

Vi vender blikket mod de tekniske prisindikatorer og ser på hvilken historie de kan fortælle os, begyndende med det *candle* der hører til kl 11:30.

Prisen er på vej ned, og de 3 EMA varianter følger efter prisgrafen med hver sin følsomhed. Nedgangens momentum er nogenlunde konstant, indikeret af de røde MACD søjler, der har en tilsvarende nogenlunde konstant højde.

Søjlehøjden svarer til forskellen mellem 2 EMA linjer, hvoraf den mere følsomme er kendt som MACD-linjen, der er forskellen mellem en 26- og 10-perioder EMA af prisen, mens den mindre følsomme er en udglatning af MACD-linjen i form af en 9-perioder EMA, også kendt som *signal*-linjen. Hvis MACD-linjen krydser under (over) *signal*-linjen, bliver søjlerne røde (grønne), hvilket indikerer, at det er tid til at sælge (købe) og hvis søjlerne er fyldte (tomme) er momentum bag den nuværende trend aftagende (øgende).

Ved 11:30 tiden begynder RSI at indikere at BTC/USD er oversolgt idet den bevæger sig under den typiske grænse på 35. Det ser mange som et godt købstidspunkt, og prisen finder også sit

lokale minimum nogle få minutter senere, opbakket af en relativt høj købsvolume. Med dette begynder *MACD*-linjen også at bevæge sig tættere og tættere op imod *signal*-linjen, indtil krydset indtræffer, og *MACD*-søjlerne flipper over til grønt.

Det varer dog ikke længe før prisen møder modstand i form af *100*-perioder *EMA*-linjen, hvilket en del aktører har valgt at sælge på. Selv om salgsvolumen er relativt høj, og *MACD*-linjen begynder at bevæge sig ned mod *signal*-linjen som en indikation på et fald i momentum bag prisstigningen, gør køberne et forsøg til efter nogle minutter, og formår at skubbe prisen tilstrækkeligt meget op til at den krydser *100*-perioder *EMA*.

Herfra, og indtil det endelige *candle*, fortsætter prisen den opadgående trend, men lader til at rammes af salgstryk når *RSI* krydser over *65*, hvor prisen befinder sig i territoriet for overkøbt. Salgstrykket, og det efterfølgende prisfald, er dog ikke nok til at presse prisen under *100*-perioder *EMA*, der nu er vendt fra at være en modstand, til at være en støtte ad flere omgange. I perioder agerer både *10*- og *21*-perioder *EMA* også som støtte, men pga. deres lavere *lags* opfattes de typisk som mindre troværdige støtter.

Som konklusion, præsenterer de tekniske indikatorer en række helt fine handelsmuligheder. Den første og mest oplagte, er ved det lokale minimum, hvor *MACD*-linjen begynder at bevæge sig imod *signal*-linjen, samtidig med at aktivet er oversolgt. Med en forventning om snarlig prisstigning, ud fra indikatorerne, skrider situationen ”køb”.

Vi klæder os om til os selv igen, og tager nogle kritiske briller på.

Den ovenstående analyse fortæller en god historie, men man får hurtigt fornemmelsen af, at de tekniske indikatorer bliver valgt og vraget af de to *traders*.

Javist, hvis man køber lige før *MACD*-linjen krydser *signal* linjen og søjlerne flipper grønt, så viser det sig at være gunstigt for nogle af de tidspunkter idet prisen stiger efterfølgende, men der er også tidspunkter hvor det måske ikke var en så god idé. Som f.eks. imellem *12:30* og *13:00*, hvor prisen gør et hop opad, og *MACD* som konsekvens akkurat krydser opad og man kan ane en helt lille grøn søjle. Den er kun grøn i *1* minut, hvorefter den bliver rød og *MACD* krydser nedad igen, da prisen igen falder. Havde man købt på det signal, så havde man været i underskud lige indtil omkring den globale (for billedet) top.

Vedr. *RSI*, så er dette over *65* fra ca. *12:30* til *12:45*, men prisen stiger alligevel. Havde man solgt så snart *RSI* var fortolket som overskøbt her, så havde man mistet muligheden for at sælge til en lidt højere pris ved slutningen af disse *15* minutter.

Hvis man tager teknisk analyse seriøst, så kan man også forestille sig, at da prisen blev afvist ved *100 SMA* lige efter det globale (i billedet) minimum, var dette et klart sælge-signal, men det vælger de to *traders* at se bort fra.

Man kan nok med rimelighed antage, at hvis en *trader* køber på alle de købssignaler der præsenterer sig, og sælger på alle salgssignaler, så vil hans portefølje hurtigt blive ædt op af transaktionsomkostninger.

Endvidere lader det dog til være svært at vide præcis hvornår man skal handle ud fra deres signaler. Bl.a. fordi der kan være modsigelser blandt de indikatorer man gør brug af.

Hvis det virkelig er muligt at handle ud fra teknisk analyse, så vil en maskine ihvertfald være i stand til at processere flere af disse samtidigt end et menneske, og muligvis skelne forskel imellem de falske og sande – hvis der altså virkelig er en forskel – og prædiktere herefter.

3 Data

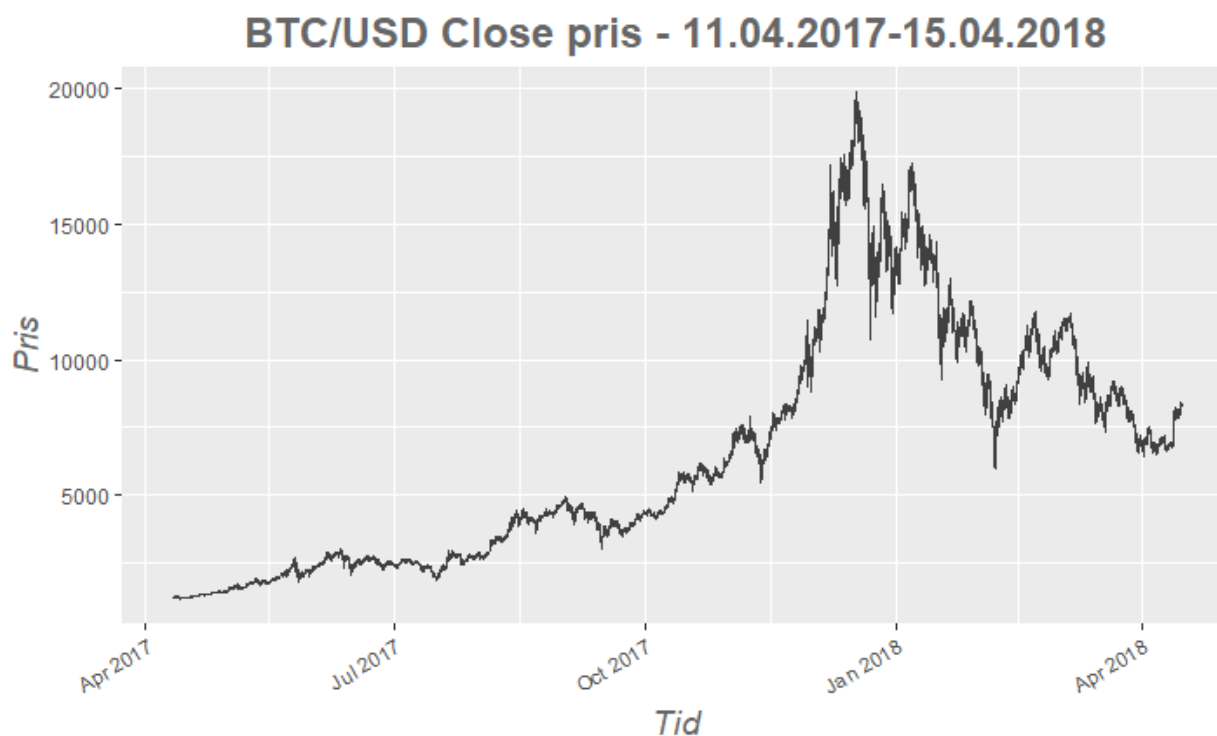
I dette kapitel ser vi nærmere på vores data.

Det indebærer bl.a. hvordan og hvor vi har indsamlet det, hvilke overvejelser har været nødvendige for at sikre at data er så brugbart som muligt, samt hvordan vi former vores respons og hvordan de forklarende variable kan beskrives.

For at opbygge en statistisk model med pris (eller en transformation af denne) som respons og tekniske indikatorer, hvoraf der findes mange forskellige, som *features*, kræves data af simpel struktur – nemlig i *Open-High-Low-Close-Volume (OHLCV)* format. Dette er i princippet en multivariat tidsrække, hvor hver observation er et bestemt tidsinterval, e.g. 1 minut. For hvert tidsinterval kan *OHLCV* data beskrives som

- *Open*: Prisen for den første eksekverede handel.
- *High*: Prisens højeste værdi.
- *Low*: Prisens laveste værdi.
- *Close*: Prisen for den sidste eksekverede handel.
- *Volume*: Den samlede mængde af *base* valuta, der er handlet for i tidsintervallet.

Den minutbaserede *Close* pris for det prisdata vi kommer til at arbejde med i denne afhandling, og som vi i resten af kapitlet kommer til at snakke mere om i detaljer, er illustreret nedenfor.



3.1 Tidsintervaller

Er man interesseret i at få et overblik over den historiske pris for en given kryptovaluta, så findes der efterhånden mange hjemmesider der tilbyder dette, bl.a. www.coinmarketcap.com. Data på disse sider kan *webscrapes* (i.e. trække prisinformation direkte fra *HTML*-koden) via diverse programmeringssprog, e.g. Python, og derefter struktureres i modelvenligt format til videre brug.

Den fælles ulempe for disse hjemmesider er dog, at de kun lagrer, og fremviser, data i høj tidsskala – typisk prisbevægelser fra dag til dag. Dermed kan man ved denne type data ikke modellere prisens opførsel *indenfor* hver dag, e.g. fra minut til minut, og ej hellere lave en handlingsstrategi der baserer sig på de lavere tidsinkremitter.

I virkeligheden er det måske netop data af lav tidsskala som man bør være interesseret i, hvis man bruger statistik og *software* som sine redskaber i sin *trading*. Argumentet er dobbelt:

- En maskine er i stand til at inddragede de minutbaserede tekniske prisindikatorer, og lave en prisprædiktions langt hurtigere end en menneskelig agent der bruger den samme information. For visse lave tidsskalaer kan man endda argumentere for, at en menneskelig agent ikke vil kunne lave sig et hurtigt nok overblik af indikatorerne, for derefter at lave en prædiktions med tilfredsstillende sikkerhed. Det kunne dermed tyde på, at en maskine har større chance for at slå en sådan agent på de lave tidsskalaer ifht. de højere af slagsen.
- Sammenhængende med det første, kan man muligvis konkludere, at da menneskelige agenter har svært ved at handle på de lave tidsskalaer, vil der være færre der handler på denne skala, og dermed er de statistiske arbitragemuligheder i langt mindre grad udnyttet på dette niveau.

En maskine har altså nogle mulige fordele på de lavere tidsskalaer, og det vil vi forsøge at udnytte.

Uanset hvilken tidsperiode man gerne vil bygge en eventuel model på, så kan man med fordel lægge ud med at indsamle *OHLCV* data af den laveste tidsskala, typisk 1 minut da det er nemt nok at lægge observationerne sammen, sådan at perioder på 5 minutter, 15 minutter, 1 time, osv. konstrueres.

3.2 Kort om API'er

For at få fat på *OHLCV* data af lav tidsskala er der kun en officiel vej: direkte digital kommunikation med en given børs der har de kryptovaluta markeder man er interesseret i. Denne digitale kommunikation foregår igennem et mellemlid, kaldet en *API* (*Application Program Interface*), og understøtter typisk meget anvendte programmeringssprog som *Python*, *Java* og *PHP*. En *API* udstedes typisk af enhver stor internetbaseret enhed (f.eks. har *Facebook*, *Twitter* og *Reddit* alle en *API*), sådan at enhedens brugere har en hurtig og sikker måde at f.eks. trække lagret information fra enhedens servere eller udføre diverse handlinger på sin private konto hos enheden – alt igennem en brugers udvalgte programmeringssprog.

Ved at bruge en *API* hos en børs er det desuden også muligt at opbygge en såkaldt *bot*, der automatisk handler ud fra nogle betingelser.

Vi vælger dog at eksekvere vores adaptive handlingsstrategi på en del af vores indsamlede data, istedet for at inddrage ovenstående *real-time* scenarie.

3.3 Scraping

Det data vi kommer til at beskæftige os med i denne afhandling er hentet fra kryptovalutabørsen *Bitfinex*.

Valget faldt på denne, fordi de fleste af dens markeder har en relativt høj volumen ifht. andre børser. Dette gælder i særdeleshed *BTC/USD* markedet, som også er det marked vi kommer til at lægge vores fokus på. Volumen på dette marked er så højt, at det af mange *traders* opfattes som *first-mover*, dvs. at hvis man kortsigtet handler *BTC/USD* på andre børser, så skal øjnene ofte først og fremmest være på *BTC/USD* på *Bitfinex*. Den generelt høje volumen medfører også, at pristidsrækkerne bliver mere sammenhængende, og dermed også – kunne man tro – forklarer markedsaktørernes psykologi mere nøje.

Forskellen er illustreret i nedenstående figur, der indeholder et tilfældigt snapshot af begge *BTC/USD* markeder på *Bitfinex* og kryptovalutabørsen *Kraken*, med 1-minut *candles*.

Bemærk hvordan prisgrafen for *Kraken* indeholder mange "huller" ifht. grafen for *Bitfinex*. Der er ikke tale om data der mangler, men istedet er det *candles* hvor prisen næsten ikke har flyttet sig den tilsvarende minut, eller hvor *Open* prisen er anten højere eller lavere end hhv. *High* og *Low* prisen i det foregående *candle*.

Et andet bemærkelsesværdigt element, er forskellen på volumen, illustreret nederst i figuren for hvert marked. Ved at se på den tilsvarende volumeskala i højre side, er det hurtigt at konkludere, at der i gennemsnit handles langt flere *BTC* på *Bitfinex* ifht. *Kraken*. For en aktør, der handler meget kortsigtet, eller med meget store summer, er en høj volume typisk meget



afgørende, idet det er langt nemmere at fylde sine ordrer her og nu. I den sammenhæng, er volumen på *Bitfinex* et stort plus for vores *trading* strategi, da ordrer har nemmere med at blive fyldt her.

Et andet argument for at *Bitfinex* er en god datakilde, er at den har en meget veludviklet digital infrastruktur. Software opdateringer er sjældent nødvendige og den har vist sig resistent imod øget press på infrastrukturen, typisk i form af pludselig høj volume på et eller flere af dens markeder.

Alt dette medfører, at børsen har en lav *down-time*, og dermed er der relativt få kronologiske "hul" i deres historiske data.

En så stor børs som *Bitfinex* har selvfølgelig en tilgængelig *API* der kan bruges til bl.a. data indsamling. Vi har valgt at benytte alt-mulig-manden blandt programmeringsprog, *Python*, i denne sammenhæng, og installeret et bruger-skrevet *module* (synonymt med en *package* i *R*), *ccxt*, der gør kommunikationen med deres *API* en smule mere overskuelig.

Helt specifikt så *scrape* vi 1-minut data (på baggrund af argumenterne i underkapitel 3.1), strækkende fra 21.11.2016 00:07:00 til 15.04.2018 23:59:00. Dette svarer til 735.833 observationer.

I vores *scrape* er vi opmærksomme på, at en del af data muligvis skal behandles før det er i brugbart *OHLCV* format. Det sker nemlig, at der slet ikke forekommer en handel for nogle af minutterne. Når disse observationer *scrapes* indeholder de derfor kun *NA* værdier i 4 de første af *OHLCV*-søjlerne, mens den sidste er lig 0 da volumen ikke er eksisterende den minut.

Det lader til at der er forholdsvis mange af disse *NA* værdier i den første del af data. Det er først lidt ind i 2017 at rigtig mange nye aktører er kommet ind på kryptovalutamarkedet, og fra da af har volumen været konsistent høj nok, til at antallet af *NA* værdier bliver ubetydeligt lille.

Siden vi har en forholdsvis høj mængde data, vælger vi at slette mange af de første observationer, sådan at det data vi kommer til at benytte videre i denne afhandling starter den 11.04.2017 00:00:00, hvilket svarer til et samlet antal observationer på 532800.

De *NA* minutter der er i data med det nye startpunkt, er korrigeret ved, at for hver minut m_t , der ikke handles, overfører vi *Close* prisen fra m_{t-1} , og indsætter i *OHLC* for m_t , mens vi beholder værdien 0 for *V*-søjlen.

3.4 Responsen

Ud fra vores overvejelser i underkapitel 3.1 vælger vi at arbejde på 1-minut skalaen.

Med data i *OHLCV* format er der mange mulige responser man kan oprette.

F.eks. kunne man lave en udgave med op til 4 serier af *OHLC*. Siden 1 minut er den laveste skala hvert *candle* kan gælde for, ville en multivariat respons som denne, hypotetisk set, kunne benyttes i en handlingsstrategi der kan handle ad flere omgange indenfor et minut. Hvis der prædikteres et forholdsvis lavt *Low* og tilsvarende højt *High*, kunne denne forskel udnyttes ved at forsøge at lægge en købs ordre lige over *Low* prædiktionen, og givet denne fyldes, sætte en salgsordre lige under *High* prædiktion. Man kan forestille sig forskellige andre kombinationer

imellem de 4 serier.

Den mest oplagde og simple, er dog at benytte den ene af disse 4 serier til vores respons.

I vores research af teknisk analyse, er det oftest *Close* prisen som *traders* først og fremmest ser på. De foretrækker ofte at vente med at udføre en handel indtil det nuværende *candle* lukker, og dens endelige *Close* pris registreres. I det samme sekund dette sker, fastlåses værdierne for de tekniske indikatorer for det *candle*, og disse har dermed en konstant værdi en *trader* kan fortolke og handle ud fra.

Vi ser ingen årsag til at fravige fra dette. Men om *Close* prisen i dens nuværen form direkte skal fittes som vores respons, er nok et bedre spørgsmål.

I denne afhandling er det direkte afkast vi først og fremmest er interesserede i at fitte som respons.

Men hvor mange *lags* skal denne respons så have? Den afgørelse kan umiddelbart virke lidt arbitrær, men for det *trading* setup der gælder i denne afhandling, kan man nok tænke sig frem til et nogenlunde fornuftigt *lag* for *Close* prisen.

Vi skal huske på, at når man handler i næsten ethvert marked, så er man pålagt at betale handlingsomkostninger. På kryptomarkedene ligger dette ofte mellem 0.05% og 0.2%, alt efter hvilke ordrer der placeres, hvilket vi selvfølgelig også må tage hensyn til. Det er værd at bemærke, at denne procent kan, med fordel, ofte sænkes forholdsvis meget, hvis man har handlet med stor volume over den seneste tid. På f.eks. *Bitfinex* kan man godt ende på 0% i *maker fee* hvis man har handlet for mere end 7.5 millioner *USD* over den seneste måned. Hvis vi lærer vores *trading* strategi at prædiktere med 1 *lag*, i.e. 1 minut frem, så vil der nok være mange tidspunkter, hvor det faktiske afkast fra $t-1$ til t , ikke er stort nok, til at mere end udligne handlingsomkostningerne. Men hvis der kigges længere frem end 1 minut, så er det meget realistisk at antage, at det forventede afkast kan være højere, og at omkostningerne dermed spiller mindre ind i den samlede profit.

Vi har valgt at konstruere vores respons som 15 minutters afkastet af *Close* prisen.

Dette har vi bl.a. udledt ud fra ovenstående overvejelser.

Med dette ser vores respons helt præcis altså således ud

$$R_t = \frac{Close_{t+15}}{Close_t} - 1$$

3.5 Features

Der er efterhånden et hav af tekniske prisindikatorer en motiveret *trader* kan trække på. Nogle af disse er designet af velkendte rutinerede *traders*, der handler ud fra teknisk analyse, som *Bollinger Bands* (udledt af John Bollinger i 80'erne), mens andre bare er velkendte ligninger indenfor matematikkens verden der er overført til brug i teknisk analyse, som et *Simple Moving Average*.

Typisk vil en *trader* ikke evaluere flere end en håndfuld indikatorer før han tager en handelsafgørelse. Det er jo begrænset hvor meget information en menneskelig aktør kan processere indenfor en bestemt tidsramme, hvilket har medført at nogle indikatorer er mere populære end andre.

I denne sektion vil vi gennemgå de typer af de forskellige tekniske prisindikatorer vi kommer til at gøre brug af som *features*, og forklare hvordan disse er opbygget, samt hvordan en *trader* kan finde på at fortolke på disse.

Listen inkluderer indikatorer der bruges af de allerfleste *traders*, men også nogle mere obskure af slagsen, der muligvis indeholder en forudsigelseskraft som er meget lidt udnyttet.

I det følgende sættes standardværdierne for deres input, typisk i form af *lags*, til at være lig de standardværdier der benyttes af *cryptowatch*. Hvis indikatoren ikke findes på denne hjemmeside, så er standardværdierne i *TTR* pakken i *R*, brugte.

- *SMA (Simple Moving Average)*
 - Er lig den gennemsnitlige værdi for *Close* prisen over et bestemt antal *lags*. Jo længere tilbage der kigges, jo mindre følsom er den overfor nylige prisbevægelser.
Benyttes bl.a. til at identificere den *laggede* trend, samt som udglatning af prisen og som en filtrering af volatiliteten. Endvidere benytter nogle *traders* en kort og en lang *SMA*, og handler ud fra hvornår de to krydser hinanden.
 - 1-dimensionel, med 15 som standard-*lag*.
- *EMA (Exponential Moving Average)*
 - En exponential variant af en *SMA* som navnet antyder. Mere følsom overfor den nuværende pris, dvs. $Close_{t-1}$ vægtes mere end $Close_{t-2}$, $Close_{t-2}$ vægtes mere end $Close_{t-3}$, osv.
Benyttes på samme måde som *SMA*. Desuden bruges de begge to også til at identificere potentiel modstand og støtte for prisen.
 - 1-dimensionel, med 10 som standard-*lag*.
- *RSI (Relative Strength Index)*

- En udglatning i form af gennemsnitlige tab eller afkast over et bestemt antal *lags* af *Close* prisen og er en af de mest populære momentum indikatorer. Ligger i $[0,100]$, og for $RSI < 35$ fortolkes aktivet som oversolgt, mens det er overkøbt for $RSI > 65$.
- 1-dimensional, med 14 som standard-lag.
- *StochRSI (Stochastic Relative Index)*
 - Er en indikator af en indikator, i den forstand, at den måler det nuværende *RSI* ifht. dets *min-max* værdier over et bestemt antal *lags*. Designet for at udlede flere overkøbt-oversolgt elementer, samt for at opnå en højere følsomhed end det traditionelle *RSI*.
 - 3-dimensional:
 - *nFastK*, med 14 som standard-lag.
 - *nFastD*, med 5 som standard-lag.
 - *nSlowD*, med 5 som standard-lag.
- *MACD (Moving Average Convergence/Divergence)*
 - En indikator der følger trenden, samt indikerer dens momentum. Udgjort af forskellen mellem en lang og en kort *EMA* af *Close* prisen. En udglatning af denne forskel, kendt som *signal-line*, konstrueres også og benyttes sammen med *MACD* serien. En *trader* vil bl.a. hyppigt se tidspunkterne når disse to krydser som handlingsmuligheder, hvor *MACD* serien krydser *signal-line* i en opadgående retning som en mulig købs indikation, mens der tales om en mulig salgsmulighed i det omvendte tilfælde.
 - 2-dimensional:
 - *MACD*, med hhv. 12 og 26 som standard-lags for den korte og den lange *EMA*.
 - *signal*, med 9 som standard-lag.
- *Aroon*
 - Designet til at udlede om et aktiv trender eller ej, og hvor stærk trenden er. Er en noget anderledes indikator idet den ikke indeholder *MAs*, men istedet 2 serier, *Up* og *Down*, der hhv. angiver hvor mange perioder siden aktivet havde et *n*-periode maksimum eller minimum. Indeholder endvidere en "forskels-serie" af disse to. Konverteres til et relativt mål mellem 0 og 100. I den forstand modellerer indikatoren ikke *Close* prisen ifht. tid, som de fleste andre indikatorer, men istedet tid ifht. *Close* pris. Serierne bruges bl.a. til at identificere nyligt opståede *trends*, samt konsoliderings- og korrektionsperioder for *Close* prisen.
 - 3-dimensional (med samme standard-lag på 25):

- *Aroon_up*
 - *Aroon_down*
 - *Aroon_oscillator*
- *ADX (Average Directional Index)*
 - Denne er også en *trend* indikator.
Dens byggesten er udgjort af *directional movements* der videre defineres som *positive* eller *negative*, alt efter hvordan 2 fortløbende *Low* priser forholder sig med de tilsvarende *High* priser. Fra disse konstrueres 2 udglatninger, *Plus Directional Index* og *Minus Directional Index*, der måler *trend-direction* over tid. Ud fra forskellen af disse 2 serier konstrueres endvidere et samlet *Directional Index*, samt en udglatning af denne, nemlig *Average Directional Index*, der repræsenterer styrken af *trenden*.
 - Sammen benyttes de til at identificere hvilken retning *trenden* har, samt dens styrke.
 - 4-dimensional (med samme standard-lag på 14):
 - *ADX_Dlp*
 - *ADX_Dln*
 - *ADX_DX*
 - *ADX*
- *BBands (Bollinger Bands)*
 - Er en volatilitets indikator der baseres på *SMA*s på $(H+L+C)/3$ værdien over en periode, samt standardafvigelsen over samme periode.
Ovenstående *SMA* udregnes, og 2σ (hvor 2 typisk er standard) lægges til og trækkes fra. Grafisk tales der om en udglatning af den typiske pris der har volatilitetsbånd over og under sig.
Endvidere konstrueres også en relativ-baseret serie, der måler prisen ifht. det øvre og nedre bånd.
 - Samlet set, benyttes den bl.a. til at identificere mulige, pludselige ændringer i volatiliteten, samt til at identificere såkaldte *M-tops* og *W-bottoms*.
 - 4-dimensional (med samme standard-lag på 25):
 - *BBands_dn*
 - *Bbands_up*
 - *Bbands_mavg*
 - *Bbands_pct*
- *TDI (Trend Detection Index)*

- En anden trend-baseret indikator der signalerer begyndelse og slutning for en trend.
Baseres på en række momentum summer, over n -lags, samt et multiplikationselement (typisk lig 2) og resulterer i en *trend-detection* serie og en *trend-direction* serie.
En positiv værdi for *trend-direction* signalerer en trend, mens en negativ værdi signalerer konsolidering. F.eks. vil en *trader* typisk fortolke på denne, ved at tage en position i den retning som aktivet trender, når indikatoren er positiv. Hvilken retning trenden har, fortolkes fra *trend-direction* serien, hvor en positiv (negativ) værdi signalerer en *up-trend* (*down-trend*).
- 2-dimensional (med samme standard-lag på 25):
 - *TDI*
 - *TDI_DI*
- *ROC (Rate-Of-Change)*
 - Den simpleste form for en momentum indikator.
Udgjort af en serie der repræsenterer den procentmæssige ændring af *Close* prisen fra sidste periode til nu.
 - 1-dimensional, med 1 som standard-lag
- *ROC-momentum*
 - I princippet den samme som ovenstående, dog som den faktiske ændring, istedet for den procentmæssige.
- *MFI (Money Flow Index)*
 - En indikator der gør brug af både typiske pris, udledt fra *HLC*, samt den tilhørende volumen. Udledes igennem flere skridt, men i princippet kan den ses som et forhold imellem positivt og negativt *money-flow*.
En anden beskrivelse for *MFI*, er som en *RSI* indikator der også tager højde for volumen.
 - Benyttes typisk til bl.a. at identificere vendinger i prisen, ved at se på fravigelser mellem *MFI* og prisen.
 - 1-dimensional, med 14 som standard-lag.
- *OBV (On Balance Volume)*
 - En indikator, der repræsenterer det løbende *total* for negativ og positiv volumen. For perioder hvor prisen stiger (falder) fra forrige periode, lægges (trækkes) volumen til (fra) *OBV* værdien fra forrige periode.
 - Anvendes typisk på samme vis som *MFI*, eller i tilfælde af at *OBV* følger retningen af prisen, som en bekræftelse for den nuværende trend.

- 1-dimensionel, med 21 som standard-lag.
- *SAR (Parabolic Stop-And-Reverse)*
 - En indikator, der følger prisens trend. Befinder sig under (over) prisen, hvis denne er i en *up-trend* (*down-trend*).
Benytter *High* (*Low*) pris indenfor *up-trenden* (*down-trenden*), sammen med en faktor, der forstærkes hver gang prisen får en ny *High* (*Low*) værdi indenfor *up-trenden* (*down-trenden*).
 - Fortolkes typisk af en *trader* til at identificere hvornår han bør lukke sin position, i.e. *Stop-And-Reverse*.
 - 1-dimensionel, med 0.02 som standard faktorværdi, og 0.2 som dens standard maks-værdi.

Vi henviser til hjemmesider dedikeret til teknisk prisanalyse, som f.eks. www.stockcharts.com, for et større indblik i hvordan de ovenstående indikatorer konstrueres og benyttes.

I kapitlet om *Feature Extraction* og *Feature Engineering* redegør vi for hvordan vi vælger at benytte flere former af ovennævnte tekniske indikatorer, samt hvordan vi vælger at transformere nogle af disse for at få dem stationære.

4 Teori og metode

I denne afhandling ønsker vi som nævnt at opbygge en række statistiske modeller, hvis formål er at prædiktere afkast for kryptovalutaen *Bitcoin* en given periode ud i fremtiden, og hvor prædiktionerne fra hver model derefter benyttes som led i hver sin adaptive *trading* strategi. Det betyder, at formålet ikke er at lave inferens (i.e. undersøge hvorfor og hvordan vores *features* hænger sammen med responsen) men istedet, at se hvorvidt det er muligt at udføre robuste prædiktioner af responsen og handle ud fra disse.

Der er et nærmest uendeligt antal måder at opbygge statistiske modeller på – både i form af det underliggende data men også i form af de mange teoretiske modeller der efterhånden findes, og endda hvordan flere af disse kan bruges sammen.

Netop den sidste af de ovenstående bemærkningner er essensen af *stacking*, som er en af de metoder vi kommer til at anvende.

I dette kapitel vil vi vise teorien bag de modeller, samt metoder, vi kommer til at beskæftige os med.

4.1 Evaluering

Vi kommer til at få brug for forskellige evalueringsmål når vi skal træffe afgørelser eller evaluere resultater. I dette underkapitel går vi igennem hvilket typer mål vi benytter videre i afhandlingen.

4.1.1 MAE-fejlmålet

Mean-Absolute-Error er et af de mest anvendte fejlmål når der arbejdes med en kontinuær respons.

Denne størrelse er defineret som

$$MAE = \frac{1}{n} \sum_{j=1}^n |y_j - \hat{y}_j|$$

Idet det er et fejlmål, og man via sine prædiktioner gerne vil ramme så godt som muligt, foretrækkes et lavere *MAE* ifht. et højere.

Dette mål bliver vores primære sammenligningsmål i vores statistiske modellering.

4.1.2 Benchmark modeller

I vores evaluering af de statistiske modeller, både i *Modellerings*-kapitlet, samt i kapitel 7, der drejer sig om *walk-forward-optimization*, vil vi være interesserede i hvordan de forskellige typer modeller klarer sig ifht. hinanden prædiktionsmæssigt. Samtidig kan det være nyttigt at sammenligne med modeller af simpel struktur, her kaldet *benchmark*-modeller. Det kan nemlig forekomme, at disse *benchmark* modeller er bedre til at prædiktere end mere avancerede modeller. Alle vores modeller vil derfor blive sammenlignet med to *benchmark* modeller, og hvis de ikke slår disse *benchmarks*, vil de ikke blive betragtet som gode modeller.

Zero-prediction

Den ene af vores to *benchmark* modeller kalder vi for *zero-prediction*, der som navnet også antyder, altid prædikterer at afkastet er lig 0 %, dvs. at prisen til tidspunkt $t + 15$ er den samme, som prisen til tidspunkt t .

Helt specifikt ser denne altså således ud:

$$Close_{t+15} = Close_t \leftrightarrow \hat{R}_t = \frac{Close_{t+15}}{Close_t} - 1 = 0$$

Denne metode har, måske undeligt nok, vist sig at virke godt i mange økonomiske og finansielle tidsserier (Rob J. Hyndman, <https://www.otexts.org/fpp/2/3>).

Mean-prediction

Den anden af vores to *benchmark* modeller kalder vi *mean-prediction*, som er udgjort af gennemsnittet af alle 15 minutters afkast over *train* perioden. Denne ser altså således ud:

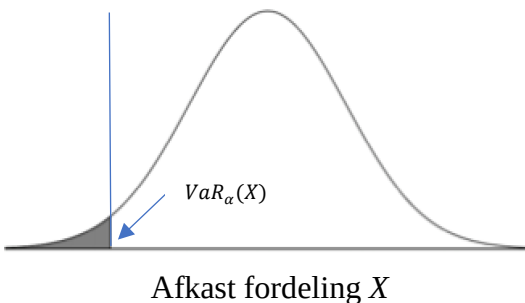
$$\hat{R}_t = \mu_{R_t}^{train}$$

4.1.3 Risiko

Når vi snakker om risiko mener vi “hvor galt kan det gå”, dvs. vi interesserer os kun for scenarier hvor vi lider store tab. Vi ser derfor på venstre hale i vores afkast fordeling.

Value-at-Risk (VaR)

VaR kan defineres ud fra porteføljens afkastfordeling eller tabsfordeling. Benyttes en



afkastfordeling, ses der blot på de udfald hvor afkast er lave og/eller negative.

Hvis man f.eks. har en afkastfordeling hvor hver observation er afkastet i % efter en lukket handel X , kan man sortere alle observationer fra den laveste til den højeste. Derefter kan $1 - \alpha$ % fraktilen af afkastfordelingen bestemmes. Dette tal ville være $VaR_\alpha(X)$.

Intuitivt:

- Med α % sandsynlighed vil vi ikke få et mindre afkast end VaR_α . α vil typisk være 95%, 99%, og i nogle tilfælde endda 99,9%.

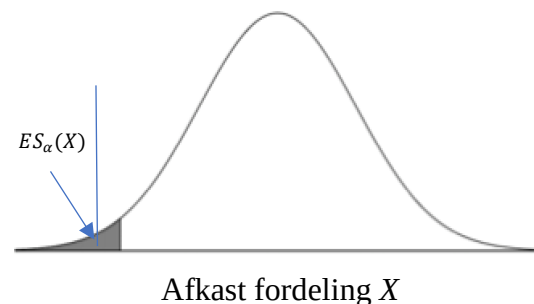
Expected-Shortfall (ES)

$ES_\alpha(X)$ kan defineres (igen, alt efter om det er en afkast- eller tabsfordeling) som forventningen af X , betinget med at man er i venstre hale af afkastfordelingen. Mere specifikt så angiver det hvad det gennemsnitlige afkast er for $1 - \alpha$ % af de værste udfald.

$$ES_\alpha(X) = E[X|X \leq VaR_\alpha(X)]$$

Intuitivt:

- Med et α på f.eks. 95%, angiver $ES_\alpha(X)$, hvad det forventede afkast er, givet man befinder sig i de 5% værste udfald.



4.2 Modeller

Vi vil i dette underkapitel redegøre for de typer af modeller vi kommer til benytte som led i vores adaptive *trading* strategi.

Dette inkluderer 4 forskellige typer – *Elastic Net*, de to træ-baserede *ensemble* modeller, *Random Forest* og *Gradient Boosting*, samt *Artificial Neural Network*.

Disse 4 typer kommer vi til benytte som *base-learners*, samt *meta-learners*, hvilket vi redegør for i det efterfølgende underkapitel.

Vi lægger ud med den simpleste af de 4 typer, nemlig *Elastic Net*.

4.2.1 Elastic Net

Elastic Net modellen, foreslået af Zou og Hastie i 2004, er en kombination af de nok mere velkendte modeller, *LASSO* (*Least-Absolute-Shrinkage-and-Selection-Operator*) og *Ridge-Regression*, der begge to er en videreudvikling af *Ordinary-Least-Squares* modellen.

Antag n antal observationer og p forklarende variable. Lad responsen være givet med $\mathbf{y} = (y_1, \dots, y_n)^T$ og $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_p)$ være matricen der indeholder de forklarende variable, hvor $\mathbf{x}_j = (x_{1j}, \dots, x_{nj})$. For en *OLS* kan prædiktionerne skrives op som

$$\hat{y} = \hat{\beta}_0 + \mathbf{x}_1 \hat{\beta}_1 + \dots + \mathbf{x}_p \hat{\beta}_p$$

hvor $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \dots, \hat{\beta}_p)$ findes som unik løsning til *Residual-Sum-of-Squares*-problemet

$$\hat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

OLS præsterer ofte forholdsvis dårligt som prædiktionsfunktion – især ved brug af mange variable da der kan opstå multikollinearitet, som kan øge variansen, og dermed også kan øge prædiktionsfejlen.

For at rette op på dette er først *Ridge-Regression* (Hoerl og Kennard, 1988) blevet introduceret, som typisk betegnes som at klare sig bedre prædiktionsmæssigt. Igennem et *bias-variance-tradeoff*, udfører denne udvidelse af *OLS* såkaldt *continuous shrinkage* (i.e. lægger en straf) på koefficienter ved at tilføje

$$\lambda |\boldsymbol{\beta}|^2 = \lambda \sum_{j=1}^p \beta_j^2$$

til minimeringsproblemet. Denne udvidelse kaldes også for *L2-regularisering*, og den medfører også at data skal være standardiseret (dette gælder desuden også for *L1-regularisering*). Man skal være påpasselig med værdierne for λ . En for høj værdi fører til potentiel *underfitting*, i.e. koefficienterne straffes i for høj grad og bliver for små, mens det omvendte kan føre til *overfitting*.

Ridge-regression fører ikke til en *parsimonious* (den simpleste, men mest forklarende) model, idet den kun reducerer koefficienterne, men stadig beholder dem.

Et alternativ til dette, udledt af Tibshirani (1996) er *LASSO*, der også udfører *continuous shrinkage* men samtidigt er i stand til at reducere antallet af koefficienter, og dermed kompleksiteten af modellen. Istedet for at lægge en skaleret sum af de kvadrerede koefficienter til minimeringsproblemet, benyttes

$$\lambda|\boldsymbol{\beta}|_1 = \lambda \sum_{j=1}^p |\beta_j|$$

Med tilføjelsen af denne "absolutværdi"-betingelse for koefficienterne til minimeringsproblemet, er modellen i stand til at reducere p ved at gruppere variable med en høj korrelation, benytte den ene af disse variable og slå de andre til 0.

En ulempe med dette, er at den lader til at være ligeglad med hvilken af disse den vælger at benytte.

Zou og Hastie påviser at denne metode kan være nyttig til *variable-selection*, men at *Ridge-regression*, for $n > p$ og hvor der er høj korrelation blandt variablene, empirisk har vist sig at klare sig bedre prædiktionsmæssigt end *LASSO*.

De foreslår derfor en løsning, der kombinerer *LASSO* med *Ridge-regression*, nemlig *Elastic-Net*. De lægger ud med at definere en *naive* version først, der tager højde for ulempen ved *LASSO*. Det lader dog til, at denne kun præsterer godt prædiktionsmæssigt, hvis den enten er tæt på at være en *Ridge-regression* eller en *LASSO*. For at korrigere for dette, skalerer de løsningen til den *naive* version, $\hat{\boldsymbol{\beta}}^{naive}$, med $(1 + \lambda_2)$.

Løsningen til det endelige minimeringsproblem, i forsimplet notation, kan skrives op som

$$\hat{\boldsymbol{\beta}}^{enet} = \arg \min_{\boldsymbol{\beta}} ((\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) + \lambda_2 |\boldsymbol{\beta}|^2 + \lambda_1 |\boldsymbol{\beta}|_1)$$

4.2.2 Træ-baserede Ensembles

De såkaldte *ensemble* metoder, som *Random Forest* og *Gradient Boosting Machine* er en del af, bruger en "del-og-hersk" tilgang idet de opsplitter optimeringsproblemet i en række *weak*

learners, der derefter kombineres til en *strong learner*. For at forstå dette i sammenhæng med vores valgte *ensemble* modelvalg, *Random Forest* og *Gradient Boosting Machine*, kræver det et kig på hvad et *decision tree* er.

Decision Trees

Decision trees kan benyttes til både regression og klassifikation, og har – ikke overraskende – en træstruktur.

Med en række *features* og tilhørende respons, opbygger algoritmen automatisk træet fra rod til top ved enten at skære numeriske *features*, eller hvis der er tale om faktorer, opdele ud fra de tilhørende niveauer ud fra et minimerings kriterie (f.eks. *Sum-of-Squares* for *regression decision trees*). Intuitivt fører disse *splits* til at data bliver delt i mindre og mindre regioner, sådan at observationerne i hver endelig region indeholder fælles, eller lignende, *feature*-værdier med tilhørende responsværdier. På denne måde kan afhængigheden imellem respons og *features* læres.

Idet vi beskæftiger os med en numerisk respons, viser vi fremgangsmåden for dyrkelsen af *regression decision trees*.

Givet observationerne, (x_i, y_i) for $i = 1, 2, \dots, N$, hvor $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})$, drejer det sig om at dele det p -dimensionale *feature* rum op i M antal regioner, $R_1, R_2, \dots, R_M$, hvor den tilhørende responsværdi defineres som konstanten c_m . For hver region har vi prædiktionsfunktionen

$$f(x) = \sum_{m=1}^M c_m I(x \in R_m)$$

Antages videre at kriteriet for minimeringsproblemet er den velkendte *Sum-of-Squares*, $\sum (y_i - f(x_i))^2$, er det bedste fit givet som middelværdien af alle y_i tilhørende R_m , altså

$$\hat{c}_m = gns(y_i | x_i \in R_m) \quad (4.1)$$

For at finde frem til den bedste opdeling af data i regioner, skal algoritmen afgøre hvilket binært split, s , variabel j skal have.

Startende med det fulde data, defineres der 2 plan der opdeler *feature* rummet således:

$$R_1(j, s) = \{X | X_j \leq s\} \text{ og } R_2(j, s) = \{X | X_j > s\}$$

Optimeringsproblemet kan nu opskrives som et indre og ydre minimeringsproblem

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

Fra (4.1) har vi, at løsningen til det indre minimeringsproblem, for ethvert j, s , er givet ved

$$\hat{c}_1 = gns(y_i | x_i \in R_1(j, s)) \text{ og } \hat{c}_2 = gns(y_i | x_i \in R_2(j, s)),$$

mens det ydre løses ved at finde den bedste kombination af variabel j og split s .

Efter at have fundet det optimale par (j, s) opdeles det fulde data i 2 regioner ud fra løsningen, og opsplitningsprocessen gentages på disse to og fortsætter på denne måde på alle resulterende regioner.

Opsplitningsprocessen fører til en træstruktur, der indeholder et bestemt antal *nodes*.

Den allerførste node i træet, hvor data stadig er urørt, kaldes typisk *root node*, efterfuldt af et bestemt antal *decision nodes*, mens de allersidste, indeholdende responsværdien for den tilsvarende region, kaldes *leaf* eller *terminal nodes*.

Hvis man dyrker træet fuldt ud, er der stor chance for at træet *overfitter* og dermed præsterer dårligt på uset data (forholdsvis høj *generalization-* eller *out-of-sample-error*), mens et for lille træ ikke har opdelt data grundigt nok til at lære sammenhængen mellem respons og *features* (i.e. det *underfitter*).

Størrelsen af træet er altså en parameter der skal optimeres for, da en optimal størrelse vil variere alt efter hvilket data der arbejdes med. Dette udføres typisk ved at dyrke et fuldt træ først, T_0 , og derefter rekursivt lægge regioner sammen ved at "slette" den *decision node* der har skabt de tilsvarende regioner. Der ledes altså efter et optimalt *subtree*, T_a af T_0 .

Idet denne *pruning* ikke forekommer i de to træ-baserede *ensemble* metoder vi kommer til at benytte, går vi ikke længere i dybden med *cost-complexity-pruning*, som denne metode også kaldes, men referer istedet til EoSL s. 308 for et detaljeret overblik af metoden.

Empirisk set har *decision trees* vist sig at være fordelagtige at bruge som prædiktionsværktøj. Selv om de er teoretisk simple og hurtige at træne, er de nemlig overraskende kraftfulde, men som indikeret tidligere kan de nemt *underfitte* i tilfælde af for små træer, og *overfitte* hvis de er for store.

For at benytte *decision trees* i praksis er det derfor ofte nødvendigt at løfte algoritmen til et højere niveau vha. *ensemble* metoderne, *boosting* og *bagging*, der netop tager hensyn til hhv. *under-* og *overfitting*, og *overfitting*.

Vi tager hul på den nok mest brugte *bagging* metode når den underliggende prædiktionsfunktion er et *decision tree*, nemlig *Random Forest*.

Random Forest

Som sagt så kan enkelte *decision trees* være kraftfulde, men hvis de dyrkes for meget vil de have en ufavorabel *generalization-error*. Breiman (2001) foreslår en løsning til dette via en *bagging* tilgang.

En simpel måde at beskrive en *Random Forest* på er som en gennemsnitsbaseret prædiktionsfunktion, der er udgjort af en samling af uafhængige *decision trees*, og hvis prædiktion er middelværdien af alle de enkelte *decision trees* prædiktioner.

Breiman viser at der findes en øvre grænse for *generalization-error* for *Random Forests*.

I forlængelse indkorporerer *Random Forest* også at *features* tilfældigt vælges som kandidater til et *split*. Hvor mange vælges tilfældigt kan knyttes direkte til *generalization error* – udvælges for mange, øges korrelationen mellem træerne, og *generalization error* for den samlede skov øges, og vælges for få, så er *generalization error* forholdsvis høj for hver enkelt træ, hvilket medfører at denne også bliver høj for hele skoven.

Det faktum at *Random Forest* producerer uafhængige træer gør at hvert træ kan dyrkes hver for sig, og dette er derfor i høj grad en paralleliserbar proces, hvilket gør at algoritmen skalerer godt med computerkraft.

Random Forest algoritmen er som følger (EoSL s.588, 2008):

1. For $b = 1$ til B
 - a. Træk en *bootstrap sample* Z^* af størrelse N fra *train data*, hvor $N = \text{antal obs. i train}$.
 - b. Fit et træ, T_b , til den trukne *sample* ved rekursivt at gentage følgende trin for hver *terminal node* i træet, indtil det mindste *node size* kriterie, n_{min} , er opnået.
 - i. Træk m tilfældige *features* fra p mulige.
 - ii. Vælg den *feature* der medfører det bedste *split*.
 - iii. *Split* den pågældende *node* i 2 *node-children*.
2. Udskriv hele samlingen af træer, $\{T_b\}_1^B$.

For at lave en regressionsbaseret prædiktion for en ny observation, x , har vi slutteligt

$$\hat{f}_{rf}^B = \frac{1}{B} \sum_{b=1}^B T_b(x).$$

Det er især parametrene der styrer antallet af træer, dybden af hvert træ, samt antallet af tilfældigt trukne *features* ved hvert niveau, der er interessante, da disse ofte har en forholdsvis høj indflydelse på *generalization error*. Man skal altså være opmærksom på at finde en nogenlunde balance mellem disse når modellen skal optimeres.

Gradient Boosting Machine

Ifht. *Random Forest*, benytter *Gradient Boosting Machine* – som navnet også antyder – en *boosting* tilgang, hvor træningsdata iterativt vægtes, alt efter hvordan det forrige træ præsterer. Denne vægtning evalueres ud fra observationernes indflydelse på tabsfunktionen, i.e. residualer.

Slutteligt opnås en samling af iterativt korrigerede træer, der er en *GBM*.

Formen for træerne i *GBM* fraviger også typisk en del fra *Random Forest*. Ligesom for *Random Forest*, så foregår der ikke *pruning* af træerne, men disse vælges til samme, konstante størrelse (dybde).

Gradient Boosting algoritmen, også kendt som *MART* i tilfælde af regression, er som følger (EoSL, s.361):

1. Initialiser $f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$.

2. For $m = 1$ til M :

a. For $i = 1, 2, \dots, N$ udregn

$$r_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}.$$

b. Fit et *regression decision tree* til de negative gradient værdier, r_{mi} , givet endelige regioner

$$R_{jm}, \quad j = 1, 2, \dots, J_m.$$

c. For $j = 1, 2, \dots, J_m$ udregn

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

d. Opdater $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.

3. Udskriv $\hat{f}(x) = f_M(x)$.

Denne generelle form for algoritmen kan ændres, alt efter hvilken tabsfunktion benyttes. For regression er disse typisk *least-squares*, et der baseres på fortegnet for residualerne, mens *Huber* metoden er en blanding mellem disse to.

MART algoritmen kan modificeres, ved at tilføje faktoren, ν , også kaldet læringsraten, til korrektionsleddet i 2.d-skridtet. Denne parameter styrer altså korrektionsniveauet, fra træ til træ i den iterative process, og den er dermed nært tilknyttet parameteren M (antal træer). Sænkes ν skal M typisk øges, for at opnå den samme *generalization error*. Men ved højere M øges også tidsomkostningerne.

Der skal altså typisk findes en balance mellem disse to der kan være forholdsvis indflydelsesrig.

Da en sluttelig *GBM* opnås iterativt (træerne er afhængige) har denne typisk ikke samme paralleliseringsfordel, som *Random Forest* har. Det er kun i træbyggelsesskridtet, 2.b, at der er en paralleliseringsmulighed.

4.2.3 Artificial Neural Network

Den centrale idé med *Artificial Neural Network* er at bruge lineære kombinationer af inputs som udledte *features* og derefter modellere responsen som en ulineær funktion af disse nye *features*. Resultatet er en kraftfuld læringsmetode, der benyttes i et vidt spænd af applikationer i mange fagområder.

En *single hidden layer perceptron*, som er et *Artificial Neural Network* med ét *hidden layer*, er en *two-stage* regressions- eller klassifikationsmodel som tit bliver repræsenteret som et netværks-diagram (se første billede på næste side for en illustration).

X er en input vektor med p komponenter og Y_k , $k = 1, \dots, K$ er responsen.

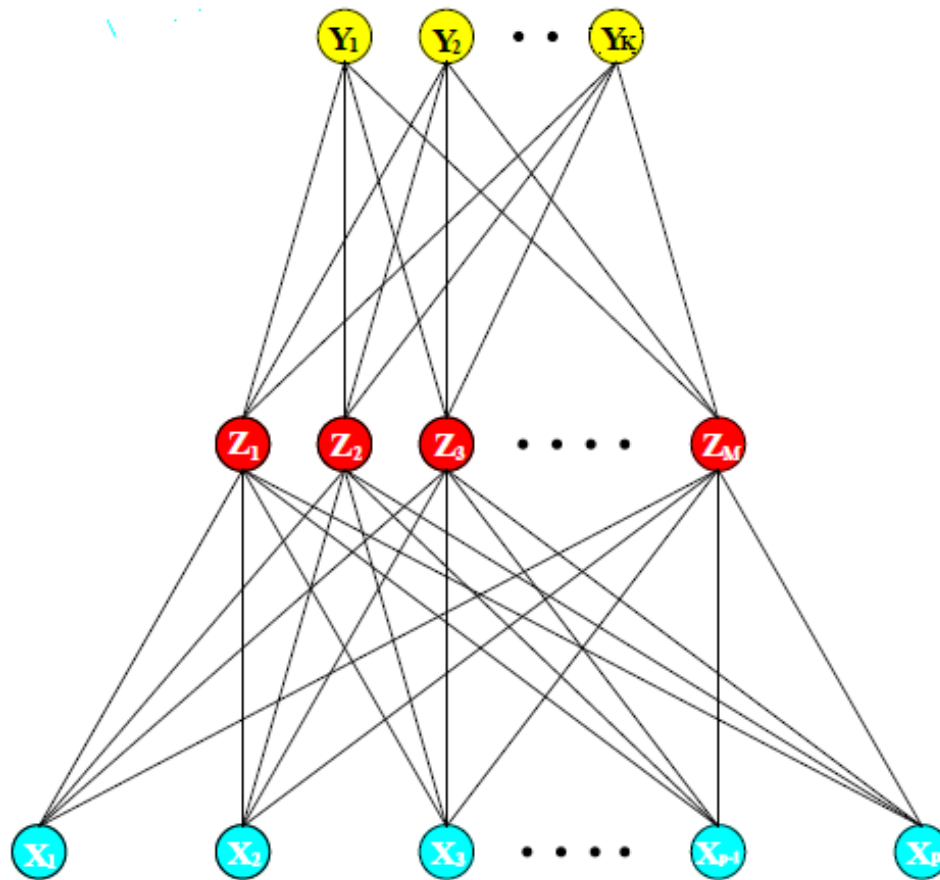
For regression er K typisk lig med 1 og der er kun et output element i toppen, altså en én dimensionel responsvariabel. Vi vil forklare den i det generelle tilfælde, nemlig den der omfatter både *K-class* klassifikation og regression. De udledte variable Z_m er konstrueret fra lineære kombinationer af inputs, og responsen Y_k er modelleret som funktion af lineære kombinationer af Z_m via. (formel 11.5, EoSL s.392):

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, \dots, M$$

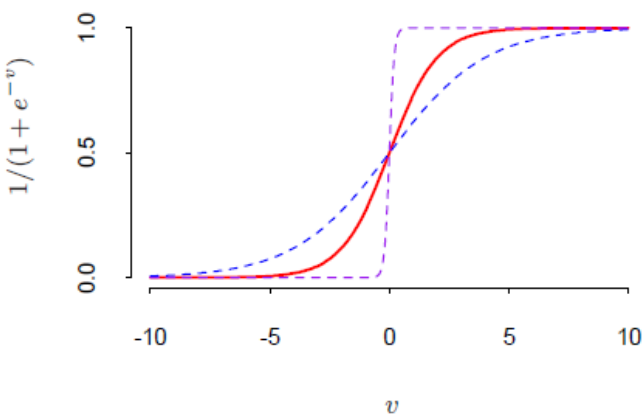
$$T_k = \beta_{0k} + \beta_k^T Z, k = 1, \dots, K$$

$$f_k(X) = g_k(T), k = 1, \dots, K$$

hvor $Z = (Z_1, Z_2, \dots, Z_M)$, og $T = (T_1, T_2, \dots, T_K)$.



Aktiveringsfunktionen $\sigma(v)$ er sædvanligvis valgt at være *sigmoid* funktionen $\sigma(v) = \frac{1}{1+e^{-v}}$ men der er mange andre funktioner som kan benyttes istedet for denne (se nedenstående figur for en illustration af denne).



Output funktionen $g_k(T)$ tillader en sidste transformation af vektoren af outputs T . For regression bruger man typisk identitetsfunktionen $g_k(T) = T_k$. Enhederne i midten af netværket, som udregner de afledte *features*, Z_k , kaldes *hidden units*, fordi værdien af Z_m ikke er direkte observeret. Der kan sagtens være mere end et *layer* af *hidden units*, selvom der kun er ét *layer* på figuren for netværket. Vi kan tænke på Z_m som

en basis ekspansion af det originale input X .

Navnet *Neural Network* stammer fra det faktum, at det blev udviklet til at modellere menneskets hjerne. Hver neuron-enhed repræsenterer en neuron og forbindelserne repræsenterer synapser i hjernen.

Artificial Neural Networks har ukendte parametre, som ofte kaldes vægte, og man prøver at finde værdier for disse, som gør at modellen passer godt til *train* data. Det skal bemærkes, at antallet af vægte er:

$$\{\alpha_{0,m}, \alpha_m; m = 1, 2, \dots, M\} \quad M(p + 1) \text{ vægte,}$$

$$\{\beta_{0,k}, \beta_k; k = 1, 2, \dots, K\} \quad K(M + 1) \text{ vægte.}$$

For regression bruges typisk *Sum-of-Squares* for at måle hvor godt modellen fitter på *train* data:

$$R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2$$

Vi vil typisk ikke finde det globale minimum af $R(\theta)$, da det højst sandsynligt vil være et overfit. Nogen regularisering er nødvendig og det bliver gjort igennem et *penalty* term, eller indirekte igennem *early stopping*.

Måden hvorpå $R(\theta)$ bliver minimeret sker ved *gradient descent*, kaldet *back-propagation* i denne sammenhæng. Pga. opbygningen af modellen kan gradienten udledes ved kæde reglen. Gradienten kan udregnes ved et *forward* og et *backward sweep* over netværket, hvor der kun fokuseres på størrelser i hver enkelt neuron. En *gradient descent* opdatering i $(r + 1)$ 'te iteration har formen:

$$\beta_{km}^{(r+1)} = \beta_{km}^r - \gamma_r \sum_{i=1}^N \frac{\partial R_i}{\partial \beta_{km}^r},$$

$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^r - \gamma_r \sum_{i=1}^N \frac{\partial R_i}{\partial \alpha_{ml}^r},$$

formel (11.13) hvor γ_r er lærings raten, som siger noget om hvor meget vægtene skal ændres.

Man kan opskrive "errors" som (formel 11.14):

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi},$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}$$

hvor δ_{ki} og s_{mi} er "errors" fra den nuværende model i henholdsvis *output* og *hidden layer units*. Disse tilfredstiller ligningen:

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^K \beta_{km} \delta_{ki}, \quad (11.15)$$

som kaldes *back-propagation* ligningen. Ved at bruge denne, kan opdateringerne af gradienten udføres i to skridt for hver iteration. I *forward* skridtet er de nuværende vægte faste og responsen $\hat{f}_k(x_i)$ bliver udregnet ud fra formelen (11.5). I *backward* skridtet bliver *errors* δ_{ki} udregnede, og derefter *back-propagated* via (11.15) for at give *errors* s_{mi} . Begge sæt af *errors* er derefter brugt til at udregne gradienten for opdateringerne i (11.13) via (11.14).

En af de mest basale parametre er antallet af *hidden units* og *hidden layers*. Det er ofte nødvendigt at eksperimentere sig frem til passende størrelser. Disse er direkte forbundet med antallet af vægte, og det anses ofte som gavnligt at have forholdsvis mange af disse. Dog er det også forholdsvis nemt at overfitte med et *Artificial Neural Network* med for mange vægte. Dette kan man tage højde for med bl.a. via noget der minder om *L2-regularisering* eller ved *early stopping*, der evalueres ud fra andet data end *train*.

4.3 Stacking

En lille bemærkning omkring betegnelser der bliver brugt i dette underkapitel:

- I en *stacking* sammenhæng er en *base-learner* enhver prædiktiv algoritme. Den eneste forskel på en *base-learner* og en *meta-learner* er deres features.

Stacking, eller *stacked generalisation*, er en procedure indenfor *ensemble* metoder, hvor en *second-level learner*, som i vores afhandling kaldes en *meta-learner*, trænes med prædiktioner fra en samling af *base-learners* som *features* (processen uddybes nedenfor). Prædiktionerne fra *base-learners* genereres ved brug af krydsvalidering. I den originale *stacking* literatur foreslog Wolpert (1992) at bruge *leave-one-out* krydsvalidering til at konstruere *features* til *meta-learners*, men pga. beregningsomkostninger bliver man i praksis tit nødt til at bruge *k-fold* krydsvalidering.

Det følgende er en mere grundig beskrivelse af hvordan disse *features* konstrueres.

Antag at det originale træningsdata består af n uafhængige og identisk fordelte observationer $\{O_1, \dots, O_n\}$, hvor $O_i = (X_i, Y_i)$, og $X_i \in R^p$ er en vektor af *feature* værdier og hvor $Y_i \in R$ er den tilsvarende respons værdi. Betragt et *ensemble* konstrueret af et sæt af L *base-learners*, $\{\psi^1, \dots, \psi^L\}$, hvor hver *base-learner* er indekseret i en algoritme klasse med et specifikt sæt af parametre. Så vil processen af at konstruere *features* være, at lave en $n \times L$ matrice, Z , bestående af *k-fold* krydsvaliderede prædiktionsværdier som vist nedenfor:

1. Det originale træningsdata O bliver delt tilfældigt i k omtrentligt lige store grupper, $O_1, \dots, O_k$.

2. For hver *base-learner*, ψ^l , er k -fold krydsvalidering brugt, til at konstruere n krydsvaliderede prædiktioner. Disse n -dimensionelle vektorer af krydsvaliderede prædiktioner bliver de L kolonner i Z matricen.

Z bliver sammen med den originale responsvariablel, $(Y_1, \dots, Y_n) \in R^n$, brugt til at træne vores *meta-learner*.

Før *meta-learner* kan prædiktere på nyt data, må de L *base-learners* først prædiktere på dette. Derefter bruges disse prædiktioner som *features* i *meta-learner*, der nu har muligheden for at udføre den endelige prædiktion på det nye data.

4.4 Walk-forward-optimization metoden

Vi vil i dette underkapitel give en kort forklaring på, hvad *walk-forward-optimization* er for noget, hvilke konsekvenser det har for måden vi opdeler vores data på, samt hvordan vi vælger at ændre den typiske måde at lede efter optimale hyperparameterværdier på.

I sidste ende medfører det at der i stedet snakkes om passende *intervaller* for disse hyperparametre, istedet for optimale værdier.

Et af de primære formål med denne afhandling, er at udvikle en adaptiv *trading* strategi, baseret på bl.a. prædiktioner fra statistiske modeller, men også nogle *trading*-signaler. Disse signaler kommer vi til at redegøre for i kapitel 7, hvor vi udfører vores *walk-forward-optimization*.

Strategien skal handle ud fra det data vi har indsamlet, hvor både modellernes hyperparametre, betinget med deres passende intervaller, samt *trading*-signaler skal optimeres.

Generelt kan det forventes at det er bedre at validere en *trading* strategis resultater over en længere periode ifht. en kortere, men hvis denne valideringsperiode er lang er det tænkeligt, at de optimale værdier man har fundet for strategien, bliver mindre optimale jo længere ind i denne valideringsperiode strategien bevæger sig. Bl.a. fordi det underliggende data muligvis har ændret sig fra den periode man har udledt de optimale værdier fra.

Det er som et svar på dette at *walk-forward-optimization* skal forstås.

Lad os først gå igennem hvad denne metode indebærer vha. et eksempel, hvor *in-sample* data her skal forstås som *train* og *test* mens *validation* er *out-of-sample* data.

Givet man har data, der løber over 1 år, kunne den umiddelbare tanke være, at optimere ens strategi over de første 8 måneder (*in-sample* data), og derefter validere strategien på de sidste 4 (*out-of-sample* data).

Istedet for dette kan man med *walk-forward-optimization* optimere strategien på f.eks. de første 8 måneder og validere på den 9. måned, sådan at man kun har brugt en del af det

samlede data man har til rådighed – et såkaldt vindue, om man vil. Efter denne initiale kørsel, rykkes vinduet med længden af *out-of-sample* datasættet, og der optimeres fra den 2. til og med 9. måned og valideres på den 10. måned.

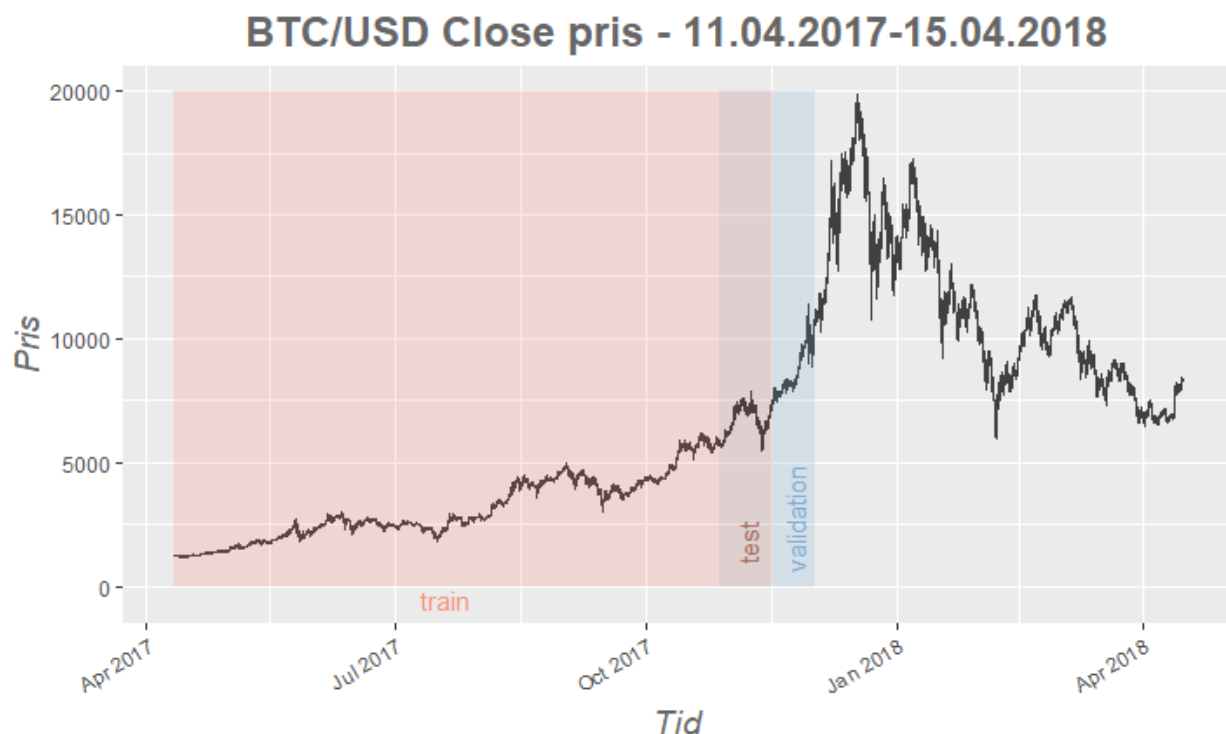
Vinduet rykkes på denne måde en måned frem for hver iteration, indtil vi har valideret på den 12. og sidste månede.

Som nævnt tidligere, kan fordelene ved at benytte denne metode, der dækker flere vinduer være, at den valgte strategi ikke baseres på forældet data der muligvis ikke er repræsentativt for nuværende markedstendenser. Resultatet er en strategi der tillades at ændre sig over tid for at tage hensyn til mulige overordnede markedsændringer, der forhåbentligt fører til et højere samlet afkast for strategien.

For at en metode der ligner ovenstående eksempel skal passe ind i vores adaptive *trading* strategi, er vi interesserede i at specificere *intervaller* for hyperparametrene, sådan hyperparametrene får mulighed for automatisk at tilpasse sig fra vindue til vindue, uden vores indblanding (hvorfor vi kalder den en adaptiv *trading* strategi).

For at finde frem til disse intervaller vælger vi i første omgang at lægge en stor del af data til side, og basere vores søgning af passende intervaller for hyperparametrene på det data der omfatter *train* og *test*, som vist i figuren nedenfor. Denne søgning udføres i kapitel 6, hvor modellerne altså kommer til at blive fittet på data der ligger i *train* (200 dage, dvs. 287.985 obs. og ikke 288.000, pga. responsens *lag*), hvorefter passende hyperparameterintervaller ud fra modellernes *generalization error* på *test* (20 dage, 28.785 obs.) udvælges til brug i vores *walk-forward-optimization*.

Det kan nævnes, at disse 2 perioder sammen med *validation* (15 dage, 21.585 obs.), kommer til



at være det første vindue i vores *walk-forward-optimization* i kapitel 7. For at referere til eksemplet ovenover, så kommer *train* og *test*, i det kapitel, til at være vores *in-sample* data, mens *validation* kommer til at være vores *out-of-sample* data.

5 Feature extraction og feature engineering

Feature extraction

Ud fra de tekniske indikatorer vi talte om i underkapitel 3.5 konstruerer vi vores *features*. Disse kan alle konstrueres ud fra vores *OHLCV* data. Til dette formål laver vi en funktion, som vi kalder *TechIndGenerator1*, der ud fra *OHLCV* data og *feature* specifikke antal *lags*, kan udregne alle de tekniske indikatorer vi gør brug af.

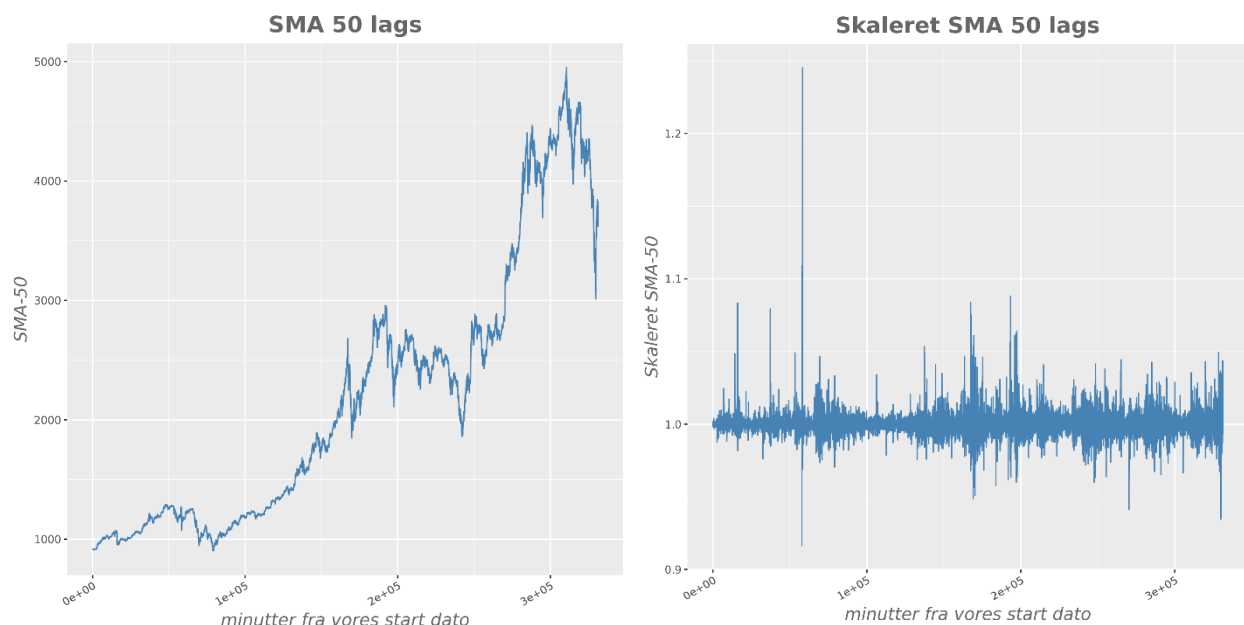
Vi vælger, som sagt i underkapitel 3.5 at bruge tekniske indikatorer med standardværdierne på *cryptowat.ch*, og hvis en teknisk indikator ikke findes på hjemmesiden bruger vi standardværdierne i *TTR* pakken i R. Samt disse standardværdier af *lags* vælger vi at basere nogle af vores *features* på tekniske indikatorer med korte *lags* og nogle med lange *lags*, der muligvis kan indeholde information om hhv. den kortsigtede- og mere langsigtet trend i *Close* prisen. Samt dette kan vi muligvis opnå yderligere forklaringsgrad ved at benytte differencede værdier af vores tekniske indikatorer. Vi beslutter os for at difference alle vores tekniske indikatorer med 1 og 5 *lags*, for at udvide vores *feature* rum til også at indeholde hældningen fra den sidste periode samt 5. sidste.

Resultatet er, at vi opnår 123 *features* der benyttes i vores modellering.

Feature engineering

Da mange af vores *features*, som forventet, viser sig at være ustationære, vælger vi at skalere dem for at rette op på dette. Et eksempel på en teknisk indikator, som ikke så uventet, er ustationær er *SMA-50* af *Close* prisen.

På næste side ses et udklip af *SMA-50* af *Close* prisen til venstre, og til højre et udklip af den skalerede *SMA-50* af *Close* prisen. For at få den stationær skaleres den altså med *Close* prisen.



De tekniske indikatorer vi skalerer med *Close* prisen er altså alle vores varianter af:

- *SMA (Simple Moving Average)*
- *EMA (Exponential Moving Average)*
- *BBands (Bollinger Bands)*
- *SAR (Parabolic Stop-And-Rverse)*

Dette inkluderer både differencede og ikke-differencede.

En af de tekniske indikatorer, som vi skalerer noget anderledes end disse ovenover er:

- *OBV (On Balance Volume)*

Denne valgte vi at skalere således:

$$\text{SkaleretOBV} = \frac{OBV - \text{SMA}_{21}(OBV)}{\text{SMA}_{21}(\text{VOLUME}(\text{BTC}))}$$

Bemærk hvordan en *SMA-21* indgår i denne skalering, da *OBV* af *traders* fortolkes som et forhold til denne *SMA-21*.

Det er endvidere nødvendigt at standardisere alle vores *features*, da regulariseringsparametrene i *Elastic Net* samt *Artificial Neural Network* vil "straffe" koefficienter. F.eks. vil regulariseringen "straffe" koefficienten for en *feature* med stor størrelsesorden uretfærdigt i forhold til koefficienten for en *feature* med lille størrelsesorden. I modellerne *Random Forest* og *Gradient Boosting Machine* er standardisering urelevant, da disse modeller er invariante overfor monotone transformationer.

Standardiseringen bliver udført ved

$$Z_i = \frac{X_i - \mu}{\sigma}$$

hvor, Z_i er den standardiserede *feature*, X_i er den original *feature*, μ er gennemsnittet af *feature* og σ er standardafvigelsen.

Vi vælger at bruge vores *in sample* data (fra vores *walk-forward-optimization*) *train* og *test* til at standardisere ud fra. Dvs. at for hvert vindue bliver både *train*, *test* og *validation* standardiseret ud fra *train* og *test*. Dette vælger vi at gøre, da vi i den virkelige verden ikke vil have nogen information om det usete data – nemlig *validation*.

6 Modellering

Det overordnede formål med dette kapitel, er at finde passende hyperparameterintervaller for de 4 typer af modeller – *Elastic Net*, *Random Forest*, *Gradient Boosting Machine* samt *Artificial Neural Network* – som vi kommer til at anvende, både som *base-learners* og *meta-learners* til videre brug i vores *walk-forward-optimization* i kapitel 7.

Vi kommer altså til at udlede 8 forskellige samlinger af hyperparameterintervaller – 4 samlinger for de 4 *base-learners* og 4 samlinger for de 4 *meta-learners*.

I første afsnit af dette kapitel gør vi os klogere på hvordan de 4 *base-learners* ser ud fittet på vores data. Her kigger vi lidt nærmere på nogle af de mere betydningsfulde hyperparametre, som vi skal være opmærksomme på i vores søgning efter passende hyperparameterintervaller for *base-* og *meta-learners*. De sammenhænge vi observerer her, vil nemlig generelt også gælde for vores *meta-learners*, idet den eneste forskel mellem disse to er hvilket *feature* rum de har.

I næste afsnit går vi ned i detaljer omkring hvilken fremgangsmåde vi har valgt at bruge i vores *grid-search* i søgningen efter passende hyperparameterintervaller, sådan at det passer ind i rammen for en adaptiv *trading* strategi.

For dette scenarie er der nemlig muligvis en række problematikker med at udføre et typisk *grid-search*, og stille sig tilfreds med nogle faste optimale værdier for hyperparametrene.

For at lave et mere passende *grid-search* kræves der overvejelser omkring mængden af data vi arbejder med og den øgede tidsomkostning det fører med sig, hvordan dimensionen af hyperparameter rummet i hver model (gør sig især gældende for *RF*, *GBM* og *ANN*) hænger sammen med faren for at ende i lokale minima og især hvordan vi vælger at udvide den noget mere sædvanlige måde at udføre et *grid-search* på, sådan at vi tager hensyn til at de optimale værdier for hyperparametrene kan ændre sig en del hver gang der skal trænes og prædikteres for et nyt vindue.

Vi viser kun udledningen af hyperparameterintervallerne for to af vores *base-learners*, da denne process hurtigt kan blive noget kedelig. De fundne hyperparameterintervaller for de resterende to *base-learners* præsenteres dog.

I det efterfølgende og sidste underkapitel præsenteres de udledte hyperparameterintavaller for vores 4 *meta-learners*, hvor processen dog udelades (se ovenstående argument).

6.1 Base-learners

Vi vil i dette underkapitel gå i dybden med de fire *base-learners*.

Vi kigge lidt nærmere på, hvilke hyperparametre kan have en stor betydning for den respektive modeltypes præstation, hvilke interessante sammenhænge der er blandt nogle af disse

størrelser og hvordan optimale værdier af disse typisk findes, givet de andre hyperparametre holdes ved lige.

6.1.1 Elastic Net

Elastic Net er langt den simpleste af de 4 *base-learners* vi har valgt at benytte.

Overordnet set kan *H2O*s *GLM*-funktion tage 4 forskellige former for en *GLM*:

en sædvanlig *GLM*, *GLM* med *L1-regularisation* (*LASSO*), *GLM* med *L2-regularisation* (*Ridge-regression*) og som en *GLM* med en kombination af *L1* og *L2* (*Elastic Net*). De 3 mulige udvidelser af den generealiserede lineære model gør den måske mindre egnet til inferens, men da vi sådan set udelukkende er interesseret i at benytte modellen til prædiktion, er dette nødvendigvis ikke en dårlig ting.

Dens 2 hyperparametre er, med tilhørende *H2O* standardværdier,

- *alpha* (*L1* og *L2* fordeling), standardværdi = 0.5 og skal ligge i $[0,1]$
- *lambda* (regulariseringsstyrken), standardværdi udregnes af *H2O* vha. en heuristisk tilgang og skal være > 0

Da vores respons er numerisk og kan være både negativ og positiv, sættes *link*-funktion til at være *identity* varianten (specificeres som *family* = "*gaussian*" i *H2O*).

Sættes de to parametre til 0, så er der tale om en helt typisk *GLM*.

Får *lambda* derimod en værdi > 0 for *alpha* = 0 er der tale om *Ridge-regression*, hvor modellen har mulighed for at reducere indflydelsen af hver variabels koefficient ved at begrænse dens størrelse. Dette er især nyttigt hvis der findes en mistanke om multikollinearitet blandt variablene, hvilket muligvis er tilfældet med de *features* vi har valgt at arbejde med. Der er tale om et relativt højt antal af dem og det kan tænkes, at nogle af disse, til en vis grad, kan forklares af andre, som f.eks. *SMA* variable.

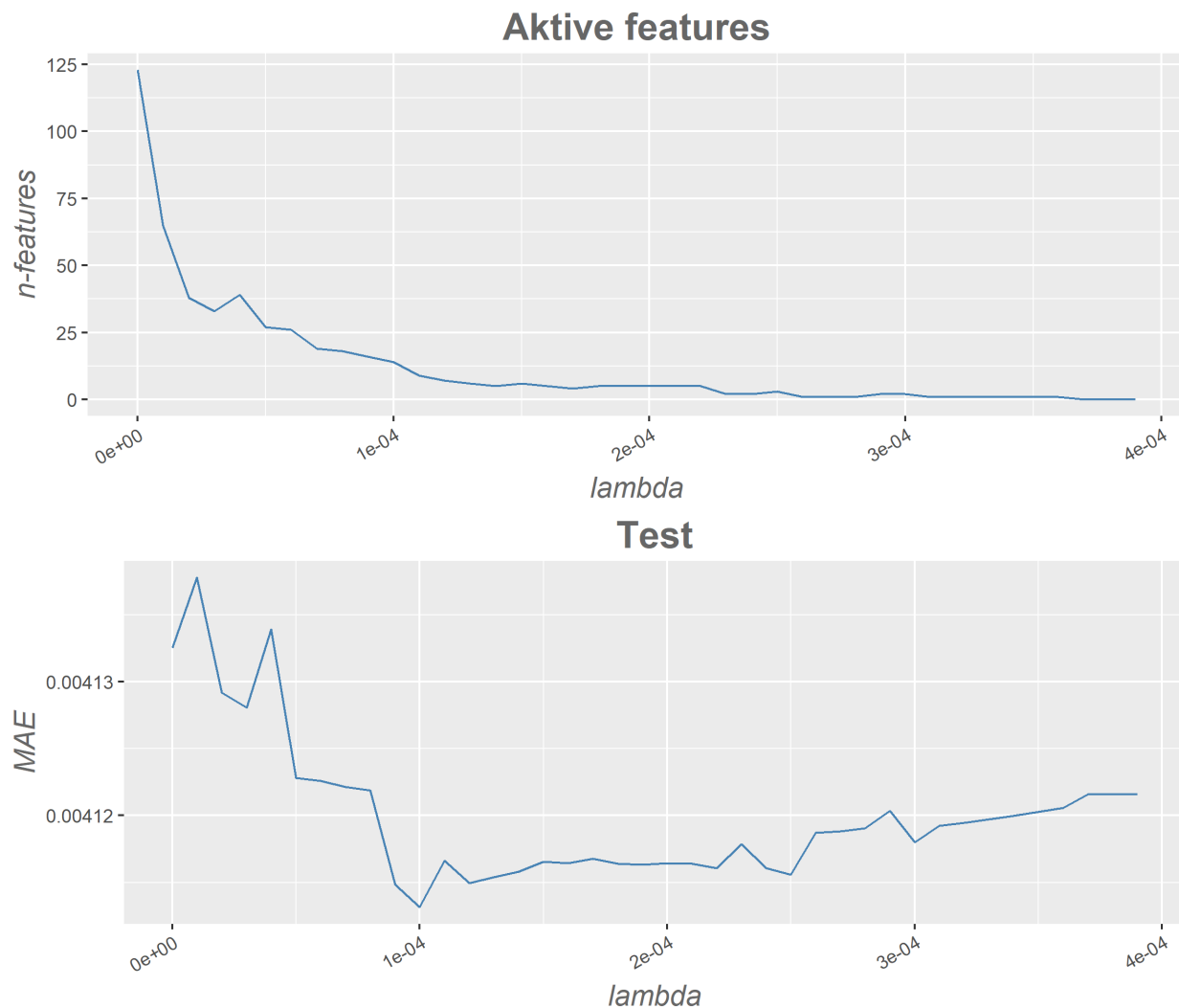
LASSO, hvor *alpha* = 1 og *lambda* > 0 , gør brug af denne reduceringseffekt men tager den også et skridt videre, hvor der særligt tages hensyn til korrelation blandt variablene. Findes der en gruppe af variable med høj intra-korrelation, så beholdes den ene af disse mens de andre slås til 0. Denne såkaldte *feature-selection* kan meget vel være gavnlig for os, da der på baggrund af samme argumenter som for *Ridge-regression*, sagtens kan findes korrelerede variable blandt de mange *features*. Ved at reducere dimensionaliteten i data, opnås ofte en bedre *out-of-sample* error for lineære modeller, hvilket i øvrigt også gælder for reduceringsmetoden i *Ridge-regression*.

Endeligt hvis $0 < \alpha < 1$, med *lambda* > 0 , har vi en *Elastic Net*, der kombinerer styrkerne fra både *LASSO* og *Ridge-regression*.

alpha styrer altså modellens fordeling mellem *L1*- og *L2-regularisation* mens *lambda* styrer styrken af denne regularisering. Passende værdier for *lambda* er ofte meget små.

Idet vi er bevidste om faren ved for mange kollineære og korrelerede *features*, er det interessant at se hvordan antallet af aktive *features* ændres ved forskellige værdier for *lambda*, og hvordan modellens præstationsevne på *test* data, i form af *MAE*, varierer for disse værdier af *lambda*.

Vi fitter en *Elastic Net* på *train* med $\lambda = 0, 0.00001, 0.00002, \dots 0.00099, 0.00100$, holdende α fast på dens standardværdi på 0.5 , hvor $L1$ og $L2$ vægter lige meget, og finder *MAE* ved at prædiktere på *test* for hver iteration.



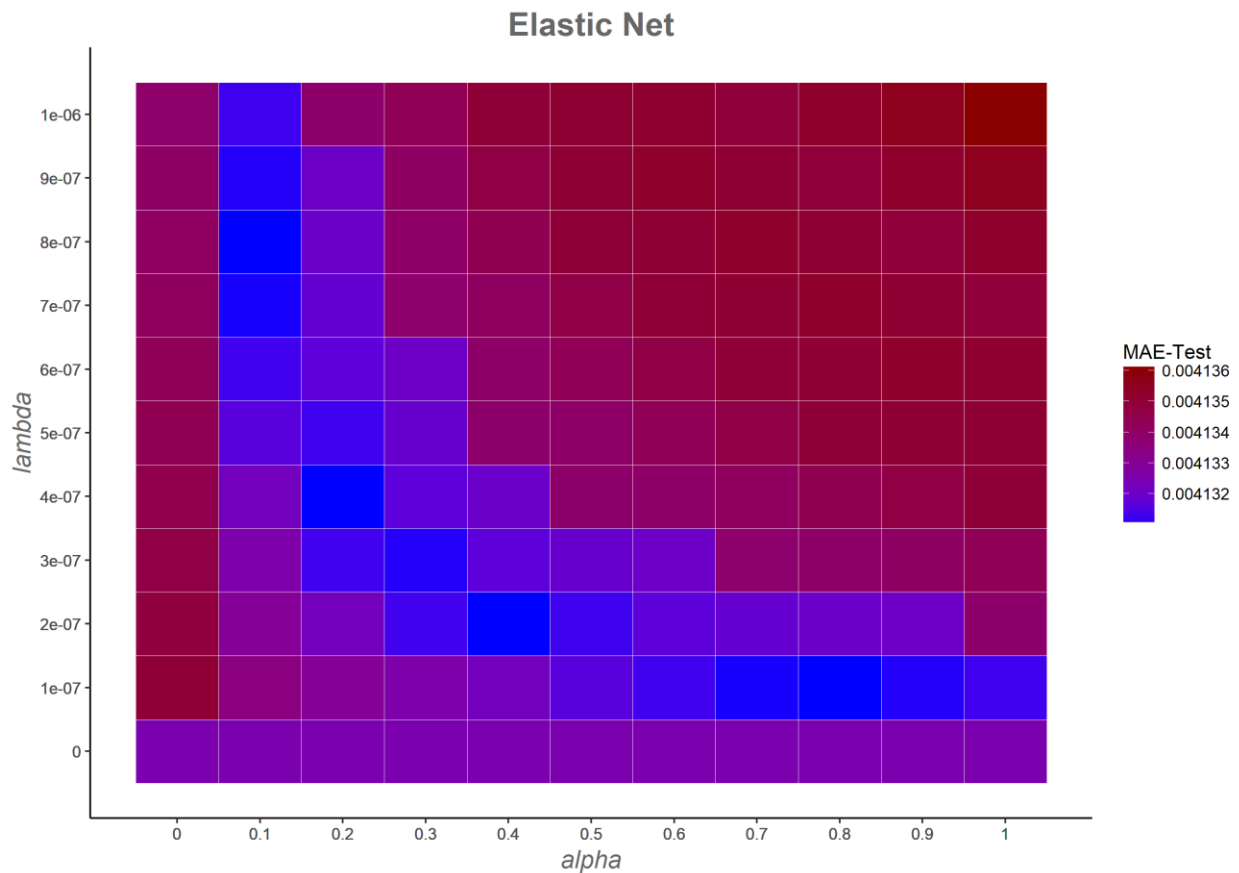
Det første vi bemærker er at et *Elastic Net* nok er til at foretrække for vores data.

Ved $\lambda = 0$, hvor *feature*-rummet er urørt, er *MAE* nemlig relativt høj. Men hvis λ øges, så har det en kraftig, men aftagende, effekt på antallet af aktive *features*, der endvidere har en gavnlig effekt på modellens præstation, da dette fører til en lavere og lavere *MAE*, med et minimum for $\lambda = 0.0001$. Herefter lader det altså til at en kraftigere regularisering, medført af højere λ , ikke resulterer i yderligere gevinst for modellen.

Det kunne tyde på at der findes korreleration, samt multikollinearitet, blandt vores *features*,

eller at flere af disse ikke har en lineær sammenhæng med vores respons, idet modellen er hurtig til at slå mange af dem til 0 og samtidig begrænses størrelsen for de andre koefficienter. Se bilag for en sammenligning af de uregulariserede (vilkårligt α , $\lambda = 0$) og regulariserede koefficienter.

Vi husker på, at ovenstående kun er gældende for $\alpha = 0.5$ hvor $L1$ og $L2$ vægtes lige højt. Lad os se på sammenspillet imellem de to hyperparametre, ved igen at fitte på *train* for forskellige værdier af λ , men også α , og prædiktere på *test*.



Det er klart, at der er en sammenhæng mellem hyperparametrene.

For $\lambda = 0$ og vilkårligt α er der ingen regularisering og MAE er konstant. Bevæger vi os over til *Ridge-regression*, hvor $\alpha = 0$, $\lambda > 0$, øges MAE ifht. den normale lineære model. Men så snart som vi har at gøre med en *Elastic Net* ($0 < \alpha < 1$, $\lambda > 0$) bliver sammensillet meget mere tydeligt.

For lave værdier af α , hvor $L2$ -regularisation foretrækkes, og holdende λ højt, præsterer modellen relativt godt på *test*. Begynder vi derimod at vægte $L1$ -regularisation mere og mere ved at øge α , må vi også sænke λ og dermed styrken af den samlede regularisering mere og mere, for at holde MAE nogenlunde ved lige.

Som konklusion er det allerhelst fornuftigt at benytte en *Elastic Net*, ifht. de andre varianter, idet prædiktionerne lader til at forbedres for $\lambda > 0$ og $\alpha = (0,1)$. Der er dog et forhold mellem disse to, som vi skal være opmærksomme på når vi eventuelt skal identificere et passende interval for hyperparametrene i vores *grid-search* senere i kapitlet.

6.1.2 Random Forest

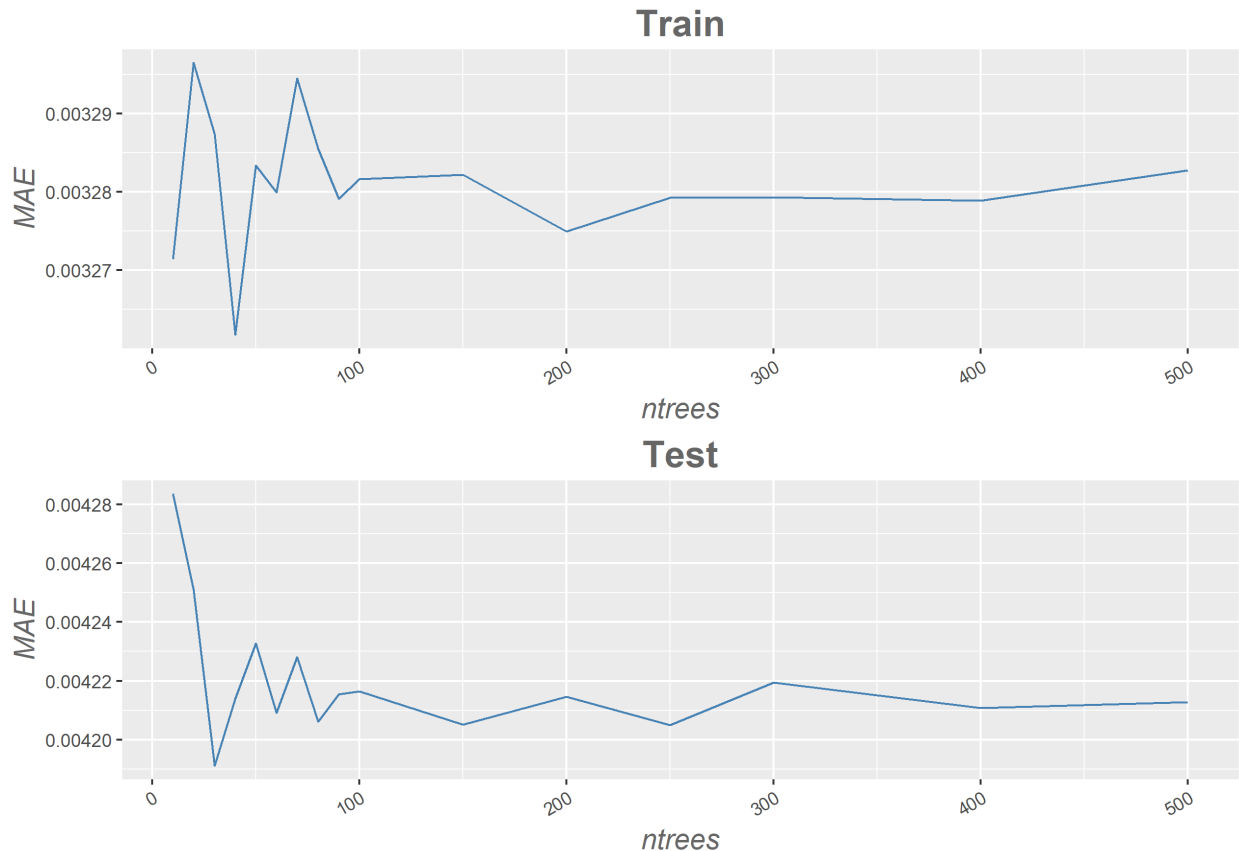
I forhold til *H2O*s *GLM* indeholder *RF*-funktionen mange hyperparametre, hvoraf nogle typisk er vigtigere at være opmærksom på end andre.

Vi begrænser os med at kigge på de hyperparametre der typisk anses som værende mest relevante for modellens præstation. Disse er givet nedenunder, med tilhørende *H2O* standardværdier:

- *max_depth* (maksimale dybde for hvert træ), standardværdi = 20
- *ntrees* (antal træer), standardværdi = 50
- *mtries* (antal tilfældigt trukne søjler for hvert niveau i træerne), standardværdi = -1 (hvor -1 indikerer $mtries = p/3$ (for regression), hvor $p = \text{antal features}$)
- *min_rows* (mindste antal obs. i en leaf node for at splitte), standardværdi = 5 (for regression)
- *nbins* (antal bins i histogrammet der splittes ud fra), standardværdi = 20
- *sample_rate* (% af obs. der skal bootstrappes for hvert træ), standardværdi = 0.632 og skal ligge i $[0,1]$
- *col_sample_rate_per_tree* (% af søjler der skal samples for hvert træ), standardværdi = 1 og skal ligge i $[0,1]$

Den første, og måske mest elementære af disse, er antallet af træer, *ntrees*, skoven skal dyrke. Generelt vil en højere værdi af denne ikke skade modellens præstationsevne, men fører istedet ofte til en lavere varians og dermed mere præcise prædiktioner.

Vi kigger på forskellige værdier af denne, ved at fitte en *RF* med standardværdier for hyperparametrene, dog hvor *ntrees* = 10,20, ...,80,90,100,150,200,250,300,400,500 og udregner *MAE* for hhv. *train* og *test* data.

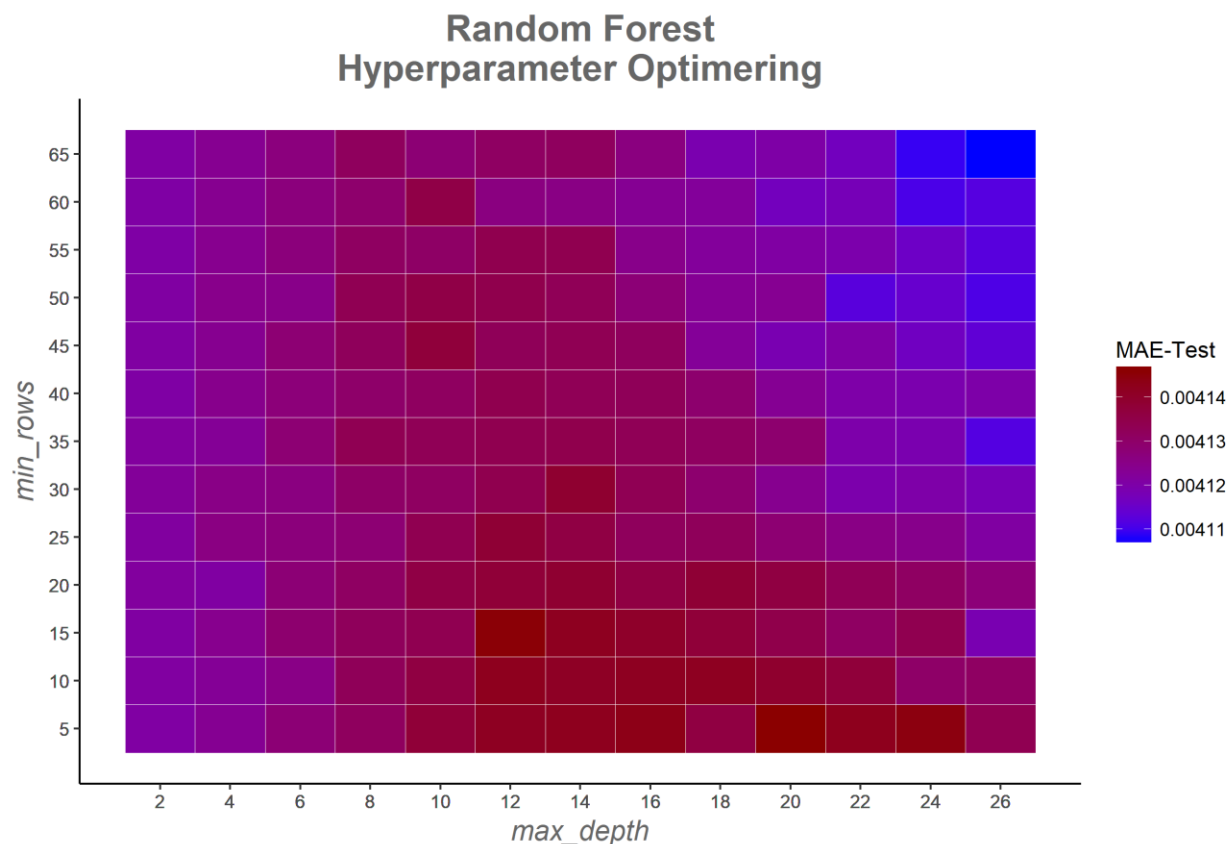


Vi bemærker først hvordan modellen generelt set er bedre til at prædikere på *train* ifht. *test*, hvilket ikke kommer som en overraskelse, idet den er opbygget på nemlig *train*. Enhver overvejelse omkring hyperparametre ud fra figurer som ovenstående, bør baseres på modellens præstation på det *usete* eller *out-of-sample* data, nemlig *test*. Baseres overvejelser ud fra *train* istedet er der chance for at modellen *overfitter* og præsterer dårligt på *out-of-sample* data.

I dette tilfælde er det optimale $ntree = 30$, med en tilhørende $MAE = 0.00419$ og for højere værdier for $ntrees$ ses MAE at konvergere mod en værdi i intervallet $[0.00420; 0.00422]$. Det lader til at gøre brug af et højt $ntree$, der også medfører en meget længere træningstid, ikke giver nogen klar forbedring for modellens præstation. $ntrees$ skal altså ikke være for lavt, og ej heller for højt. Umiddelbart er intervallet $ntree = [30; 150]$, hvor standardværdien 50 også er indeholdt, et fint udgangspunkt i vores eventuelle søgning efter passende hyperparameterintervaller.

Ligesom for *ntrees*, er *max_depth* en hyperparameter der styrer selve træstrukturen i *RF*, idet den afgør hvor mange niveauer, og dermed indirekte, hvor mange *decision nodes* og tilhørende *splits* hvert træ indeholder. Generelt vil træningstiden være eksponentielt voksende med *max_depth*.

Idet *min_rows* styrer hvor mange observationer der *mindst* skal være i en *leaf node* før algoritmen vælger at splitte den tilsvarende gren yderligere, deler den "træstruktur-egenskaben" med *max_depth*, dog kan den have en omvendt effekt på træningstiden: jo højere *min_rows*, jo hurtigere stoppes yderligere *splits* i de enkelte grene. Derudover kan disse to hyperparametre have et sammenspil ifht. *over-* og *underfitting*, illustreret i nedenstående figur.



For helt lave værdier af *max_depth* præsterer modellen forholdsvis gennemsnitligt for samtlige værdier af *min_rows*. i.e. der *underfittes*. Årsagen er, det lave *max_depth* forhindrer samtlige træer i at udføre et tilstrækkeligt antal *splits* til at fange strukturen i *test* data, og samtidig er antallet af observationer i *train* så højt, at selv *min_rows* = 65 ikke er højt nok til at have en effekt. Husk på, at *train* er udgjort af 287.985 observationer.

I takt med at *max_depth* øges, startende fra omkring 8-10, og for lave værdier af *min_rows*, vil træerne begynde at *overfitte* forholdsvis meget, fordi antallet af observationer i hver *leaf node*, for disse værdier af *max_depth*, stadig er for højt for at de lavere værdier af *min_rows* kan have en effekt. Hvis vi nu øger *min_rows* ses en mere betydelig sammenhæng mellem de to hyperparametre. Vi ser at øgningen i *min_rows* forhindrer træerne i at grave unødvendigt

dybt i nogle af grenene, hvilket dæmper overfitningen, og dermed reducerer *MAE* ifht. de lavere værdier for *min_rows*.

Denne effekt kommer fuldt til syne ved de to højeste værdier (i figuren) for de to hyperparametre, hvor *MAE* er lavest.

Det lader altså til, at vi skal finde en passende balance imellem disse to hyperparametre i vores eventuelle søgning efter passende hyperparameterintervaller. *min_rows* skal allerhelst øges fra standardværdien på 5, mens at tage udgangspunkt i en søgning omkring *max_depths* standardværdi på 20 nok er en fin start.

Som sagt, er der flere andre relevante hyperparametre vi skal være opmærksomme på i vores interval søgning og som vi kunne illustrere på denne måde, men vi vælger at bevæge os videre til den anden træ-baserede *ensemble*, nemlig *GBM*.

6.1.3 Gradient Boosting Machine

Som med *H2O*s *RF*-funktion indeholder *GBM*-funktionen mange hyperparametre.

Vi begrænser os igen med at se på de hyperparametre der ofte anses som værende mest relevante for modellens præstation. Disse er givet nedenunder, med tilhørende *H2O* standardværdier:

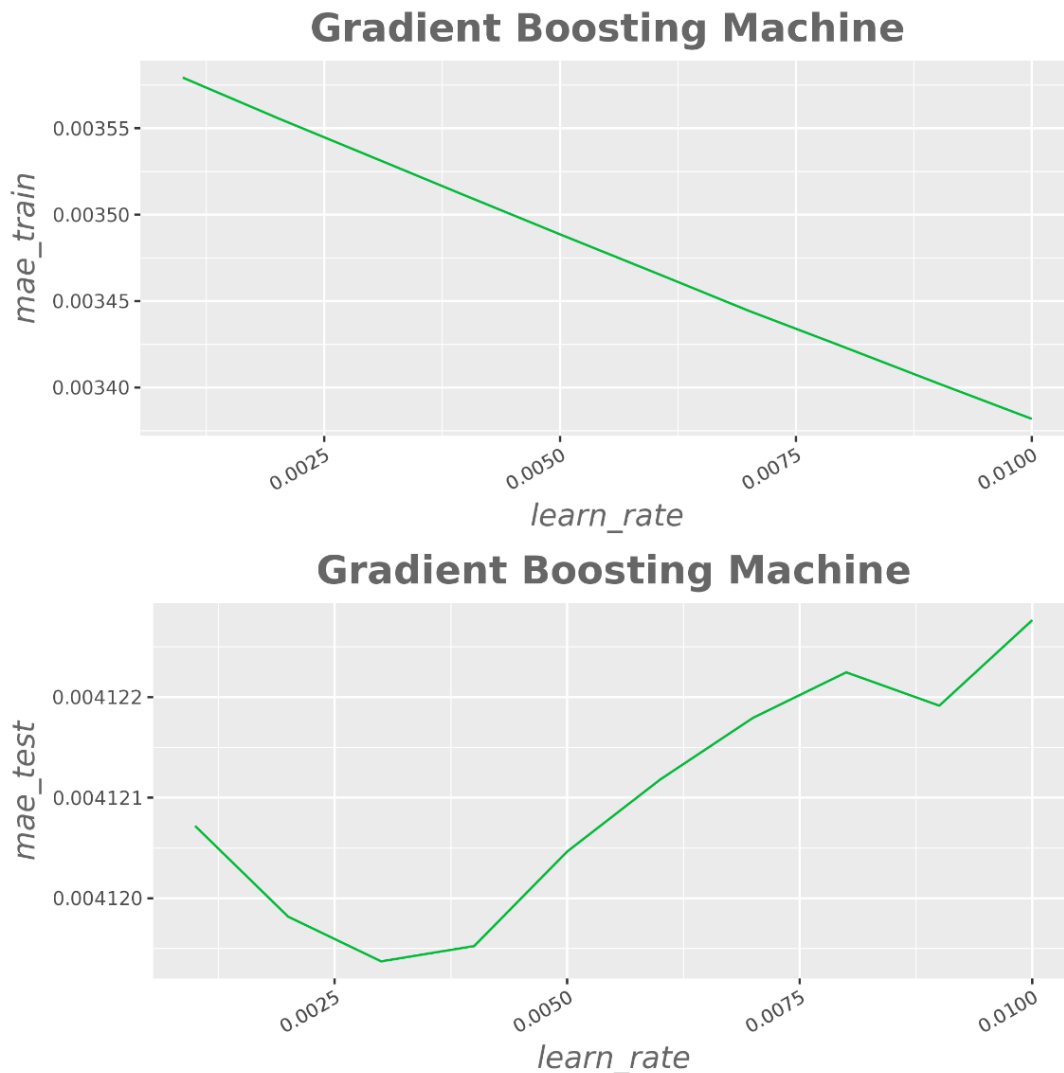
- *max_depth* (maksimale dybde for hvert træ), standardværdi = 5
- *ntrees* (antal træer), standardværdi = 50
- *col_sample_rate* (antal tilfældigt trukne søjler for hvert niveau i hvert træ), standardværdi = 1 (hvor 1 indikerer at alle søjler trækkes. Denne er altså en ratio af antal *features*)
- *min_rows* (mindste antal obs. i en leaf node for at splitte), standardværdi = 5 (for regression)
- *learn_rate* (hvor meget der læres fra træ til træ), standardværdi = 0.1 og skal ligge i $[0,1]$
- *learn_rate_annealing* (en faktor ganget på *learn-rate*, der resulterer i at *learn_rate* reduceres fra træ til træ), standardværdi = 1 skal ligge i $[0,1]$.
- *sample_rate* (% af obs. der skal samples uden tilbagelægning), standardværdi = 1 og skal ligge i $[0,1]$ (dette er en ratio af antal observationer i *train* datasættet).

Den mest interessant hyperparameter i *GBM*, er måske *learn-rate*. Denne spiller meget sammen med *ntrees*.

Generelt vil man typisk kunne fortsætte med at gøre *out-of-sample error* bedre ved at øge *ntrees*, men tilgængæld bliver den optimale værdi for *learn_rate* da lavere. Det betyder at man typisk skal finde et antal træer som man har tid til at træne og derefter optimere *learn_rate*, til det antal træer man har valgt. Nedenfor ses to plots af henholdsvis *train MAE* og *test MAE*, hvor

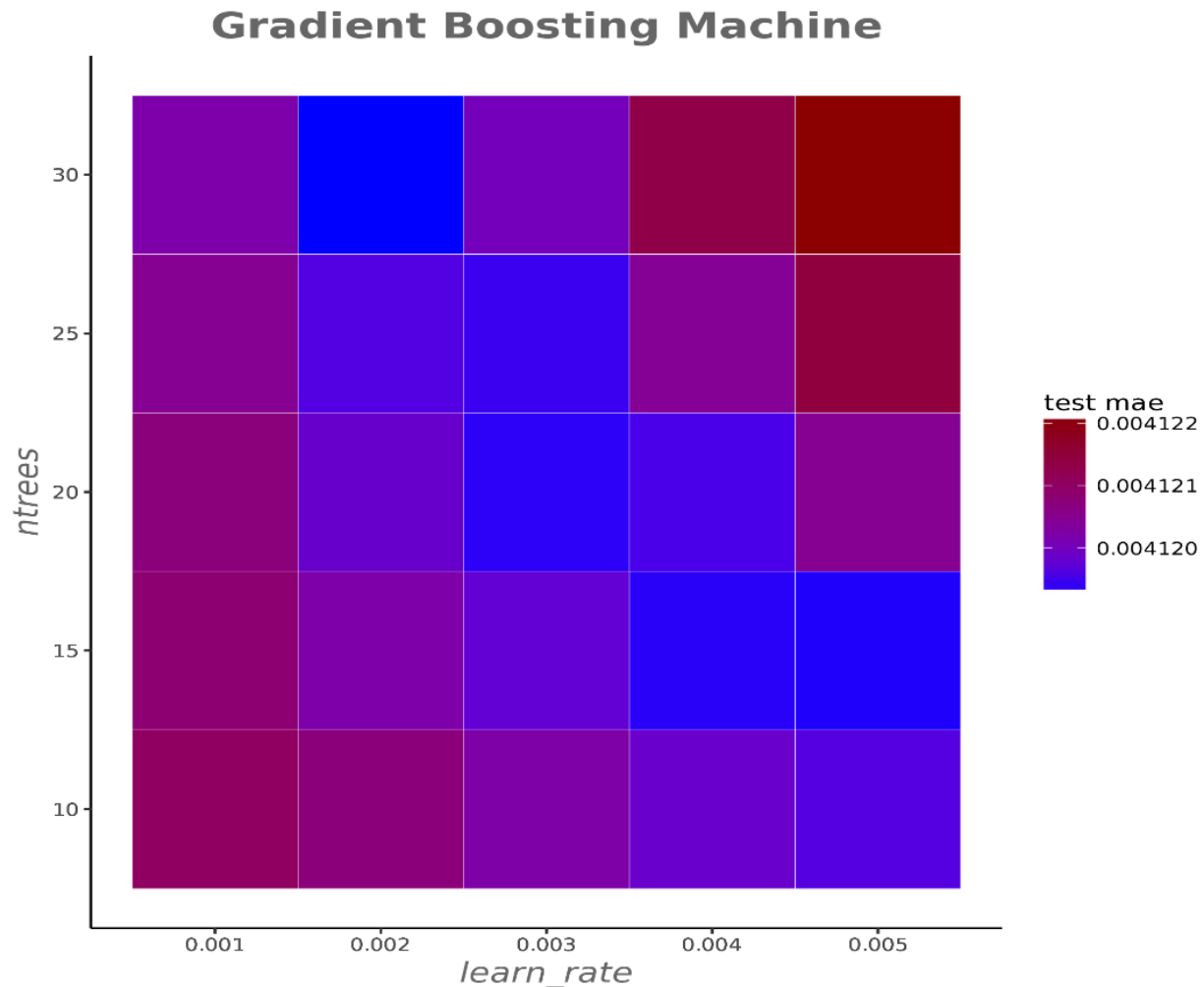
der ses på *learn_rate* (*ntrees* og *max_depth* er i disse plots henholdsvis 20 og 30).

Det ses at *train MAE* falder med lavere værdier for *learn_rate*, men at *test MAE* tilgængæld kun falder indtil *learn_rate* har en værdi på 0.003, hvorefter det øges for højere værdier af *learn_rate*. Dette giver god mening, da de 20 træer, der kommer efter det første træ i modellen, prøver at korrigere for *errors* i *train* data fra de foregående træer og *train MAE* vil dermed fortsætte med at falde for højere værdier af *learn_rate*. Når *learn_rate* i dette tilfælde stiger over 0,003 begynder vi at fitte data specifikt til *train* (overfitte), og *test MAE* stiger som følger.

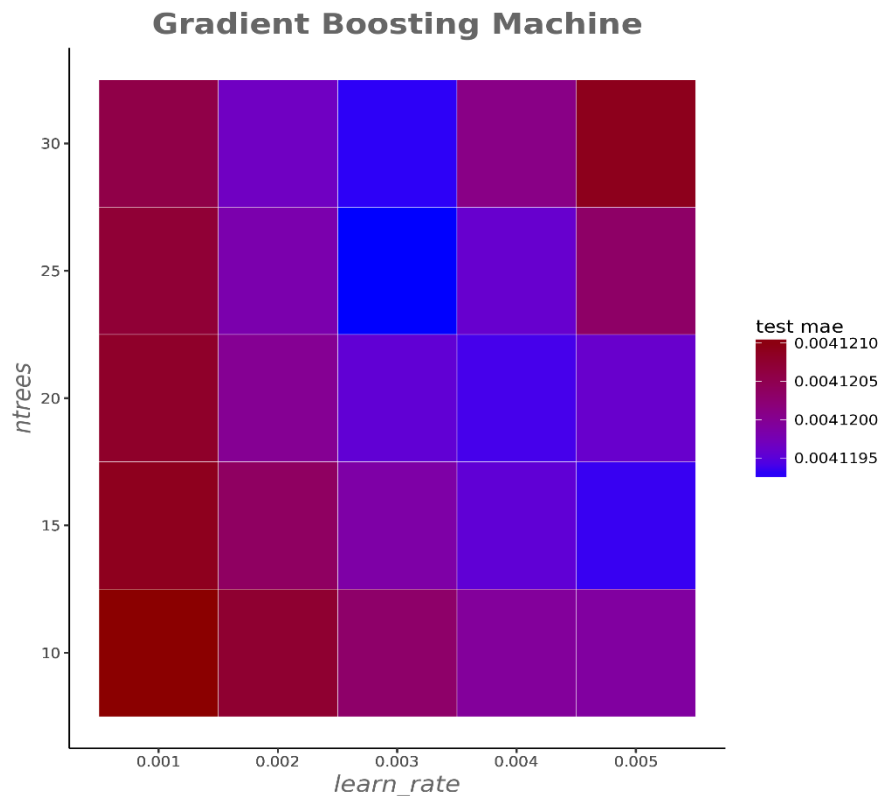


Nedenfor ses et *heatmap* af sammenhængen mellem *ntrees* og *learn_rate*, hvor de mest blå kasser, er de kombinationer af *ntrees* og *learn_rate* med de laveste *test MAE*. Her er det tydeligt, at en optimal værdi af *learn_rate* er negativt korreleret med *ntrees* og vi skal derfor sænke *learn_rate* hvis vi øger *ntrees*, alt andet lige. Det ses også, at den laveste *test MAE* er når

$ntrees = 30$ og $learn_rate = 0,002$, alt andet lige, hvilket kunne det tyde på at vi ville kunne få et lavere *test MAE* ved at øge $ntrees$ og reducere $learn_rate$.



Som sagt så er *learn_rate_annealing* en hyperparameter, der bliver ganget på $learn_rate$ sådan at dennes effekt reduceres for hvert træ. Det gør at man kan vælge en højere startværdi af $learn_rate$ uden nødvendigvis at skulle reducere antallet af træer. Efter et antal af træer er dyrket er $learn_rate$ blevet meget, eller noget, lavere (dette vil afhænge af hvor mange træer er bygget og hvor langt faktoren er fra 1). Med andre ord er *learn_rate_annealing* en faktor der gør at korrektionsniveauet for hvert træ falder i antallet af træer dyrket, og man kan argumentere for at dette kan være gunstigt, da modellen forventet har et bedre fit efter et bestemt antal korrektionstræer og derfor ikke behøver at korrigeres så meget, som efter det første korrektionstræ.



Til venstre er effekten af at tilføje *learn_rate_annealing* = 0.98 illustreret. I forhold til det første *heatmap*, ses det at de blå felter (med lav *test MAE*) generelt er rykket en smule til højre, hvilket illustrer, at den optimale *learn_rate* for et givent antal *ntrees* er øget, hvilket er i overensstemmelse med at *learn_rate* skal starte højere, hvis man specificerer *learn_rate_annealing*, alt andet lige.

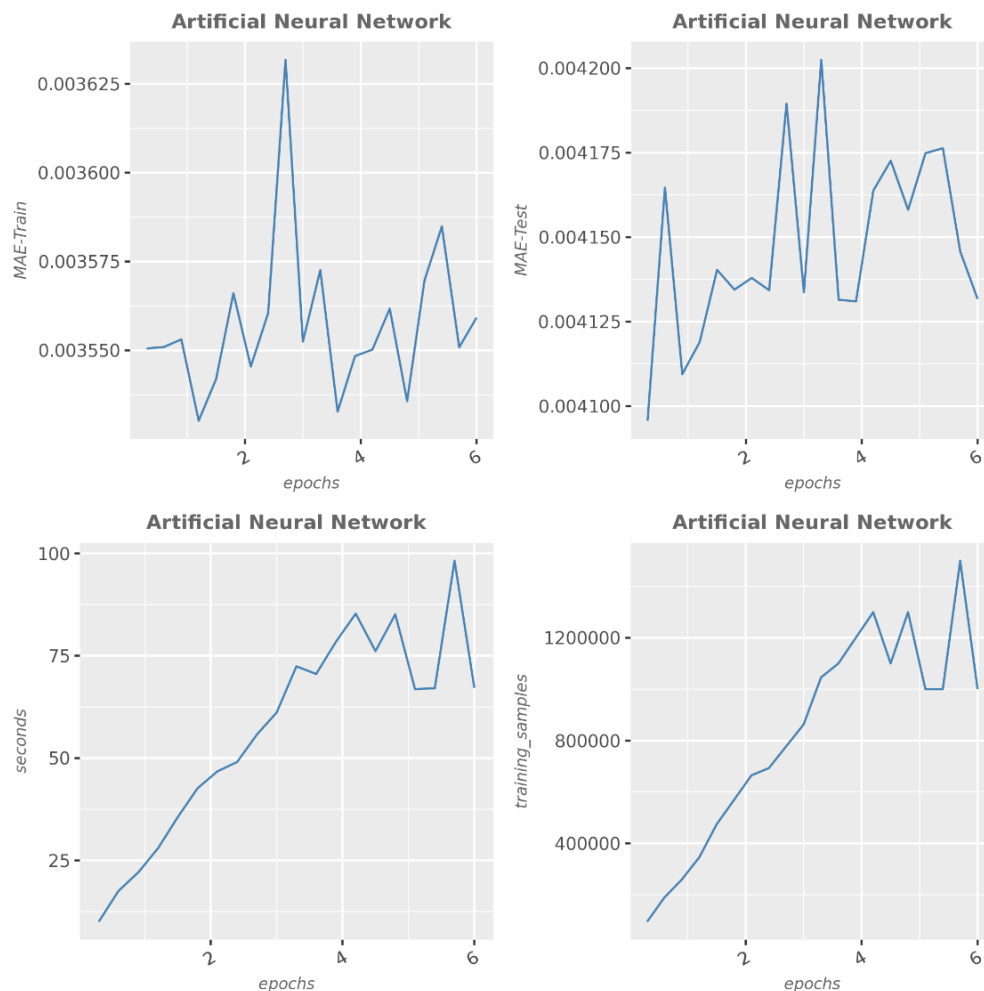
6.1.4 Artificial Neural Network

H2O's deeplearning-funktion er som standard et *Feed-Forward Multilayer Artificial Neural Network*, og har rigtig mange hyperparametre. Vi begrænser os igen ved at kigge på de hyperparametre der typisk anses som værende mest relevante for modellens præstation. Disse er givet nedenunder, med tilhørende *H2O* standardværdier:

- *hidden* (antallet af neuroner i hvert *hidden layer*), standardværdi = (200, 200), hvor hvert element siger hvor mange *hidden neurons* er i det respektive *layer*. Længden af vektoren siger dermed hvor mange *hidden layers* er i modellen.
- *loss* (hvilken fejl-metric vægtene fittes ud fra i *Back-propagation* processen) = "Automatic"
- *L2* (styrken af *L2-regularisering*), standardværdi = 0 (hvis specificeret til > 0 vil nogle af vægtene i *hidden neurons* reduceres til tæt på 0, ligesom *lambda* i *Ridge-regression*).
- *epochs* (maksimalt antal gange *train* strømmer igennem netværket), standardværdi = 10. Siger hvor mange gange *train* skal strømme gennem netværket i træningsfasen, men bliver overtrumpet af *stopping rounds*, hvis *stopping_rounds* ikke er inaktivt.
- *stopping_metric* (hvilken metric *early-stopping* kriteriet er baseret på) = "deviance".

- *rho* (minder om GBMs *learn_rate_annealing*), standardværdi = 0.99 og skal ligge i $[0,1]$.
- *epsilon*, (forhindrer at der divideres med 0 i vægt opdateringsprocessen (ADADELTA)) = $1e - 08$ og skal ligge i $[0,1]$.
- *activation* (aktiveringsfunktion for hver neuron), standardværdi = "Rectifier". Kan være en af "Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout", "Maxout", "MaxoutWithDropout".
- *hidden_dropouts*(andelen af neuroner i hvert hidden layer der skal slettes), standardværdi = (0.5, 0.5).
Er en ratio af *hidden neurons* der bliver smidt ud af modellen, hvis en aktiveringsfunktion med "WithDropout" er specificeret.

Vi vil i det følgende kigge lidt nærmere på hvilken betydning *epochs* har på den samlede træningstid, samt på *MAE*.



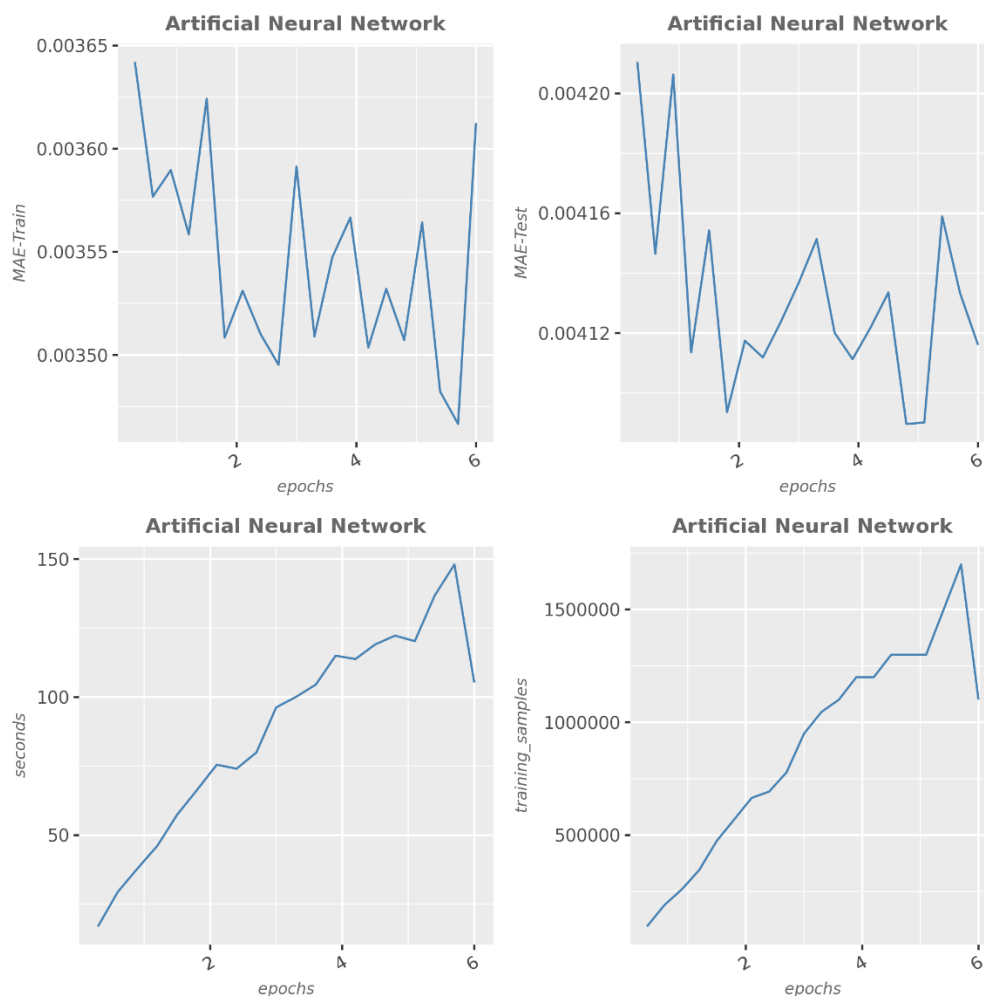
Ovenover ses i de to øverste plots udklip af *train* og *test MAE* af forskellige ANNs, hvor alle hyperparametre har standardværdier, uden *epochs*, som varierer fra 0.3 til 6. Det vil sige at data strømmer 0.3 til 6 gange igennem modellen i træningsfasen, hvis ikke *stopping rounds* var

aktiveret.

Her ses det at *early_stopping* er aktiveret som standard, og efter at *train* er strømmet igennem modellen omkring 4 gange, hvilket svarer til omkring 1-1.3 millioner observationer, stopper algoritmen med at træne. Ud fra de to nederste plots, ses det, at netværket stopper med at træne, da både tiden af træningen og antal observationer, der er strømmet igennem netværket, flader ud. Stoppet er sat af en simple moving average af *test errors*, som bliver tjekket efter omkring 90.000 observationer i vores tilfælde (vælges af *H2O* ud fra størrelsen af *train*). Hvis denne moving average af *test error* ikke bliver bedre, stopper træningen af netværket.

Nedenfor har vi ændret den faktor, der minder om *learn_rate_annealing* hos *GBM* modellen, nemlig *rho*, fra dens standardværdi på 0.99 til 0.5.

Man kan ane, at *test* og *train MAE* har fået en aftagende trend, istedet for næsten udelukkende at springe op og ned, som på det forrige plot. Samtidig ses det, at der nu skal flere observationer til før vores stop kriterie bliver udløst (ved ca. 1.2-1.6 millioner samples i forhold til ca. 1-1.2 millioner samples for *rho*= 0.99).



6.2 Grid-search

I dette underkapitel lægger vi ud med at tale mere detaljeret om, hvorfor vi vælger at finde frem til passende hyperparameterintervaller istedet for optimale hyperparametre. Endvidere redegør vi for hvad et *random-grid-search* er, ifht. det mere sædvanlige og udtømmende *Cartesian-grid-search*. Herefter beskriver vi mere nøje hvilken fremgangsmåde vi bruger for at finde frem til de passende hyperparameterintervaller.

I efterfølgende sektion udleder vi de passende hyperparameterintervaller for alle vores *base-learners*, som vi kommer til at benytte videre i vores *walk-forward-optimization* i kapitel 7. Dog viser vi kun den faktiske udledning for *Elastic Net* (*Cartesian-grid-search*) samt *Random Forest* (*Random-grid-search*).

Sidst i kapitlet sammenligner vi de bedst præsterende *base-learners* af hver type, med hinanden, og med *benchmark*-modellerne.

Redegørelse for hyperparameterintervaller

Det sædvanlige formål med et *grid-search*, er at finde værdier for en models hyperparametre der resulterer i de bedste *out-of-sample errors*.

Helt specifikt, så fittes modellen på *train* for alle mulige kombinationer af de specificerede værdier for hyperparametrene og hver model bruges til at prædiktere på *test*, og det foretrukne fejl-mål (f.eks. *MAE*) udregnes ud fra *out-of-sample errors*. Herefter udvælges modellen med det laveste fejl-mål, og på et fremtidigt tidspunkt, kan modellen benyttes til at prædiktere på nyt data.

Givet den bagvedliggende population for data ikke ændrer sig over tid, burde en model med de optimale hyperparametre stadig føre til gode *out-of-sample-errors*.

Er det derimod tilfældet at populationen ændrer sig over tid, er det nødvendigt at gentage ovenstående process for det nyindsamlede data for at "opdatere" de optimale parametre.

Vi har en formodning om, at det sidste tilfælde gælder for vores data.

Man kan nemlig forestille sig, at priser på valuta eller aktiver i høj grad er en refleksion af markedsaktørernes psykologi og sentiment – hvilket hurtigt kan skifte.

Vi vil derfor forsøge at tage højde for dette i vores adaptive *trading* strategi, ved at benytte intervaller for vores hyperparametre, istedet for fastsatte værdier.

Cartesian grid-search ifht. random grid-search

Et andet element vi må tage højde for er den store mængde data vi arbejder med, og den tidsomkostning det medfører at fitte en enkelt model på dette – selv på servere med mange kerner.

At fitte 100 forskellige *Elastic Nets* er typisk lynhurtigt, takket være modellens simplicitet. Denne metode hvor alle mulige kombinationer af hyperparameterværdier i de specificerede intervaller afprøves, kaldes et *Cartesian-search* i *H2O*.

At gøre det samme for de 3 andre *base-learners* er en helt anden sag.

Istedet for at lave et *Cartesian-grid-search*, så udfører vi det vi kalder for et *random-grid-search* (navnet er også udledt fra *H2O*) på de 3 andre *base-learners*.

Dette indebærer, at *H2O* tilfældigt udvælger hyperparameter kombinationer ud fra de specificerede intervaller, indtil enten antallet af fittede modeller overskrider en brugervalgt præference eller den samlede tid algoritmen har kørt overskrider et brugervalgt antal sekunder. Vi vælger at tage højde for den sidste mulighed – den samlede tid i sekunder.

Der er ulemper ved begge af disse typer *grid-search*.

Fordi der i et *random-grid-search* trækkes tilfældige kombinationer af hyperparametre fra deres respektive intervaller, er der en chance for at en kørsel ikke finder den bedste kombination.

Konsekvensen kan være, at vi ikke formår at finde frem til de bedste intervaller for hyperparametrene i vores *base-learners*.

Hvis vi derimod benytter *Cartesian-grid-search*, vil de hyperparameterintervaller vi lægger ud med at undersøge for de forskellige modeller, i håbet om at kunne forbedre disse, indeholde for mange kombinationer. Hvis vi f.eks. specificerer 10 forskellige værdier i intervallet for hver af de 7 hyperparametre vi har tænkt os at lede over i *Random Forest*, så skal 10.000.000 forskellige skove fittes, hvilket er noget af en opgave, selv for de store servere på *Amazon Web Service*.

Vi laver altså et kompromis for de 3 modeller vi kommer til at udføre *random-grid-search* på, hvor vi benytter *max_runtime_secs* parameteren i *grid.search*-funktionen i *H2O*. Det medfører, at vi ikke prøver alle kombinationer, hvis tiden det ville tage, overskrider vores brugervalgte antal sekunder.

Antallet af sekunder vi vælger at benytte for hver af disse 3 modeller er som følger:

- *Random Forest*, *max_runtime_secs* = 2000
- *Gradient Boosting Machine*, *max_runtime_secs* = 2000
- *Artificial Neural Network*, *max_runtime_secs* = 3000

Vi tillader *ANN* lidt mere tid end de to træ-baserede *ensembles*, da den indeholder lidt flere hyperparametre.

Fremgangsmåden for vores søgning af hyperparameterintervaller

Kort sagt, forløber vores søgning efter passende hyperparameter intervaller, for hver modeltype, således:

- Startende med et *grid-search* med meget grove intervaller for hyperparametrene, ser vi på hvordan hyperparameter værdierne for de 20 bedst præsterende modeller (i form af *MAE* på *test*) ser ud. Vi forsøger derefter at bevæge intervallerne i den retning, der tyder på en forbedring af *MAE*.
- Herefter køres *grid-search* igen og ovenstående procedure gentages.
- Hvis en hyperparameter konsistent holder sig indenfor det specificerede interval ved de forskellige ændringer i de andre hyperparametre, så forsøger vi at indskrænke dens interval og holder antallet af mulige værdier i intervallet nogenlunde lige.

Denne process vil vi fortsætte til vi er nogenlunde tilfreds med de top præsterende modeller i vores *grid-search*.

Vi lægger ud med vores *Elastic Net base-learner*.

6.2.1 Elastic Net (*Cartesian*-)grid-search

Som sagt så er *Elastic Net* den eneste af vores *base-learners* vi udfører et *Cartesian grid-search* på. Med de servere vi er udrustet med, er det forholdsvis hurtigt at fitte mange *Elastic Nets*.

Vi starter med nogle grove intervaller for hyperparametrene, givet ved

- *alpha* (*L1 og L2 fordeling*) = 0, 0.1, ..., 0.9, 1
- *lambda* (*regulariseringsstyrken*) = $0, 1e-8, 1e-7, \dots, 1e-3, 1e-2$

hvilket svarer til 88 forskellige modeller.

Vi fitter på *train* og finder *MAE* på *test* for samtlige modeller. Se figuren til højre for de 20 bedste modeller, sorteret efter *MAE*.

Den bedste af de 88 modeller er givet ved *alpha* = 0.5 og *lambda* $1e-04$, med en tilsvarende *MAE* = 0.00411314.

Umiddelbart er der et forholdsvis tydeligt interval for *lambda*, der fører til gode *MAE* størrelser, nemlig $1e-04$ til $1e-03$.

Derimod er signalerne fra *alpha* noget uklare, men det lader til at

alpha	lambda	mae_test
0.5	1e-04	0.00411314
0.6	1e-04	0.00411497
0.7	1e-04	0.00411581
0.9	1e-04	0.00411641
0.1	0.001	0.00411643
1	1e-04	0.00411644
0.8	1e-04	0.00411646
0.1	0.01	0.00412158
0.4	0.001	0.00412158
0.6	0.01	0.00412158
1	0.01	0.00412158
0.9	0.001	0.00412158
0.3	0.001	0.00412158
0.8	0.01	0.00412158
0.7	0.001	0.00412158
0.9	0.01	0.00412158
0.6	0.001	0.00412158
0.5	0.01	0.00412158
0.4	0.01	0.00412158
1	0.001	0.00412158

modeller med et forholdsvis høj værdi af α foretrækker en λ værdi på $1e-04$ (med "foretrækker" mener vi, at disse kombinationer fører til en forholdsvis lav MAE på test).

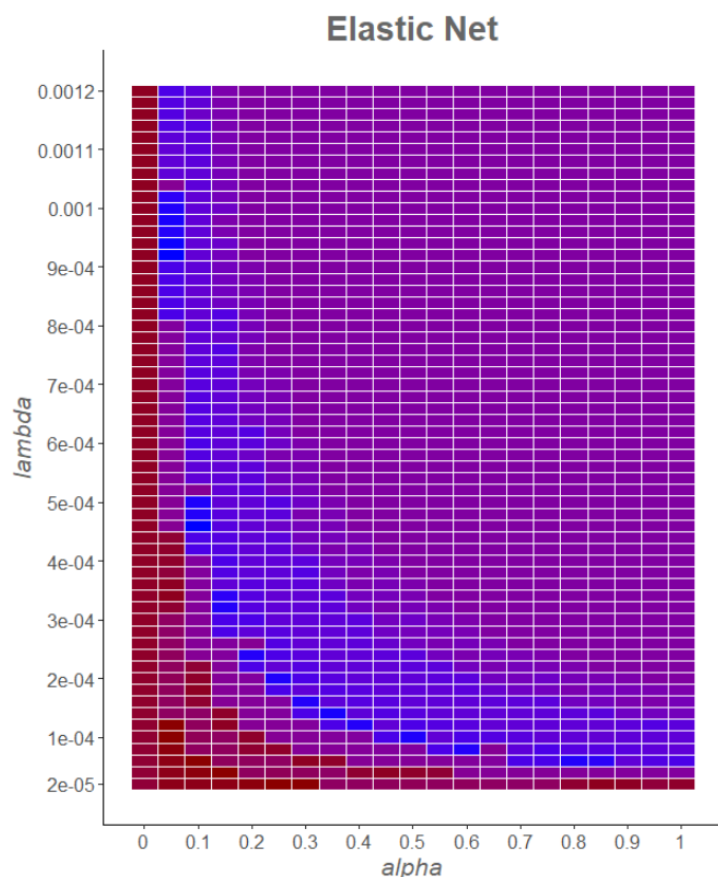
Vi vælger at indsnævre intervallet for λ ved at lede omkring dets 2 bedste præsterende værdier, $1e-04$ og $1e-03$, og holder intervallet for α ved lige. Dog øges de mulige udfald i α 's interval, i håbet om hyperparameteren sender nogle mere klare signaler for de nye værdier af λ .

Vi kører næste iteration af vores *grid-search*, hvor de nye intervaller er givet ved

- α (L1 og L2 fordeling) = 0, 0.05, ..., 0.95, 1
- λ (regulariseringsstyrken) = $2e - 5$, $4e - 5$, ..., $1.18e - 3$, $1.20e - 3$

Idet vi har øget de mulige udfald i begge parameters intervallet, resulterer den nye iteration i 1260 forskellige modeller.

Ud fra tabellen til højre for de 20 bedst præsterende modeller, ses det, at MAE generelt er forbedret efter at have konstrueret de nye intervaller for hyperparametrene. Antallet af modeller ved denne iteration er dog meget højt, så det er nok en god idé, at visualisere samtlige modeller med en *heatmap*, givet nedenunder.



alpha	lambda	mae_test
0.1	0.00046	0.00411276
0.05	0.00092	0.00411277
0.05	0.00094	0.00411288
0.35	0.00014	0.00411301
0.05	0.00098	0.00411303
0.5	1e-04	0.00411314
0.25	2e-04	0.00411315
0.1	5e-04	0.00411315
0.05	0.001	0.00411316
0.8	6e-05	0.00411325
0.6	8e-05	0.00411325
0.4	0.00012	0.00411326
0.3	0.00016	0.00411326
0.2	0.00024	0.00411326
0.15	0.00032	0.00411326
0.1	0.00048	0.00411326
0.05	0.00096	0.00411328
0.85	6e-05	0.00411349
0.15	0.00034	0.00411349
0.05	0.00102	0.00411351

Visualiseringen gav pote. Det viser sig, at det inverse forhold imellem hyperparametrene vi observerede i vores praktiske gennemgang af *Elastic Net*, også holder for disse intervaller, idet en optimal λ værdi generelt falder når α stiger.

De 3 bedste modeller ud fra MAE, har alle en lav α værdi på hhv. 0.1, 0.05 og 0.05, og tilsvarende høje

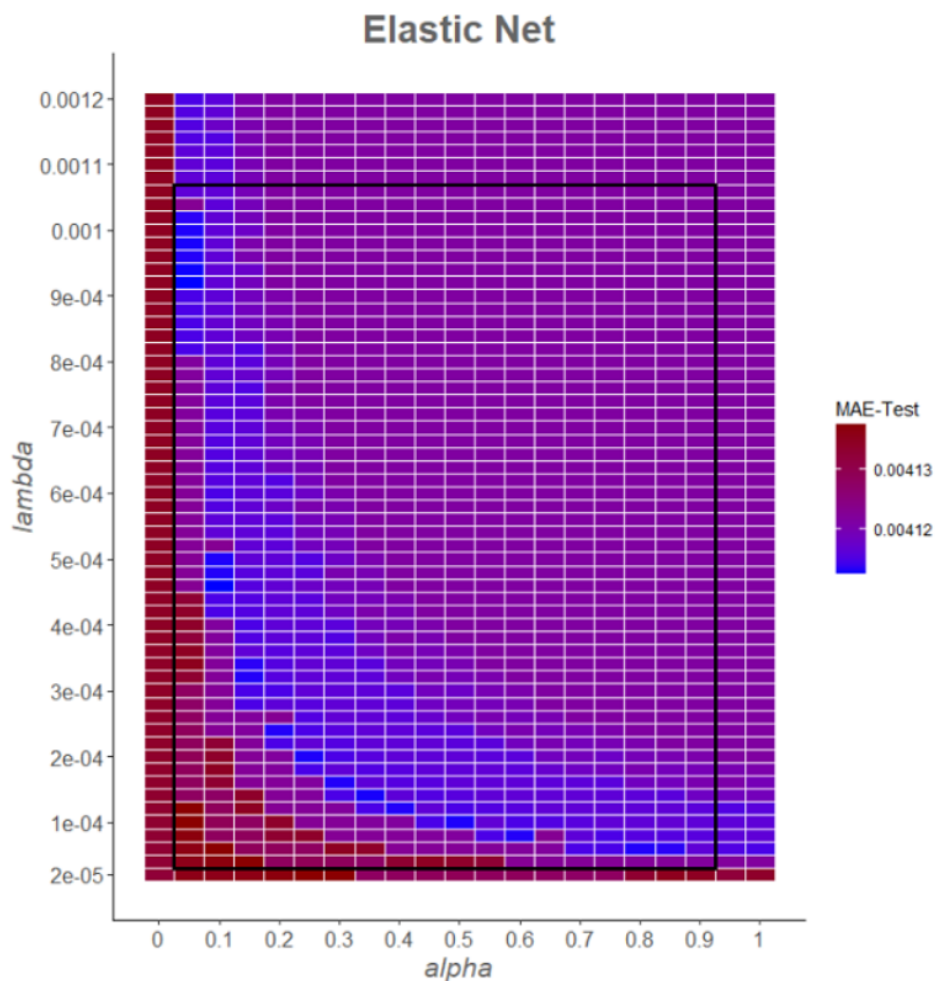
værdier for λ . Der findes også en koncentration af meget passende modeller i omkring midten af α intervallet, hvor λ er lavere. 2 af de bedste 20 modeller har en relativ høj α værdi, på hhv. 0.8 og 0.85.

Vi husker på, at et *grid-search* af *Elastic Net* er meget hurtigt ifht. de andre 3 *base-learners*, og vi kan derfor tillade os at træne langt flere forskellige modeller for hvert vindue i vores *walk-forward-optimization*, end for de andre.

Ud fra ovenstående overvejelser, ser vi os nogenlunde tilfreds med at specificere vores endelige, passende hyperparameter intervaller som

- α (L1 og L2 fordeling) = 0, 0.05, ..., 0.85, 0.9
- λ (regulariseringsstyrken) = $4e - 5$, $6e - 5$, ..., $1.04e - 3$, $1.06e - 3$

Disse intervaller er illustreret i figuren nedenfor.



6.2.2 Random Forest (*random-grid-search*)

Random Forest er den ene af vores 3 mere komplicerede *base-learners*, hvor et *Cartesian-grid-search* hurtigt bliver meget tidskrævende. Med 7 hyperparametre man kan skrue på kan antallet af mulige modeller blive skyhøjt, alt efter hvor mange skridt der findes i de forskellige intervaller der specificeres.

Vi laver derfor et *random-grid-search* ved at trække tilfældigt valgte kombinationer af hyperparametrene fra deres specificerede intervaller. På den måde kan vi benytte forholdsvis brede intervaller indeholdende flere værdier i vores *grid-search*, men uden de store tidsomkostninger. Ved et *random-grid-search* som dette, udelukker vi med stor sandsynlighed at ramme både lokale og globale minima af *MAE* for *test*, men der er gode muligheder for at lokalisere forholdsvis passende områder af hyperparameterrummet og samtidig også finde ud af hvilke parametre er mest indflydelsesrige på præstationen af modellen.

Der må naturligvis vælges et stop-kriterie for denne type *grid-search* i *H2O*.

Vi husker på, at *max_runtime_secs* bliver sat til 2000 for hvert *random-grid-search* for *RF*.

Vi starter igen med nogle grove intervaller for hyperparametrene, og lader disse ligge omkring *H2O*s standardværdier for de respektive hyperparametre. De er indledelsesvis

- *max_depth* (maksimale dybde for hvert træ) = 5, 10, ..., 25, 30
- *ntrees* (antal tilfældigt trukne søjler for hvert niveau i hvert træ) = 10, 20, ..., 40, 50
- *min_rows* (mindste antal obs. i en leaf node for at splitte) = 2, 6, ..., 26, 30
- *nbins* (antal bins i histogrammet der splittes ud fra) = 10, 20, ..., 40, 50
- *ntrees* (antal træer) = 30, 60, ..., 90, 120
- *sample_rate* (% af obs. der skal bootstrappes) = 0.2, 0.4, ..., 0.6, 0.8
- *col_sample_rate_per_tree* (% af søjler der skal samples for hver træ) = 0.5, 0.6, ..., 0.9, 1

Denne første iteration producerer 96 forskellige modeller før vores *max_runtime_secs* overskrides. De 20 bedste modeller ses i figuren nedenunder (se *Grid 1* figuren).

Det allerførste vi bemærker, er hvordan samtlige af de top 20 *RF* modeller præsterer bedre end vores bedste *Elastic Net*, der havde en *MAE* = 0.00411276.

En af årsagerne til *RF*s klare overlegenhed ihft. *Elastic Net* kan nok forklares ved, at vi i vores setup, hverken har fittet *Elastic Net* med vekselvirkninger eller ikke-lineære sammenhænge, mens *RF* netop tager højde for dette.

Lad os tage et nærmere blik på hvad vi kan udlede om hyperparametrene ved at kigge på de bedste modeller.

Hvis vi begynder ved at se på den første søjle i tabellen, så ses det at relativt høje værdier af *col_sample_rate_per_tree* fører til gode *MAE*, men hyperparameterens lavere grænse på 0.5 forekommer 6 gange. Vi rykker dens lavere grænse længere ned.

col_sample_rate_per_tree	max_depth	min_rows	mtries	nbins	ntrees	sample_rate	mae
1	30	30	10	40	60	0.2	0.00409438
0.9	25	30	10	40	60	0.2	0.00409554
0.9	30	30	10	10	30	0.4	0.00409653
0.6	30	22	20	30	120	0.4	0.00409817
0.7	25	22	10	30	60	0.2	0.00409893
0.5	30	26	30	10	120	0.6	0.00409939
0.5	25	26	10	40	60	0.6	0.00409981
0.7	25	26	10	40	120	0.6	0.00410026
0.5	30	26	30	10	30	0.4	0.00410144
0.5	30	18	10	10	30	0.4	0.00410348
0.9	30	18	10	40	120	0.8	0.00410478
0.8	30	22	20	20	90	0.8	0.00410568
0.7	25	10	10	30	120	0.2	0.00410577
0.6	20	22	20	50	60	0.2	0.0041059
0.5	25	30	20	50	90	0.8	0.00410641
0.7	20	14	10	20	120	0.2	0.00410964
0.5	20	6	10	50	120	0.2	0.00411039
0.9	25	6	10	40	120	0.4	0.00411068
0.8	30	30	50	50	60	0.4	0.00411091
1	30	26	50	20	60	0.4	0.00411161

Random Forest – Grid 1

For de to næste hyperparametre, *max_depth* og *min_rows*, der er med til at styre strukturen af hvert træ, ser vi for det meste værdier tæt på den øvre grænse i deres respektive interval. Det tyder på at en øgning af begge intervaller, kunne være en god idé. Det forsøger vi os med.

Vedrørende *mtries*, så ses det at det generelt er de lavere værdier i intervallet der ligger højest i tabellen. Det kunne måske være en fordel at rykke intervallet længere ned, men vi ser tiden an, og venter med at korrigere i dens interval indtil resultatet fra næste iteration.

De næste to, *nbins* og *ntrees*, er noget uklare, men et lille ryk opad i deres intervaller er måske værd at udforske.

Den øverste grænse på 0.8 for den sidste hyperparameter er forholdsvis underrepræsenteret i toppen af tabellen. Derimod er den lavere halvdel, 0.2 og 0.4, indeholdt i 14 af de 20 bedste modeller. Vi rykker intervallet for *sample_rate* ned.

Ud fra ovenstående overvejelser sætter vi intervallerne for næste iteration til

- *max_depth* (maksimale dybde for hvert træ) = 20, 24, ..., 36, 40
- *mtries* (antal tilfældigt trukne søjler for hvert niveau i hvert træ) = 10, 20, ..., 40, 50
- *min_rows* (mindste antal obs. i en leaf node for at splitte) = 20, 30, ..., 90, 100
- *nbins* (antal bins i histogrammet der splittes ud fra) = 20, 30, ..., 50, 60

- *ntrees* (antal træer) = 40, 60, ..., 120, 140
- *sample_rate* (% af obs. der skal bootstrappes) = 0.1, 0.2, ..., 0.5, 0.6
- *col_sample_rate_per_tree* (% af søjler der skal samples for hver træ) = 0.3, 0.4, ..., 0.9, 1

Denne kørsel resulterer i 59 forskellige modeller, hvilket er en del færre end den første iteration (se *Grid 2* figuren).

<i>col_sample_rate_per_tree</i>	<i>max_depth</i>	<i>min_rows</i>	<i>mtries</i>	<i>nbins</i>	<i>ntrees</i>	<i>sample_rate</i>	<i>mae</i>
1	36	70	10	50	120	0.5	0.0040881
0.7	40	40	10	50	80	0.6	0.00408868
0.7	40	50	30	60	140	0.2	0.00408913
0.9	40	80	50	30	140	0.3	0.00409193
0.5	40	60	30	40	120	0.2	0.00409194
1	32	40	40	50	120	0.1	0.00409221
0.4	32	30	10	20	120	0.5	0.00409344
0.3	24	100	20	40	100	0.3	0.00409373
0.5	40	90	30	20	60	0.2	0.00409407
0.6	28	40	10	40	140	0.6	0.00409431
0.9	40	90	30	30	60	0.4	0.00409448
0.9	32	70	30	20	60	0.4	0.00409472
0.3	32	50	40	40	80	0.1	0.00409546
1	32	100	10	40	60	0.3	0.00409573
1	40	50	20	60	80	0.2	0.00409634
0.9	40	60	40	50	40	0.3	0.00409649
1	24	100	10	40	140	0.4	0.00409682
0.3	24	90	10	40	100	0.5	0.00409687
1	36	80	30	60	60	0.4	0.004097
0.7	40	30	40	30	120	0.3	0.00409702

Random Forest – Grid 2

Vi holder os stadig til *max_runtime_secs* = 2000, men pga. specielt øgningen i intervallet for *max_depth* samt *ntrees*, er den gennemsnitlige træningstid for modellerne øget.

De nye intervaller har generelt resulteret i en bedre præstation, idet den bedste model har en *MAE* = 0.0040881. Samtidig er der kun 3 modeller fra forrige kørsel, der præsterer bedre end den dårligste af disse 20.

Vi kigger på hyperparametrene igen.

Det lader til, at udvidelsen af den nedre grænse for *col_sample_rate_per_tree* ikke rigtig gav pote, da de højere værdier stadig ser ud til at dominere. Beslutningen var nok en fejltagelse. Vi prøver istedet at lægge mere vægt på den øvre halvdel af intervallet, begyndende med 0.7, bl.a. da 13 af de 20 bedste modeller ligger i dette interval.

max_depth har samme tendens som tidligere, da de øvre værdier i høj grad er dominerende. Vi rykker dens interval igen en smule op.

På den anden side, er budskabet fra *min_rows* uklart, men dets interval er også forholdsvis stort. Det er måske en indikation på, at hyperparameteren ikke er så afgørende for vores data. Vi vælger dog at indsnævre det en smule og ser om den fortæller os noget nyt efter næste iteration.

Vedr. *mtries*, så ses det igen, at *mtries* = 10 er dominerende i tabellen. Vi vælger nu at handle på det, og rykker derfor hele intervallet ned.

Der er en vis indikation på, at *nbins* øvre værdier er de mest interessante. Vi rykker intervallet lidt op.

8 af de bedst præsterende modeller, har en *ntrees* værdi over 60. Værdierne 40 og 60 ekskluderes derfor.

Det nye interval for *sample_rate* er uklart. Vi lader det stå, og ser om det bliver mere klart om et øjeblik.

Intervallerne for iteration #3 sættes til

- *max_depth* (maksimale dybde for hvert træ) = 32, 35, ..., 47, 50
- *mtries* (antal tilfældigt trukne søjler for hvert niveau i hvert træ) = 7, 14, ..., 35, 42
- *min_rows* (mindste antal obs. i en leaf node for at splitte) = 40, 50, ..., 80, 90
- *nbins* (antal bins i histogrammet der splittes ud fra) = 35, 45, ..., 65, 75
- *ntrees* (antal træer) = 80, 95, ..., 105, 120
- *sample_rate* (% af obs. der skal bootstrappes) = 0.1, 0.2, ..., 0.5, 0.6
- *col_sample_rate_per_tree* (% af søjler der skal samples for hver træ) = 0.7, 0.75, ..., 0.95, 1

Denne kørsel resulterede i 29 modeller og forbedrede desværre ikke præstationen (se Grid 3). Den bedste model blandt de 20 er en smule dårligere end den tilsvarende model fra forrige iteration, mens den dårligste fra denne iteration er bedre end den dårligste fra forrige.

Det er måske tid til at ændre strategi, og indsnævre intervallerne forholdsvis meget for de fleste hyperparametre, i forsøget på at få nogle konsistent gode modeller i toppen.

col_sample_rate_per_tree	max_depth	min_rows	mtries	nbins	ntrees	sample_rate	mae
0.85	41	40	14	55	125	0.4	0.00408897
0.85	41	90	14	65	110	0.4	0.00408907
0.95	44	50	14	55	140	0.5	0.00408911
0.8	47	90	7	55	80	0.6	0.00409057
0.8	38	80	14	55	140	0.6	0.00409149
0.8	47	60	7	65	125	0.3	0.0040917
1	35	40	14	35	140	0.5	0.0040924
0.7	41	80	21	35	95	0.5	0.00409242
0.75	47	90	28	35	140	0.5	0.00409254
0.8	32	70	42	55	80	0.2	0.00409292
0.8	38	50	7	35	125	0.3	0.00409296
0.75	32	60	35	35	140	0.2	0.00409327
1	41	80	42	55	80	0.1	0.00409384
0.7	41	70	42	55	125	0.3	0.00409433
0.75	32	40	21	35	110	0.2	0.00409503
1	35	40	21	45	80	0.3	0.0040958
0.7	44	80	21	55	110	0.6	0.00409582
0.75	47	60	21	65	80	0.6	0.00409585
0.85	44	70	42	55	80	0.5	0.00409592
0.8	41	70	14	75	95	0.1	0.00409597

Random Forest – Grid 3

Begge grænseværdier for *col_sample_rate_per_tree* ser ud til at kunne udelukkes, da en af dem først forekommer i den 7. model.

For værdier af *max_depth* i fyrrerne ser modellerne ud til at præstere forholdsvis godt. Vi indsnævrer hyperparameteren, ved fortrinsvis at udelukke dens lavere værdier.

min_rows er stadig uklar. Vi vælger at lade den stå i dette fleksible interval.

Ændringen i *mtries* interval, sammen med de nye værdier for de andre hyperparametre, viser at det nok er i den lavere ende at hyperparameteren skal befinde sig. 7 af de bedste modeller indeholder $mtries < 20$. Vi handler på dette ved at fokusere på denne lavere ende i vores næste iteration.

nbins viser også forholdsvis klare signaler. Det er i den højere ende denne skal ligge – 55 lader til at være meget foretrukket. Vi vælger at lede omkring denne.

Af de forskellige værdier for *ntrees* er der ikke rigtig nogen der dominerer. Høje værdier for denne hyperparameter fører næsten altid til mere generaliserbare modeller, men medfører også en højere tidsomkostning. Som et kompromis, fastlåser vi denne til 125.

sample_rate indsnævres en smule fra begge sider, sådan at den ikke forstyrrer de andre hyperparametre for meget, da den for det meste ikke har været i stand til at bestemme sig.

Vi kører igen med følgende intervaller

- *max_depth* (maksimale dybde for hvert træ) = 38, 40, ..., 46, 48
- *mtries* (antal tilfældigt trukne søjler for hvert niveau i hvert træ) = 6, 9, ..., 15, 18
- *min_rows* (mindste antal obs. i en leaf node for at splitte) = 40, 50, ..., 80, 90
- *nbins* (antal bins i histogrammet der splittes ud fra) = 50, 53, ..., 62, 65
- *ntrees* (antal træer) = 125
- *sample_rate* (% af obs. der skal bootstrappes) = 0.3, 0.35, ..., 0.45, 0.5
- *col_sample_rate_per_tree* (% af søjler der skal samples for hver træ) = 0.75, 0.80, ..., 0.90, 95

Efter 2000 sekunder opnår vi 30 modeller og det ser ud til, at ændringen i strategi har resulteret i meget passende intervaller for hyperparametrene.

<i>col_sample_rate_per_tree</i>	<i>max_depth</i>	<i>min_rows</i>	<i>mtries</i>	<i>nbins</i>	<i>ntrees</i>	<i>sample_rate</i>	<i>mae</i>
0.75	46	50	9	50	125	0.35	0.00408487
0.75	46	60	6	65	125	0.5	0.00408548
0.85	44	70	15	59	125	0.3	0.00408862
0.8	38	60	15	59	125	0.35	0.0040888
0.8	46	50	9	53	125	0.3	0.0040889
0.9	38	90	9	56	125	0.45	0.00408892
0.9	46	80	12	65	125	0.3	0.00408913
0.9	48	50	15	50	125	0.4	0.00408914
0.9	46	50	15	50	125	0.45	0.00408942
0.8	48	70	6	59	125	0.4	0.00408987
0.85	42	80	15	50	125	0.45	0.00408996
0.85	48	80	18	50	125	0.5	0.00409001
0.95	42	40	9	65	125	0.45	0.00409001
0.9	48	80	9	50	125	0.4	0.00409004
0.85	46	50	18	62	125	0.5	0.00409014
0.85	42	60	18	50	125	0.4	0.00409027
0.75	40	90	9	62	125	0.35	0.00409045
0.85	44	60	6	59	125	0.5	0.00409055
0.8	42	40	15	65	125	0.5	0.00409064
0.85	44	90	12	50	125	0.35	0.00409069

Random Forest – Grid 4

Den bedste model har en $MAE = 0.00408487$. En helt fin forbedring ifht. den bedste fra forrige kørsel. Udover dette, er samtlige af disse 20 modeller bedre end den bedste model fra første kørsel. Endvidere, er der kun 4 modeller fra forrigere iteration der overgår den dårligste i denne tabel.

Vi kan i princippet fortsætte denne proces i rigtig lang tid, men vi vælger at stoppe her, da det lader til at vi har fundet intervaller for vores hyperparametre, der er specifikke nok til at levere

modeller med forholdsvis lave *MAE*. Forhåbenligt er disse intervaller alligevel brede nok til at tage højde for potentielle ændringer i kryptovalutamarkedet.

For de to resterende *base-learners*, *GBM* og *ANN*, udføres deres *random-grid-search* på tilsvarende vis. Processen er udeladt her, men vi henviser til bilag for tabeller af de bedste 20 modeller, udledt af de endelige hyperparameter intervaller vi har fundet for de to *base-learners*.

De passende intervaller for *GBM* fandt vi til at være:

- *max_depth* (maksimale dybde for hvert træ) = 30, 35, ..., 45, 50
- *ntrees* (antal træer) = 75
- *col_sample_rate* (antal tilfældigt trukne søjler for hvert niveau i hvert træ) = 0.2, 0.4, ..., 0.8, 1
- *min_rows* (mindste antal obs. i en leaf node for at splitte) = 60, 70, ..., 100, 110
- *learn_rate* (hvor meget der lærer fra træ til træ) = 0.004, 0.008, ..., 0.026, 0.032
- *learn_rate_annealing* (en faktor ganget på *learn_rate*, der resulterer i at *learn_rate* reduceres fra træ til træ) = 0.985, 0.99, 0.995
- *sample_rate* (% af obs. der skal bootstrappes) = 0.05, 0.1, 0.15, 0.2

For *ANN* er disse givet med:

- *hidden* (størrelse af hvert hidden layer) = (100,100,100), (200,200,200), (300,200,100), (100,300,100)
- *loss* (hvilken metric vægtene fittes ud fra i Back-propagation processen) = "Absolute"
- *L2* (styrken af L2-regularisering) = $1e-4$, $1e-5$, $1e-6$, $1e-7$, $1e-8$
- *epochs* (maksimalt antal gange train strømmes igennem netværket) = 20
- *metric* (hvilken metric early-stopping kriteriet er baseret på) = "mae"
- *rho* (minder om *GBMs learn_rate_annealing*) = 0.8, 0.9, 0.99
- *epsilon* (der forhindrer at der divideres med 0 i vægt opdateringsprocessen (ADADELTA)) = $1e-8$, $1e-9$, $1e-10$
- *activation* (aktiveringsfunktion for hver neuron) = "RectifierWithDropOut"
- *hidden_dropout_ratios* (andelen af neuroner i hvert hidden der layer skal smides væk) = (0.3, 0.3, 0.3), (0.4, 0.4, 0.4), (0.5, 0.5, 0.5), (0.6, 0.6, 0.6)

6.2.3 Opsummering

Vi har nu, for samtlige *base-learners*, fundet nogle passende intervaller for hyperparametrene, men der er stadig nogle relevante sammenligninger at udføre.

Det er nemlig oplagt, at se hvorvidt vores *grid-search* proces rent faktisk har ført til bedre resultater på *test* – dvs. om de på dette data har lavere *MAE* end de 4 *base-learners*, fittet med *H2O*s standard hyperparameterverdier – det vi kalder for *baseline* modeller.

Vi vil selvfølgelig også sammenligne vores *base-learners* med vores *benchmark* modeller, *zero-prediction* og *mean-prediction*.

Vi fitter de 4 *baseline* modeller på *train*, med respektive standard hyperparametre, og finder *MAE* ved at prædiktere på *test*. Vi udregner også *MAE* for de to *benchmark*-modeller. Dette sammenligner vi med vores bedst præsterende *base-learners* for denne *test* periode. Se nedenstående figur.

Test MAE			
	Baseline base-learners	Optimale base-learners	
EN	0.00413416	0.00411276	
RF	0.00414886	0.00408487	
GBM	0.00413936	0.00408544	
ANN	0.00414326	0.00406376	
	Zero-prediction		Mean-prediction
Benchmark modeller	0.00412427		0.00412063

Vi er umiddelbart tilfredse med resultat. Det ser ud til at vores *grid-search* har været belønnende, da samtlige af vores optimale *base-learners* slår deres respektive *baseline* modeller, og endvidere slår de alle de andre *baseline* modeller.

Det næste vi bemærker, er at ingen *baseline* model slår hverken *zero-prediction* eller *mean-prediction*.

Derimod slår alle vores *base-learners* de to *benchmark* modeller, takket være vores *grid-search* proces. Det skal dog siges, at *Elastic Net* er forholdsvis tæt på *MAE* for *benchmark* modellerne.

RF og *GBM* ses at være tætte konkurrenter, hvor *RF* dog har overhånden, mens vores *Artificial Neural Network* er langt den bedste af modellerne. *ANN*s roses ofte for at være velegnede til rigtig stort data, hvilket vi i den grad arbejder med her.

Den umiddelbare reaktion efter at observere forholdsvis gode resultater som disse på *test*, kunne være at stille sig tilfreds, men da det er prædiktionskraften af hver *base-learner* på fremtidigt data vi er interesserede i, skal disse resultater valideres på hidtil uset data. I vores *walk-forward-optimization* i kapitel 7 udfører vi denne validering på vores *validation* data.

Med dette er vores søgning af passende hyperparameter intervaller for vores *base-learners* afsluttet. Det næste skridt er, at benytte prædiktioner fra vores 4 bedste *base-learners* som *features* i vores 4 *meta-learners*, for at finde frem til passende hyperparameterintervaller for vores *meta-learners*.

6.3 Meta-learners

I dette underkapitel kommer vi til at se nærmere på vores *meta-learners*, der også skal bruges i vores *walk-forward-optimization*. Modeltyperne vi anvender for disse 4 *meta-learners*, er de samme typer vi har anvendt som *base-learners*.

Disse skal selvfølgelig optimeres, dog ikke i den forstand, at optimale hyperparameterverdier findes, men istedet finder vi passende intervaller for vores *meta-learners*, ligesom vi gjorde for vores *base-learners*. Dvs. at vi udfører *random-grid-search* for 3 af disse og *Cartesian-grid-search* for *Elastic Net meta-learner*, sådan at de også passer ind i rammerne for vores *walk-forward-optimization*.

Endeligt ser vi på hvordan den bedste *meta-learner* af hver type præsterer på *test*, og forholder dens resultater med den tilsvarende *baseline meta-learner* (med *baseline* menes der en *meta-learner* med standard *H2O* hyperparametre), men også de andre bedste *meta-learners*.

De passende hyperparameterintervaller for vores *meta-learners* bliver fundet som følger:

1. For vores 4 bedste *base-learners* udføres der *5-fold* krydsvalidering på *train*, og alle *out-of-sample* prædiktioner gemmes.
2. De 4 typer af *meta-learners* fittes nu med *out-of-sample* prædiktionerne fra forrige punkt som input, og de tilsvarende afkastverdier som respons, med forskellige kombinationer af hyperparametre der ligger indenfor de hyperparameterintervaller vi afprøver.
3. For hver kombination der afprøves, bruger vi de omtalte *base-learners* prædiktioner på *test* som *input* og der prædikteres herefter. Passende hyperparameterintervaller evalueres ud fra *test MAE*.

I det forrige underkapitel fik vi vist, i detaljer, hvordan vi arbejder os igennem et *grid-search*. Vi har derfor valgt at udelade denne proces, og henviser istedet til bilag for tabeller af de bedste 20 modeller for hver type *meta-learner*, udledt af de endelige hyperparameter intervaller vi har fundet. Dog røres der overfladisk ved de mest interessante ændringer for intervallerne for *meta-learners*, ifht. intervallerne for *base-learners*.

Intervallerne for samtlige *meta-learners* er som følger:

Elastic Net meta-learner:

- α (L1 og L2 fordeling) = 0, 0.05, ..., 0.95, 1
- λ (regulariseringsstyrken) = $2e - 3$, $3e - 3$, ..., 0.019, 0.020

Bemærk hvordan intervallet for λ ikke indeholder de tilsvarende lave værdier som for den optimale *base-learner* for denne periode. Antallet af features er kun 4 for denne udgave af *Elastic Net*, og den lader til foretrække en kraftigere regularisering af koefficienterne ifht. tidligere.

Siden modeltypen er lynhurtig at fitte for rigtig mange kombinationer af hyperparameterene,

hvilket udføres endnu hurtigere med så få *features*, gav vi *alpha* muligheden at dække alt intervallet fra 0 til 1. I den forstand er det ikke sikkert at den kan omtales som en *Elastic Net* fra vindue til vindue, i vores eventuelle *walk-forward-optimization*, da de to absolutte grænseværdier for hyperparameteren er indeholdt i intervallet. Ikke desto mindre omtaler vi den stadig som *Elastic Net*.

Random Forest meta-learner:

- *max_depth* (maksimale dybde for hvert træ) = 4, 8, ..., 16, 20
- *mtries* (antal tilfældigt trukne søjler for hvert niveau i hvert træ) = 1, 2, 3, 4
- *min_rows* (mindste antal obs. i en leaf node for at splitte) = 10, 15, ..., 25, 30
- *nbins* (antal bins i histogrammet der splittes ud fra) = 30, 40, 50
- *ntrees* (antal træer) = 100
- *sample_rate* (% af obs. der skal bootstrappes) = 0.8, 0.9, 1
- *col_sample_rate_per_tree* (% af søjler der skal samples for hver træ) = 1

Intervallerne for denne *meta-learner* fraviger også en del fra den tilsvarende *base-learner*, bl.a. igen takket være de drastisk lavere antal *features*. Det lader ikke til at være nødvendigt med den samme store dybde længere, og nu kan selvfølgelig maksimalt afprøves 4 *features* for hvert *split*. Endvidere skal den også for hvert træ *bootstrappe* noget mere af *train* end sin *base-learner*.

Gradient Boosting Machine meta-learner

- *max_depth* (maksimale dybde for hvert træ) = 4, 5, ..., 9, 10
- *ntrees* (antal træer) = 150
- *col_sample_rate* (antal tilfældigt trukne søjler for hvert niveau i hvert træ) = 1
- *min_rows* (mindste antal obs. i en leaf node for at splitte) = 100, 110, 120, 130, 140
- *learn_rate* (hvor meget der lærer fra træ til træ) = 0.005, 0.006, ..., 0.009, 0.01
- *learn_rate_annealing* (en faktor ganget på *learn-rate*, der resulterer i at *learn_rate* reduceres fra træ til træ) = 0.97, 0.972, ..., 0.988, 0.99
- *sample_rate* (% af obs. der skal bootstrappes) = 0.9, 0.92, ..., 0.98, 1

Det er ingen overraskelse, at vi kan observere nogenlunde de samme ændringer i intervallerne for *GBM* ifht. sin *base-learner*, som vi gjorde for *RF* ifht. sin *base-learner*, da algoritmerne har en del til fælles.

Udover dette foretrækker *learn-rate* at ligge noget lavere, sammen med en *learn_rate_annealing* der også har mulighed for ligge noget lavere. Sammen medfører de, at modellen forbedres mindre og mindre fra træ til træ end tidligere. Dette opvejes dog med, at antallet af træer er sat til at være dobbelt så højt.

Artificial Neural Network meta-learner

- *hidden* (antal neuroner i hvert hidden layer) = (200,30,15), (300,40,25), (80,40,20), (50,100,50)
- *loss* (hvilken metric vægtene fittes ud fra i Back-propagation processen) = "Absolute"
- *L2* (styrken af L2-regularisering) = $2e - 5, 4e - 5, 6e - 5, 8e - 5, 1e - 4$
- *epochs* (maksimal antal gange train strømmer igennem netværket) = 1, 2, 3, 4, 5
- *metric* (hvilken metric early-stopping kriteriet er baseret på) = "mae"
- *rho* (minder om GBMs *learn_rate_annealing*) = 0.99
- *epsilon* (forhindrer at der divideres med 0 i vægt-opdateringsprocessen (ADADELTA)) = $1e - 8, 1e - 9, 1e - 10$
- *activation* (aktiveringsfunktion for hver neuron) = "RectifierWithDropOut"
- *hidden_dropout_ratios* (andelen af neuroner i hvert hidden der layer skal smides væk) = (0.3, 0.3, 0.3), (0.5, 0.5, 0.5), (0.7, 0.7, 0.7)

Det første man kan bemærke ved denne *meta-learner* ifht. sin *base-learner*, er at der generelt kræves færre neuroner i er hvert *hidden-layer*. Ændringen skal nok igen forklares med ændringen i antallet af *features*.

Den anden nævneværdige ændring er de mulige værdier for *epochs*. *ANN meta-learner* har ikke brug for det samme antal strømme af *train* igennem netværket, i dannelsen af vægtene, som tidligere.

6.3.1 Opsummering

Som det sidste, ser vi på hvorvidt udledningen af hyperparameterintervallerne har været gunstig. Vi ser på *MAE* for hver optimal *meta-learner*, samt *MAE* for de tilsvarende *baseline meta-learners*. Dette er givet i nedenstående figur.

Test MAE			De valgte hyperparameterintervaller for samtlige <i>meta-learners</i> er i stand til at producere bedre <i>MAE</i> præsterende modeller end deres tilsvarende <i>baselines</i> . For 3 af vores optimale <i>meta-learners</i> ses forskellen at være forholdsvis høj, mens <i>RF</i> kun klarer sig en smule bedre end sin <i>baseline</i> .
	Baseline meta-learners	Optimale meta-learners	
EN	0.00417048	0.00408661	
RF	0.00422311	0.00415307	
GBM	0.00419331	0.00408583	
ANN	0.00412385	0.00406952	
	Zero-prediction	Mean-prediction	
Benchmark modeller	0.00412427	0.00412063	

Forholder vi de optimale *meta-learners* til zero- og mean-prediction, så har *RF* sværere med at ramme de sande værdier end de to *benchmark* modeller, mens de 3 resterende klarer sig

udmærket. Det kunne tyde på, at *Random Forest*, som model, ikke er så velegnet til brug som *meta-learner*. En anden mulighed er, at vi ikke har været grundige nok i udførelsen af dens *random-grid-search* proces for hyperparameter intervallerne. Dens præstation i vores *walk-forward-optimization* bliver alligevel interessant at observere.

Den bedste af disse 4 *meta-learners* er *ANN meta-learner* – ligesom denne type model også præsterede bedst som *base-learner*.

Et interessant resultat i tabellen, er den optimale *Elastic Net meta-learners* præstation, da den overbevisende slår de to *benchmark*-modeller.

Vi husker på, at blandt *base-learners* var *Elastic Net base-learner* den der klarede sig dårligst, og slog akkurat de to *benchmark*-modeller. Ifht. denne type af model, hyldes de andre typer, der benyttes i denne afhandling, også ofte for at være i stand til at præstere godt på højdimensionelt data, hvilket samlet set også gælder for de *features* der er blevet brugt for vores *base-learners*. Det lader dog til, at en *stacking* tilgang for denne type model, hvor vi altså flytter vores *feature space* fra 123 tekniske indikatorer til 4 optimerede prædiktionsserier, har den største forholdsvis gevinst for denne type af model.

Med dette er vi kommet til vejs ende for *Grid-Search* kapitlet.

Vi har udledt hyperparameter intervaller for *base-* og *meta-learners*, der, for det meste, lader til at være passende at benytte som input i den store finale – nemlig *walk-forward-optimization*, som vi går i dybden med i næste kapitel.

7 Walk-forward-optimization

Igennem det sidste kapitel har vi udledt vores tilsyneladende passende hyperparameterintervaller for vores *base-* og *meta-learners*.

Selvom vi har fundet en række modeller, der ud fra *test MAE* ikke præsterer på højde med andre, er vi alligevel interesserede i at validere alle vores *8 learners* på hidtil uset data – både statistisk, samt afkastmæssigt, i et *trading* setup.

Som nævnt i afgrænsningen, var den oprindelige idé, at benytte en af vores modeller som prædiktionsværktøj i et *real-time trading* system. Hvis den statistiske model var en *meta-learner*, kunne det i algoritmisk forstand udføres således for hvert minut:

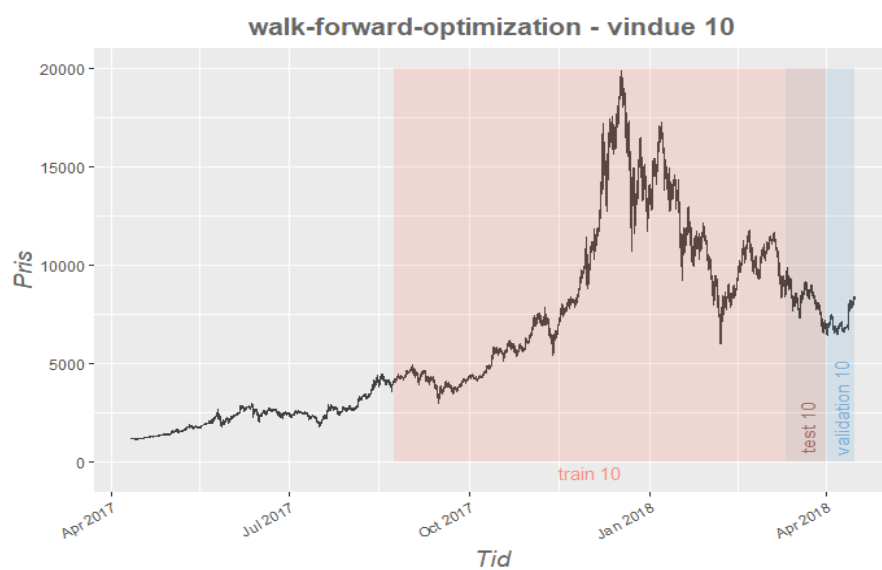
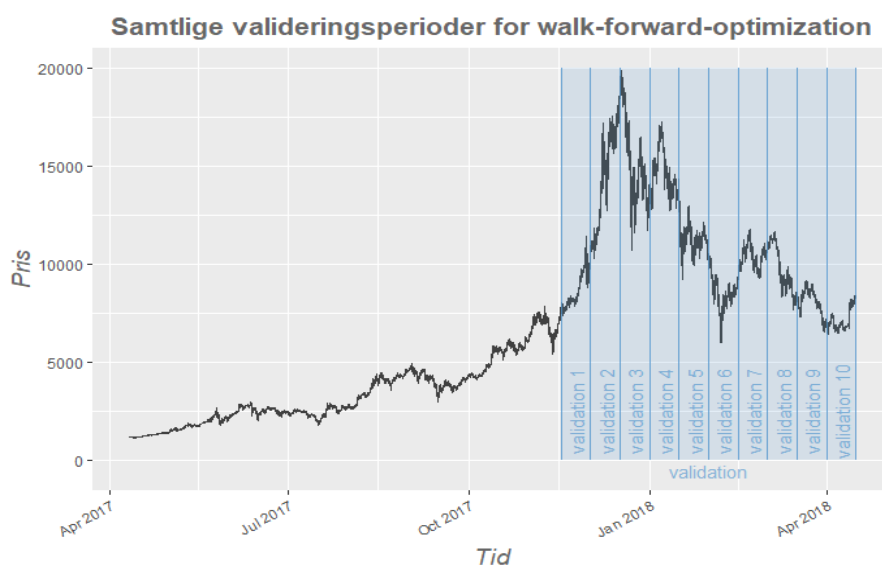
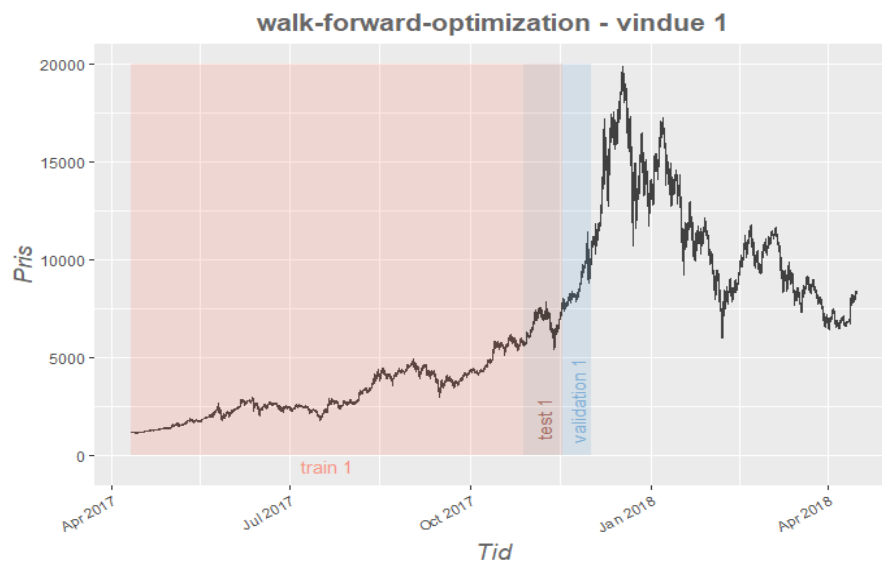
- *OHLCV*, ordrebogen, samt alle *matched* ordrer for minuttens *scrapes*.
- *features* for *OHLCV* udregnes.
- *base-learners* prædikterer på disse, og prædiktionerne benyttes som inputs i *meta-learners*.
- *meta-learners* udfører den endelige prædiktion.
- Prædiktionen fodres til en *bot*, der programmeres til at handle ud fra denne. Ordrebogen, samt de *matched* ordrer, benyttes af *botten* til at placere ordrer fordelagtigt.

Ved at udføre ovenstående ville det være muligt at bekræfte, om en statistisk tilgang til kryptovalutamarkedet kan være profitabel. Men som vi også kom ind på, så blev det nødvendigt at begrænse os til at opbygge og evaluere vores *trading* strategi på allerede indsamlet data. I den forstand er det altså kun hidtil indsamlet *OHLCV* data vi gør brug af.

Der er en del overvejelser vi må lave os, for at konstruere en *trading* strategi. Disse dækker bl.a. overvejelser omkring hvilke *trading* signaler strategien skal indeholde og hvilke handlinger vores strategi skal kunne foretage sig. Vi er også bevidste om den mulige fare ved at bruge forældet data i optimeringen af handlingsstrategien (dette omfatter både *trading* signaler samt de statistiske modellers hyperparametre).

I en *trading* sammenhæng, kaldes den måde vi kommer til at optimere vores strategi på, for *walk-forward-optimization*. Denne metode tager højde for den sidste af de nævnte overvejelser.

Vores format af *walk-forward-optimization* kommer til at indeholde 10 vinduer, der samlet set strækker sig fra 11.04.2017 til 15.04.2018. Hvert vindue består af 235 dage, hvoraf de første 220 bruges til at optimere strategien, i.e. *in-sample* data, mens de resterende 15 dage (hidtil nævnt *validation*) bruges som *out-of-sample-data*, til at validere den optimale strategi. Af de 220 dage der benyttes til optimering af strategien allokeres de 200 første dage (hidtil



nævnt *train*) til at fitte vores modeller. De sidste 20 (hidtil nævnt *test*) bruges til at finde optimale hyperparametre, ud fra de passende intervaller vi fandt i vores *Grid-search* kapitel, samt til at finde optimale *trading* signaler, som vi kommer mere ind på i vores redegørelse af den samlede handlingsstrategi.

Det data (*train* og *test*) vi har arbejdet med hidtil, benytter vi sammen med *validation* som det første vindue i vores *walk-forward-optimization*, og betegner disse som *train 1*, *test 1* og *validation 1*. Se den første figur på forrige side.

Den næste figur illustrerer valideringsperioderne, fra vindue 1 til og med 10.

Og da *train* og *test* efter hvert vindue, rykkes med længden af valideringsperioden, er de rykket med 9 gange 15 dage i det 10. vindue, som illustreret i den 3. og sidste figur.

7.1 Redegørelse af vores *trading* strategi

I dette afsnit går vi ned i detaljer omkring vores *trading* strategi, der løber over alle vinduer i vores *walk-forward-optimization*.

Trading strategien kan for hvert vindue opdeles i 3 dele:

- 1 Træning og optimering af statistiske modeller, samt prædiktion på *test* og *validation*.
- 2 Optimering af *trading* signaler baseret på prædiktioner på *test*.
- 3 Validering af vores *trading* strategi baseret på prædiktioner på *validation*, samt optimale *trading* signaler.

Vi lægger ud med den første del.

Første del

For at opsummere en smule og for at give indblik i, hvilken rolle det statistiske arbejde vi har gjort hidtil har i vores *trading* strategi, ser vi nærmere på hvad der præcis sker i hvert vindue i den første del af vores *trading* strategi.

Den første del af vores *trading* strategi er altså, for hvert vindue, som følger:

- 1 OHLCV data indrages, *features* og respons konstrueres og splittes i *train*, *test* og *validation*.
- 2 De 4 typer af *base-learners* fittes på *train* med forskellige kombinationer af hyperparametre der ligger indenfor de passende hyperparameter intervaller vi fandt i *Grid-search* kapitlet.

3. For hver type af *base-learner*, udvælges den hyperparameterkombination der fører til lavest *MAE* ved prædiktion på *test*. Prædiktionerne fra den bedste model af hver type gemmes.
4. For disse 4 bedste *base-learners* udføres der *5-fold* krydsvalidering på *train*, og alle *out-of-sample* prædiktioner gemmes.
5. De 4 typer af *meta-learners* fittes nu med *out-of-sample* prædiktionerne fra forrige punkt som *features*, og de tilsvarende afkastværdier som respons, med forskellige kombinationer af hyperparametre der ligger indenfor de passende hyperparameter intervaller vi fandt i *Grid-search* kapitlet.
6. For hver type af *meta-learner*, bruges prædiktionerne fra punkt 3 som *features* og der prædikteres på den tilsvarende *test* afkastværdi, og den hyperparameterkombination der fører til lavest *MAE* udvælges.
7. Prædiktionerne fra punkt 3 og 6 gemmes som led i at finde vores optimale købs- og salgssignaler for hhv. de optimale *base-learners* og optimale *meta-learners*.

Før vi kan redegøre for hvad der sker efterfølgende, forklarer vi den næste del af vores *trading* strategi.

Anden del

Her går vi ned i detaljer i omkring hvilke antagelser strategien er omfattet af, hvilke handlingsmuligheder den har, samt hvordan vi vælger at definere strategiens købs- og salgssignaler.

Trading strategien er omfattet af to antagelser:

- Strategien handler altid til *Close* prisen, inden næste *Close* pris observeres (grafisk, vil dette dog ses som øjeblikkeligt).
- Strategiens ordrer fyldes altid.

Strategiens mulige handlinger er meget simple og baseres på to forskellige signaler – et købs- og et salgssignal. Hvis ingen signaler er aktive er strategien passiv.

For hvert minut kan strategien

- købe *BTC* for hele dens *USD* beholdning.
- sælge hele dens *BTC* beholdning for *USD*.
- være passiv.

Den må altså hverken udføre såkaldte *gearede* handler eller gå *short*.

I princippet betyder dette, at porteføljeværdien altid følger samme afkastrate som *BTC* hvis strategien holder *BTC*. Hvis strategien ikke holder *BTC* ændrer porteføljeværdien sig selvfølgelig ikke.

Den kan altså, for hver handel, ikke have større afkastrate end afkastraten for *BTC* over en handelsperiode, og ej heller have et positivt afkast, hvis *BTC* prisen falder over handelsperioden.

En handelsperiode er her defineret til at starte ved tidspunktet for et køb af *BTC* og slutte ved det efterfølgende salgstidspunkt.

Et vigtigt element, som strategien skal tage hensyn til, er transaktionsomkostninger.

Disse er sat til 0.05%, og gælder for både købs- og salgsordrer.

Hvis strategiens porteføljeværdi er lig *1000 USD*, og strategien køber *BTC*, så falder værdien af porteføljen øjeblikkeligt til *999.5 USD*, og *BTC* beholdningen bliver $\frac{999.5 \text{ USD}}{\text{BTC/USD}}$, så for $\frac{\text{BTC}}{\text{USD}} = 1000$ ville *BTC* beholdningen være *0.9995*.

Princippet gælder som sagt også når vi sælger.

Der skal også indgå købs- og salgssignaler i strategien. Disse to signaler baseres på prædiktioner fra en statistisk model og optimeres over *test* perioden.

Nedenstående pseudo kode, giver et overblik over den funktion vi benytter, for at finde frem til den optimale kombination af købs- og salgssignaler for strategien:

#####Pseudo code#####

Funktion: Backtest3:

Input: wallet = 1000 USD, nominel *BTC* pris, prædiktioner af *BTC* afkast 15 perioder frem, købs- og salgssignal, samt transaktionsomkostninger.

case (1): prædiktionen er større end købssignalet og der holdes ikke *BTC*
case (2): prædiktionen er lavere end salgssignalet og der holdes *BTC*
case (3): prædiktionen er ikke større end vores købsignal og der holdes ikke *BTC*
case (4): prædiktionen er ikke lavere end vores salgssignal og der holdes *BTC*

For hver prædiktion gør:

Hvis (1):

* køb *BTC* og udregn hvor mange *BTC* du har råd til, når transaktionsomkostninger er trukket fra, og registrer antallet på følgende måde:

(USD i porteføljen * (100 - transaktionsomkostninger i %) / 100) / nominel pris på *BTC*

* sæt antal USD i porteføljen til 0

* udregn nu porteføljeværdien, som: antal *BTC* * prisen på *BTC*

Hvis (2):

* salg *BTC* og udregn antal USD efter transaktionsomkostninger er trukket fra:

(nominel pris på *BTC* * antal *BTC* i porteføljen * (100 - transaktionsomkostninger i %) / 100)

* sæt antal *BTC* i porteføljen til 0

* sæt porteføljeværdien = antal USD

Hvis det er det første salg af *BTC*:

* udregn første afkast som: (porteføljeværdi - wallet)/wallet

Ellers

* udregn afkast som:

(nuværende porteføljeværdi - porteføljeværdien(efter sidste salg)) / porteføljeværdien(efter sidste salg)

Hvis (3):

* sæt porteføljeværdien lig med antal USD i porteføljen

Hvis (4):

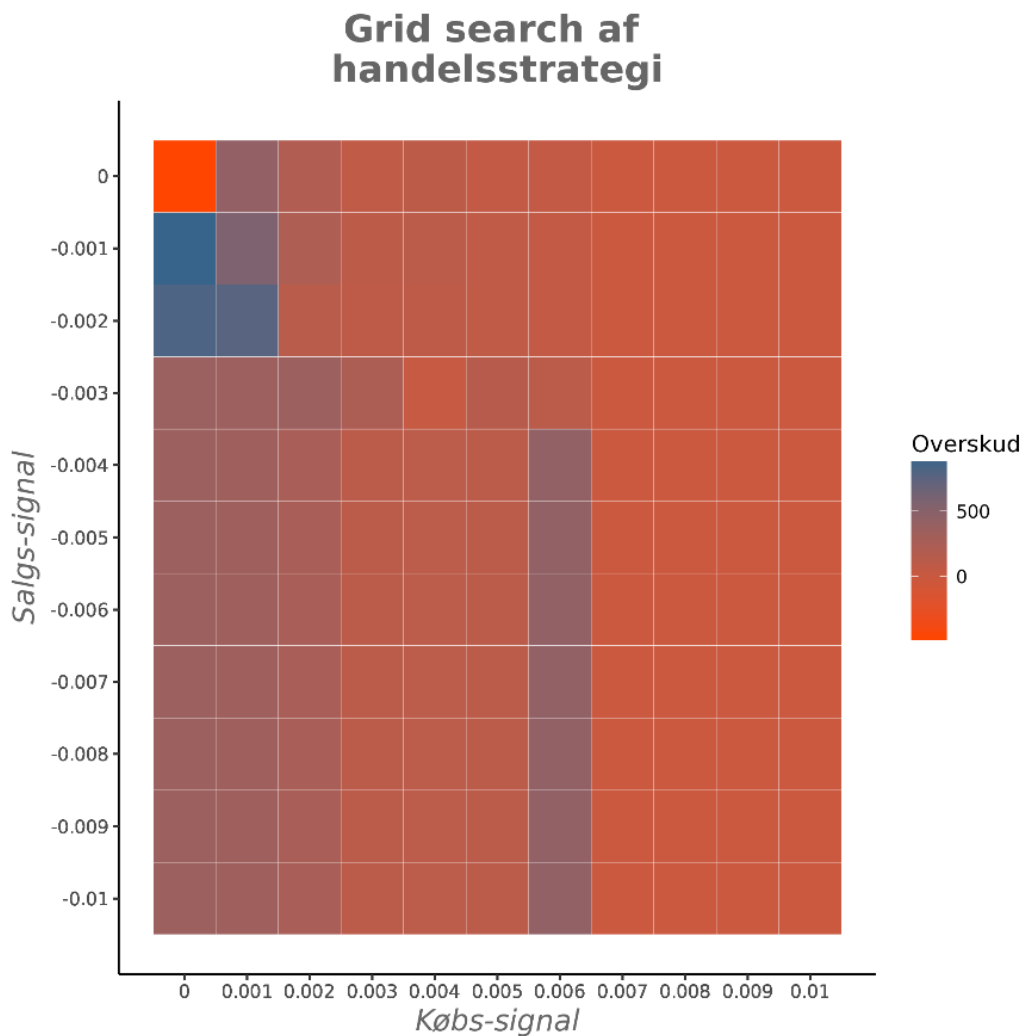
* sæt porteføljeværdien = antal *BTC* i porteføljen * nominel pris af *BTC*

Returner:

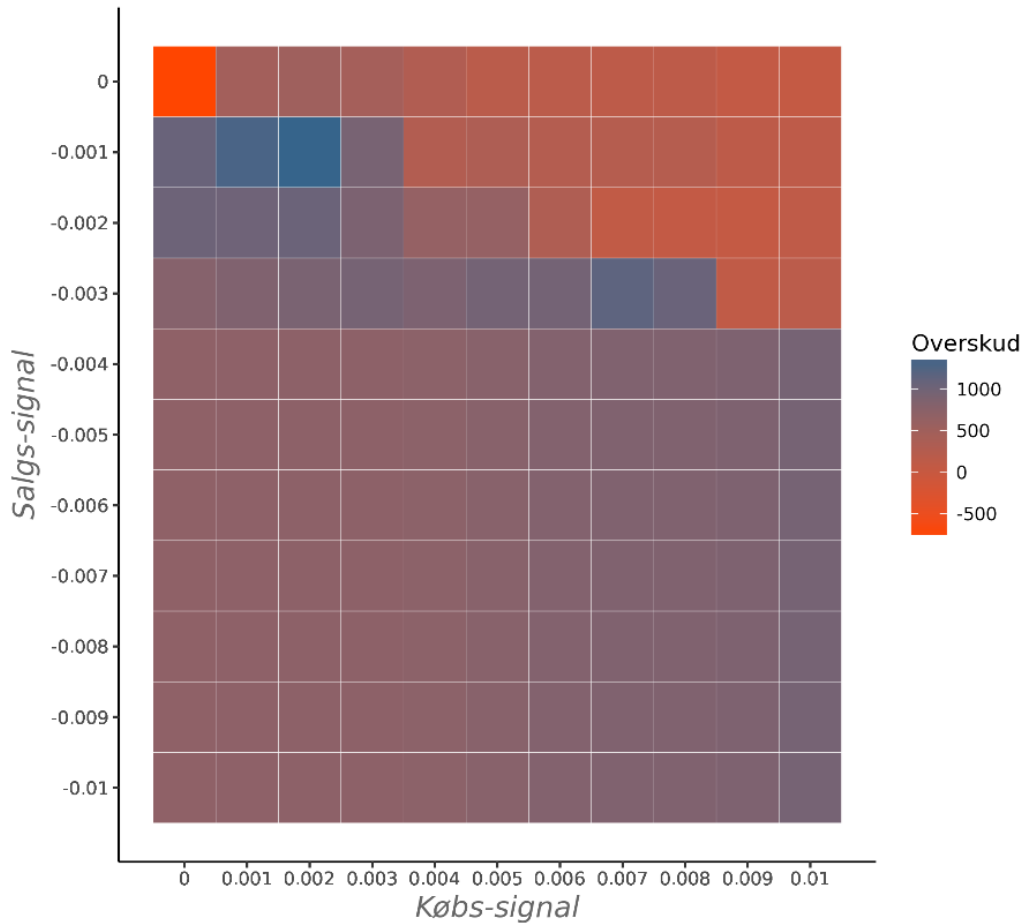
porteføljeværdien over alle vores prædiktioner, købs- og salgspriser, samt afkast for hver handel

I hvert vindue *loopes* der 121 gange over *Backtest3* funktionen, der for hvert loop får en kombination af købs- og salgssignaler som input, som specificerer hvor høj, eller lav, en models 15 minutters prædiktion af *BTC* afkastet skal være før strategien enten køber eller sælger. Slutværdien af porteføljen, for hver kombination af købs- og salgssignaler, bliver gemt for at afgøre hvilken kombination af disse signaler giver det bedste afkast. Alt dette baseres på prædiktioner på *test* perioden. Dette bliver altså gjort for alle vores statistiske modeller.

På denne side, samt næste side, ses to *heatmaps* der illustrerer optimeringen af købs- og salgssignaler for *ANN meta-learner* strategien i hhv. første og andet vindue i vores *walk-forward-optimization* ("*ANN meta-learner strategien*" refererer til den strategi der består af prædiktioner fra *ANN meta-learner*). Jo mere blå farven er, jo højere ville afkastet over *test* perioden have været. I venstre *heatmap* ser vi, at et købsignal på 0 og et salgssignal på -0.001 havde optimeret afkastet. Det fortæller, at det højeste afkast, strategien ville kunne opnå over



Grid search af handelsstrategi



test perioden, er hvis den køber hver gang modellen prædikerer et afkast på over 0% og sælger hver gang den prædikerer et fald på mere end 0.1%. I vindue 2 af *walk-forward-optimization* (figuren øverst på denne side), ser vi at strategien her vil være lidt mere sikker på et overskud, før vi skal købe (et prædikeret afkast på over 0.2%), mens det optimale salgssignal stadig er et prædikeret fald på over 0.1%.

Med dette, er beskrivelsen af anden del af vores *walk-forward-optimization* – nemlig at finde optimale *trading* signaler for vinduerne – udført.

Den tredje, og sidste del af vores *walk-forward-optimization*, er at validere strategien på *out-of-sample data* – nemlig *validation*.

Tredje del

Når den kombination af købs- og salgssignaler, der giver det største afkast på *test* er fundet, bruges modellen der tilhører strategien, samt den optimale kombination af *trading* signaler, til at prædiktere og handle over *validation* perioden.

Dette bliver gjort med vores *Backtest4* funktion, der udregner porteføljeværdien for hvert minut af valideringsperioden.

Backtest4 er i store træk den samme som *Backtest3*. Forskellen ligger i, at *Backtest4* tager højde for om strategien holder *BTC* eller *USD* fra forrige vindue (husk at porteføljen enten er udgjort af 100% *BTC* eller 100% *USD*), således at strategien f.eks. ikke kan begynde med at købe *BTC* i andet vindue, hvis der holdes *BTC* sidst i første vindue og vice versa.

Backtest4 indeholder også en mekanisme, der holder styr på porteføljeværdien efter seneste salg af *BTC*, selvom salget blev eksekveret i et foregående vindue. Det medfører, at den altid kan udregne afkastet for en handel der har starttidspunkt i et foregående vindue. Husk, at afkastet udregnes baseret på porteføljeværdien efter sidste salg og porteføljeværdien efter nuværende salg.

Wallet, er ligesom i *Backtest3*, et input i *Backtest4*. Den starter med at være 1000 *USD* i første vindue, og opdateres i efterfølgende vinduer, til at være lig med den sidste porteføljeværdi fra det foregående vindue.

Før vi begynder at grave i de endelige resultater for de 8 forskellige *trading* strategier, beskriver vi som det sidste, hvordan den type figur der illustrerer udviklingen af porteføljeværdien er opbygget. Endeligt fortolker vi helt kort på denne.

Disse figurer er i realiteten produktet af *Backtest4*.

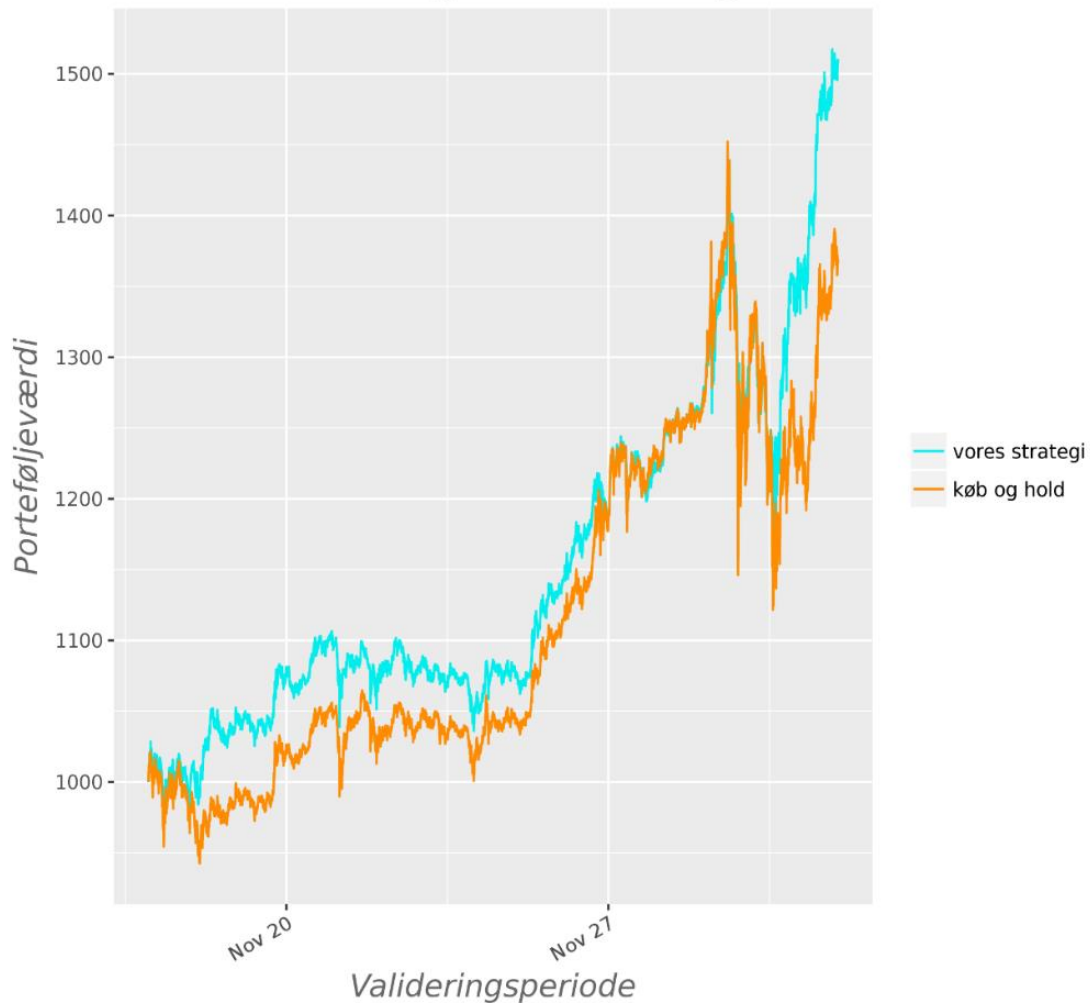
Vi ser på porteføljeværdien for *ANN meta-learner* strategien over *validation* perioden for det første vindue (se figuren på næste side).

Vindue 1 begynder med *Close* prisen for 17.11.2017 00:00:00 og slutter ved *Close* prisen for 01.12.2017 23:44:00.

Udover strategiens porteføljeværdi, indikeret med den blå linje, er en køb-og-hold strategi indtegnet. Denne benyttes som en *benchmark*-strategi i samtlige af disse porteføljeværdi figurer.

Køb-og-hold strategien starter med at holde et antal *BTC* der svarer til en porteføljeværdi på 1000 *USD*. Da den, som navnet antyder, ikke sælger, så reflekterer dens udvikling nøjagtigt afkastraten for *Close* prisen over perioden.

Vores optimale strategi vs. køb og hold strategi



I dette eksempel ses det, at vores strategi giver et højere afkast over perioden, end en køb-og-hold strategi. Køb og hold strategien begynder i midten af perioden at give højt afkast, som jo betyder at *Close* prisen stiger over denne periode. Vores strategi ser ud til at have svært med at følge køb og hold strategien de dage, hvor *Close* prisen stiger meget og hvor den lader til at have lav kortsigtet volatilitet. Derimod tyder det på at vores strategi generelt klarer sig bedre end køb og hold strategien i perioder med høj kortsigtet volatilitet, som kan anes fra de sidste dage i perioden.

Vi har nu redegjort for den tredje, og sidste del, af vores *trading* strategi.

7.2 Resultater fra *walk-forward-optimization*

Som nævnt tidligere i kapitlet laver vi et *grid-search* for alle vores *base-learners* og *meta-learners* for hvert vindue. For *Elastic Net* *base-* og *meta-learner* bruges som sagt et *Cartesian grid search*, med hyperparameterintervallerne der blev specificerede i kapitel 6. For *base-* og *meta-learners* *RF*, *GBM* og *ANN* bruges et *random-grid-search* over deres intervaller, der blev specificerede i kapitel 6.

Tiden der bruges på *random-grid-search* for hvert vindue sættes til:

- 3000 sekunder for *RF*, *GBM* og *ANN base-learner* (1000 sekunder for hver)
- 750 sekunder for *RF*, *GBM* og *ANN meta-learner* (250 sekunder for hver)

Vi bruger færre sekunder på vores *meta-learners* da det er meget hurtigere at fitte hver kombination af hyperparametre, end det er for vores *base-learners* (specielt på grund af, at vores *meta-learners* kun indeholder 4 *features*). Udover dette, så er det nok svært at få gode modeller (og et godt afkast for strategien), hvis ikke vi har rimelig gode hyperparametervalg for vores *base-learners* (da deres prædiktioner jo også skal bruges som *features* i vores *meta-learners*).

Som blev vist i de to *heatmap* figurer i underkapitel 7.1, så sættes intervallerne for købs- og salgssignalet til hhv.

- 0, 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01
- 0, -0.001, -0.002, -0.003, -0.004, -0.005, -0.006, -0.007, -0.008, -0.009, -0.01

Der mindes om, at alle efterfølgende resultater vi ser på, er baseret på *validation* perioderne.

Lad os som det allerførste se hvordan de 8 forskellige modeller klarer sig, statistisk, ved at se på deres *MAE* for hvert af de 10 vinduer, samt et samlet *MAE* over de 10 vinduer (se tabellen på næste side).

Siden tabellen indeholder forholdsvis mange elementer, er *MAE* ganget op med 1000 for at skabe os et bedre overblik.

Det første vi bemærker er at der er en forholdsvis klar vinder – nemlig *ANN meta-learner*. Den har den laveste *MAE* i 6 af de 10 vinduer, og også det laveste samlede *MAE*.

Blandt *base-learners* er *RF* den bedste, ud fra et samlet *MAE*. Den bliver kun slået af *ANN meta-learner* samt *EN meta-learner*, som er vores to bedst præsterende modeller igennem vores *walk-forward-optimization*.

På den anden side, så er *RF meta-learner* klart den dårligste blandt vores modeller, samlet set, og bliver som den eneste slået af *zero-* og *mean-prediction*.

ANN base-learner klarer sig samlet set middelmådigt ifht. de andre modeller. Hvis vi ser nærmere på de enkelte vinduer, så ses det, at den har forholdsvis ustabile resultater. Den har

Modeltype	1	2	3	4	5	6	7	8	9	10	Alle vinduer
EN base-learner	4.05656	6.46855	8.32604	5.14487	6.99532	7.10447	4.22915	4.26916	3.81903	2.80874	5.32228
RF base-learner	4.04509	6.40223	8.23423	5.12061	6.89556	6.99183	4.14828	4.22830	3.80046	2.81002	5.26775
GBM base-learner	4.04444	6.41400	8.24039	5.13361	6.92216	7.01895	4.14957	4.22812	3.79830	2.80047	5.27508
ANN base-learner	4.06629	6.40154	8.43020	5.12058	7.01749	6.99427	4.14150	4.18767	3.81345	2.79523	5.29691
EN meta-learner	4.04326	6.40786	8.23365	5.11960	6.89394	6.99525	4.14534	4.23424	3.79985	2.80173	5.26756
RF meta-learner	4.08275	6.43964	8.26177	5.15804	6.94496	7.04396	4.16730	4.45083	3.91579	2.90224	5.33681
GBM meta-learner	4.05235	6.41914	8.23147	5.12512	6.89345	6.97789	4.15120	4.23831	3.80596	2.80084	5.26966
ANN meta-learner	4.04103	6.38153	8.22192	5.12069	6.90195	6.96637	4.14433	4.23341	3.79173	2.78369	5.25875
Zero-prediction	4.07234	6.49523	8.34793	5.17060	7.00624	7.11410	4.22714	4.28373	3.83287	2.79693	5.33480
Mean-prediction	4.06738	6.49000	8.34850	5.17012	7.01087	7.11837	4.22266	4.28523	3.83501	2.79682	5.33458

den bedste *MAE* i vindue 7 og 8, mens den i vindue 3 og 5 bliver slået af alle vores andre modeller, samt *zero-* og *mean-prediction*. Det kunne tyde på, at vores valgte hyperparameterintervaller for denne *base-learner* ikke blev indskrænket tilstrækkeligt fra vores side, eller at den skal tildeles længere tid i sin *random-grid-search* for at opnå mere stabile resultater.

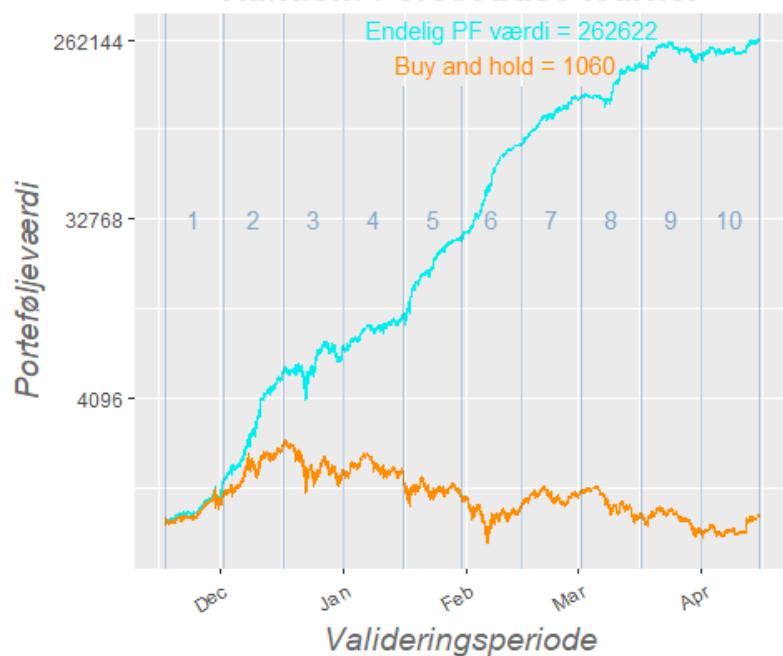
Ud fra tabellen, er der en *base-learner* og tre *meta-learners* der skal placeres i top 4, ud fra et samlet *MAE*. Det lader altså til, at en *stacking* tilgang kan være gunstig – ihvertfald i vores setup.

Lad os nu se hvor brugbare vores modeller er til deres egentlige formål – nemlig som det fundamentale værktøj i hver sin *trading* strategi.

Indledelsesvis vælger vi kun at se på udviklingen af porteføljeværdien over alle vinduer for den bedste *base-learner* og den bedste *meta-learner* i tabellen – hhv. *RF base-learner* og *ANN meta-learner*. Figurer for de andre strategier findes i bilag.

I figurerne på næste side er Y-aksen skaleret med den naturlige logaritme.

Random Forest base-learner

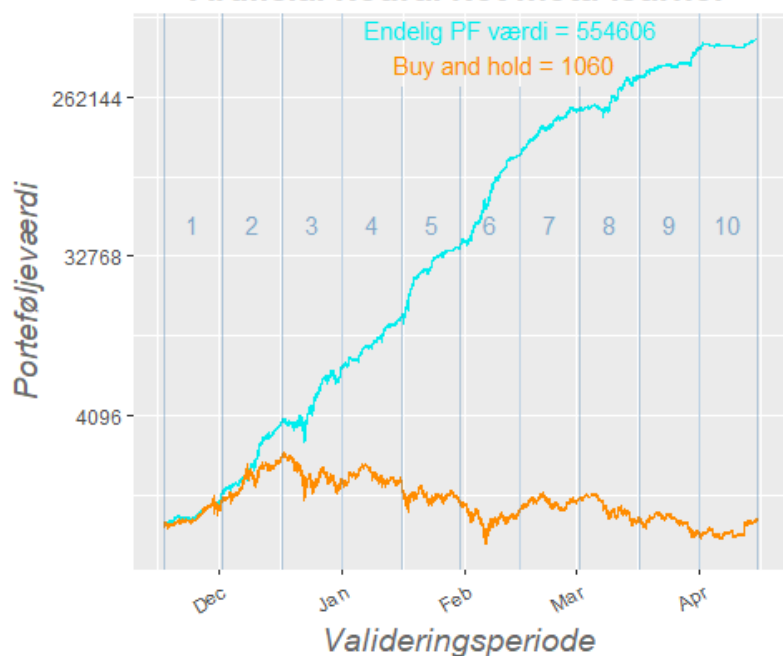


Porteføljeadviklingen for *RF base-learner* strategien ser umiddelbart meget fornuftig ud.

Begyndende med et beløb på 1000 USD, formår strategien, vha. bl.a. prædiktioner fra denne *base-learner*, at forrente initialbeløbet næsten 262 gange, da porteføljen den 15.04.18 har en værdi på 262.622 USD.

Havde vi istedet brugt en køb-og-hold strategi over disse 150 dage, ville den endelige porteføljeværdi være 1060 USD – et afkast på 6%.

Artificial Neural Net meta-learner



For strategien der indeholder den statistisk bedste af modellerne fra *walk-forward-optimization*, nemlig *ANN meta-learner*, er den endelige porteføljeværdi på 554.606 USD mere end dobbelt så stor som strategien med *RF base-learner*, og slår dermed også køb-og-hold i endnu højere grad.

Ud fra tabellen nederst på siden, ses det at de to *trading* strategier har givet afkast i alle 10 vinduer.

Procentmæssigt afkast for disse to strategier for hvert af de 10 vinduer

Modeltype	1	2	3	4	5	6	7	8	9	10
RF base-learner	60.51	256.33	29.37	41.19	154.37	192.64	74.00	43.85	12.07	20.39
ANN meta-learner	50.99	154.36	99.53	91.93	158.16	222.85	81.28	51.41	47.58	11.69

Det skal siges, at perioden som *walk-forward-optimization* løber over, ikke er den bedste for en aktør, der vælger at udføre en køb-og-hold strategi på. Javist, den faktiske pris løber lidt løbsk i starten, og topper lige under 20.000 USD den 17. december, men de efterfølgende måneder tærer hårdt på prisen – i form af flere pludselige negative chok – der når bunden på omkring 6.000 USD den 6. februar og slutter derefter omkring de 8.000 USD ved det endelige tidspunkt. Det er dog interessant at se, hvor ”resistente” de to *trading* strategier er overfor de fleste af disse choks.

Hvis vi ser på tiden omkring det første, store negative chok efter den globale top, dvs. i starten af valideringsperiode 3 (indikeret med den vertikale linje før 3-tallet i figuren), ses det, at porteføljeværdierne for vores to *trading* strategier er modtagelige overfor det omtalte chok, men ikke ligeså meget som for køb-og-hold strategien.

Vi husker på hvordan data er opdelt i hvert vindue. Vi ved, at det data der er brugt til optimering af købs og salgssignaler, samt at finde de optimale hyperparametre for modellerne, nemlig *test*, er de 20 dage før starten af periode 3. Disse 20 dage er en meget opadgående periode for *Close* prisen, og det kan derfor tænkes, at de fundne købs- og salgssignaler, samt hyperparametre, ikke er optimale til denne kraftigt nedadgående periode.

RF base-learner – trading signaler

Vindue	købs-signal %	salgs-signal %
1	0	-0.1
2	0.1	-0.2
3	0.1	-0.2
4	0.1	-0.1
5	0.1	0
6	0.2	0
7	0.1	-0.1
8	0.1	-0.1
9	0	-0.1
10	0	-0.1

I slutningen af denne, for det meste, nedadgående periode 3 ses et andet chok, der også har en negativ effekt på begge *trading* strategiers porteføljer, men ikke i lige høj grad som for det første chok.

Kaster vi derimod blikket på periode 4, 5, og 6, så indeholder disse, ligesom periode 3, nogle kraftige negative chok, men denne gang ses de to *trading* strategier ikke at have samme modtagelighed overfor disse, som de havde i periode 3. Faktisk ser det ligefrem ud til at de omkring de store negative chok har forholdsvis høje afkaststrater, som tabellen på forrige side også viser. Dette er nok mest tydeligt i starten af periode 5, samt lidt inde i 6, hvilket også er perioder med nedgang.

ANN meta-learner – trading signaler

Vindue	købs-signal %	salgs-signal %
1	0	-0.1
2	0.2	-0.1
3	0.1	-0.1
4	0.2	-0.1
5	0.2	0
6	0.2	-0.1
7	0.1	-0.1
8	0.1	-0.1
9	0.1	0
10	0.1	0

En mulig forklaring på dette, er at *test* for hvert af disse tre vinduer, også vil indeholde negative perioder, og at vores hyperparametre, samt købs- og salgssignaler, vælges således, at de tager dette i betragtning og strategien derfor er ”tilpasset” denne type perioder. Hvis vi ser på den øverste af de to tabeller på denne side, så ses det, at salgssignalet var -0.2% for *RF base-learner* i periode 2 og 3. Der skulle altså en forholdsvis kraftig negativ prædiktion til for at aktivere dens salgssignal, muligvis fordi *test* perioden i disse to vinduer har bestået

af en opadgående trend. Kigges der derimod på vindue 4, 5 og 6 ses det, at salgssignalet er hhv. -0.1% , 0% og 0% , muligvis fordi *test* perioden i disse tre vinduer har bestået af en nedadgående trend, som alt andet lige, gør at strategien ikke kræver så kraftige negative prædiktioner for at aktivere dens salgssignal.

Som sagt, så har *test* for disse perioder været omfattet af de omtalte negative trends og choks. Ser vi udelukkende på periode 6, så ses strategiens købssignal at være på sit højeste og der skal en forholdsvis kraftig positiv prædiktion til før dens købssignal aktiveres.

For *ANN meta-learner* strategien kan man ane nogle af de samme tendenser, men dette er ikke lige overbevisende.

Overordnet set, lader det til, at hvis *test* perioden generelt har været opadgående så er signalerne lavere, og vice versa for en nedadgående periode.

I denne sammenhæng kan det også tænkes, at kortere valideringsperioder kan give en bedre tilpasningseffekt, da observationerne i *validation*, i gennemsnit, vil ligge tættere på observationerne i det tilsvarende *test* sæt.

Efter at have lavet denne inspektion på det samlede resultat for *RF base-learner*, samt *ANN meta-learner*, lad os se hvordan de forholder sig til de 6 andre modeller samlet set, men også for hvert vindue. Endvidere indrages *MAE* fra forrige tabel igen.

Modeltype	MAE	Endelig PF værdi	<i>ANN meta-learner</i> var altså klart den bedste, afkastmæssigt, af vores 8 modeller, hvilket dens <i>MAE</i> også indikerede. Ud fra <i>MAE</i> er det måske heller ikke så overraskende at <i>EN base-learner</i> strategiens endelige porteføljeværdi er lav. Derimod, og måske noget overraskende, så giver modellen med den højeste <i>MAE</i> det næsthøjeste samlede afkast, nemlig <i>RF meta-learner</i> .
EN base-learner	5.3223	2612	
RF base-learner	5.2678	262622	
GBM base-learner	5.2751	174709	
ANN base-learner	5.2969	64633	
EN meta-learner	5.2676	266504	
RF meta-learner	5.3368	335320	
GBM meta-learner	5.2697	115611	
ANN meta-learner	5.2588	554606	

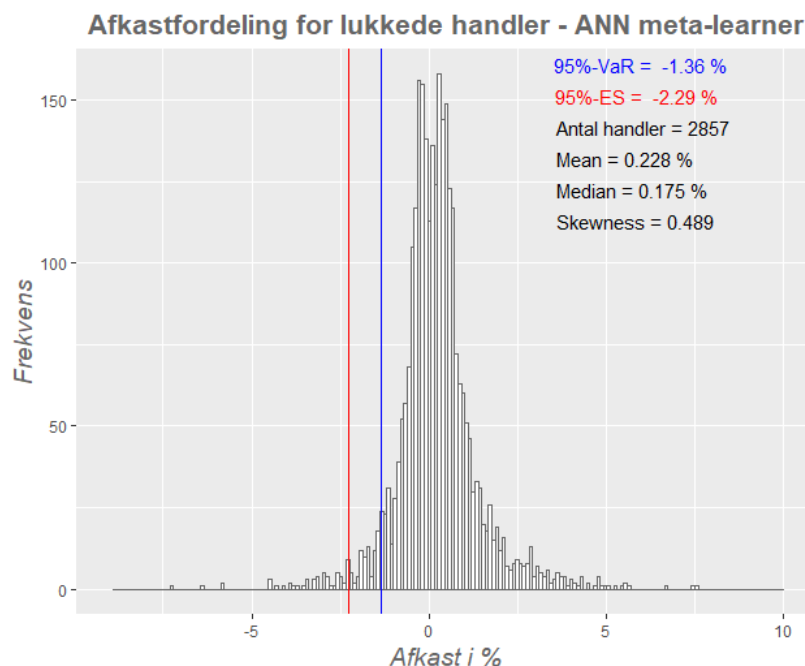
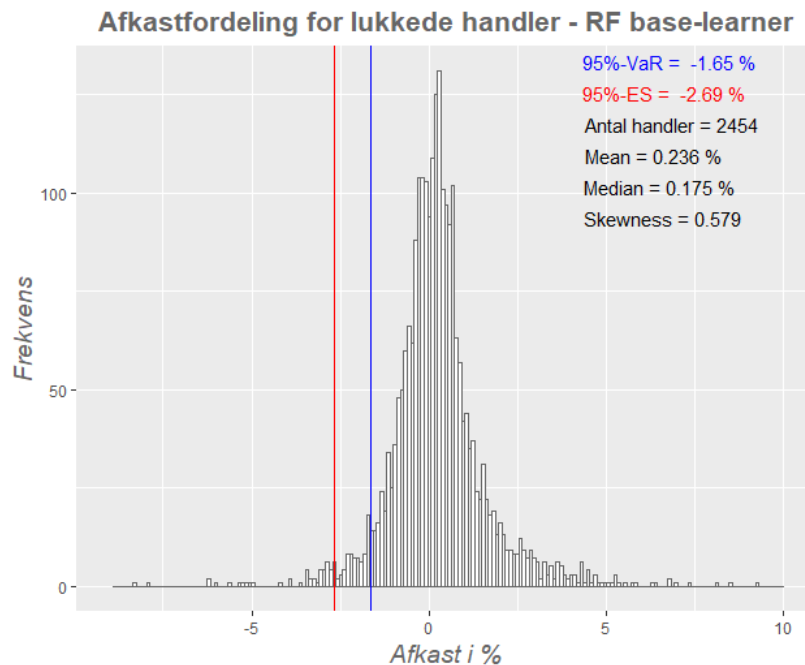
Det kunne altså tyde på, at *MAE* ikke er det bedste mål for, hvor godt en model klarer sig afkastmæssigt i vores setup.

Vi husker på at et af formålene med denne afhandling er at se hvorvidt en *stacking* tilgang i vores setup, kan forbedre de endelige resultater, dvs. klare sig bedre statistisk og afkastmæssigt, end den bedste af vores fire *base-learners*.

Ud fra de resultater vi har opnået, kunne det tyde på at dette er tilfældet.

Vi laver en allersidste evaluering af vores modeller, før vi endegyldigt ser os tilfreds med vores resultater. Udover at være interesseret i afkast, er man i forbindelse med handel nemlig ofte interesseret i et andet mål der kan være relevant – risiko. Dette gælder også i vores tilfælde. Til dette formål, har vi valgt at bruge risikomålene, *Value-at-Risk* (*VaR*) og *Expected-Shortfall* (*ES*).

For at gøre denne gennemgang en smule mere overskuelig, illustrerer vi kun disse to risikomål for den bedste *base-learner* og den bedste *meta-learner*. Risikomålene for de andre modeller er givet i bilag.



Vi ser altså på afkastfordelingerne af lukkede handler efter transaktionsomkostninger for de to modeller, hvor *VaR* og *ES* baseres på et 95%-*alpha* niveau.

Det ses, at den bedste *meta-learner* også har bedre risikomål end den bedste *base-learner*. Med et 95%-*VaR* på -1.36% kommer *meta-learner* strategien med 95% sandsynlighed ikke til at lide tab større end 1.36% af porteføljeværdien for en handel, hvorimod dette mulige tab for *base-learner* strategien er en smule højere – nemlig 1.65%.

Idet *ES* er et mere yderliggående risikomål, er dette højere for begge strategier. Givet *meta-learner* strategien lider et større tab end 95%-*VaR*, vil det forventede tab være 2.29%, hvorimod dette for *base-learner* strategien vil

medføre et forventet tab på 2.69%.

Hvis vi flytter os fra de to risikomål, og kigger direkte på deres afkastfordeling, så er der en række andre interessante ting at observere.

Medianerne for begge fordelinger er positive, så strategien med *ANN* udfører profitable handler i mere end halvdelen af tilfældene, mens vi kan konkludere at strategien med *RF* gør det samme.

Ud fra deres *skewness*, kan vi konkludere, at begge fordelinger er højreskæve, da denne værdi er positiv.

Ud fra medianen og *skewness*, kan vi for begge fordelinger konkludere, at mean også er positiv, samt at denne er større end medianen.

Vi kigger nu nærmere på hvordan fordelingerne for disse to strategier fraviger fra hinanden. Ser vi på *mean* for strategien med *Random Forest base-learner*, så har denne i gennemsnit et større afkast for lukkede handler. Vi ser dog, at strategien med *ANN meta-learner* udfører samlet set flere handler.

Vha. grafisk inspektion, ser vi at *Random Forest* strategien har flere handler med tab større end 5%, men at den også har flere handler med større end 5% afkast og dette er altså på trods af at *ANN*-strategien samlet set har flere handler.

For at konkludere på dette, så har *ANN meta-learner* strategien en lavere risiko, ud fra de to risikomål, *VaR* og *ES*.

Opsummering

Som vi kom ind på tidligere, så slår både *ANN meta-learner* og *EN meta-learner* den bedste af vores *base-learners*, nemlig *RF-base-learner*. Så ud fra det statistiske perspektiv, så tyder det på, at en *stacking* tilgang i vores setup faktisk er i stand til at forbedre prædiktionskraften.

Ser vi i stedet på den afkastmæssige præstation, så er hele 3 af vores *meta-learners* bedre end den bedste *base-learner*. De tre der slår *RF-base-learner*, er *EN*, *RF* og *ANN meta-learner*, hvor det dog skal siges, at det er et meget tæt løb mellem nr. 3 og 4.

Endvidere have *ANN meta-learner* et bedre risikomål end den bedste *base-learner* – afkastmæssigt, samt statistisk.

Med dette afkastmæssige perspektiv, lader *stacking* tilgangen, i dette setup, altså også til at forbedre resultaterne.

7.3 Detaljer og overvejelser

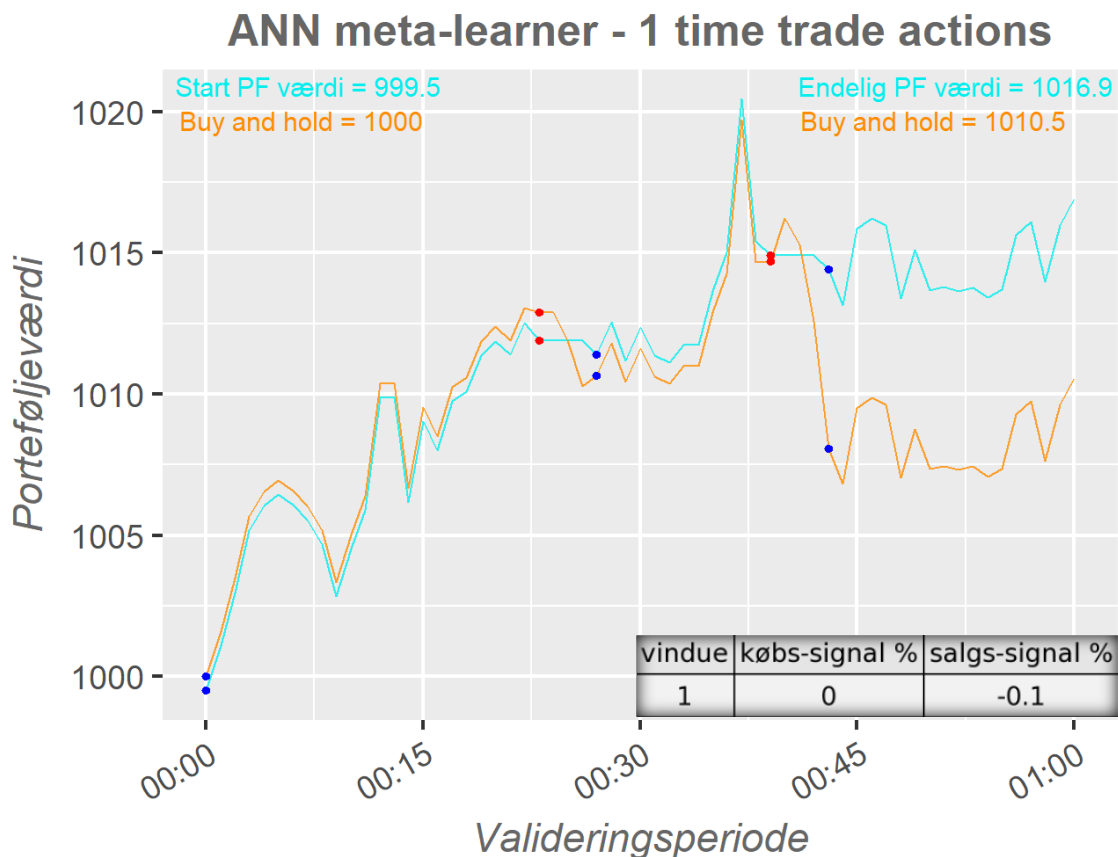
I dette underkapitel ser vi lidt nærmere på hvordan en af vores *trading* strategier handler, hvorefter vi drøfter vores *trading* antagelser.

Endeligt vises lidt praktisk information omkring vores kørsel af denne *walk-forward-optimization*.

Trading Close-Ups

I dette afsnit ser vi lidt nærmere på hvordan vores bedst præsterende strategi, *ANN meta-learner*, handler – helt tæt på.

Vi lægger ud med at se på den første time af valideringsperioden i det første vindue.



Den blå linje repræsenterer porteføljeværdien for *ANN meta-learner* strategien og den orange linje er køb-og-hold strategiens porteføljeværdi.

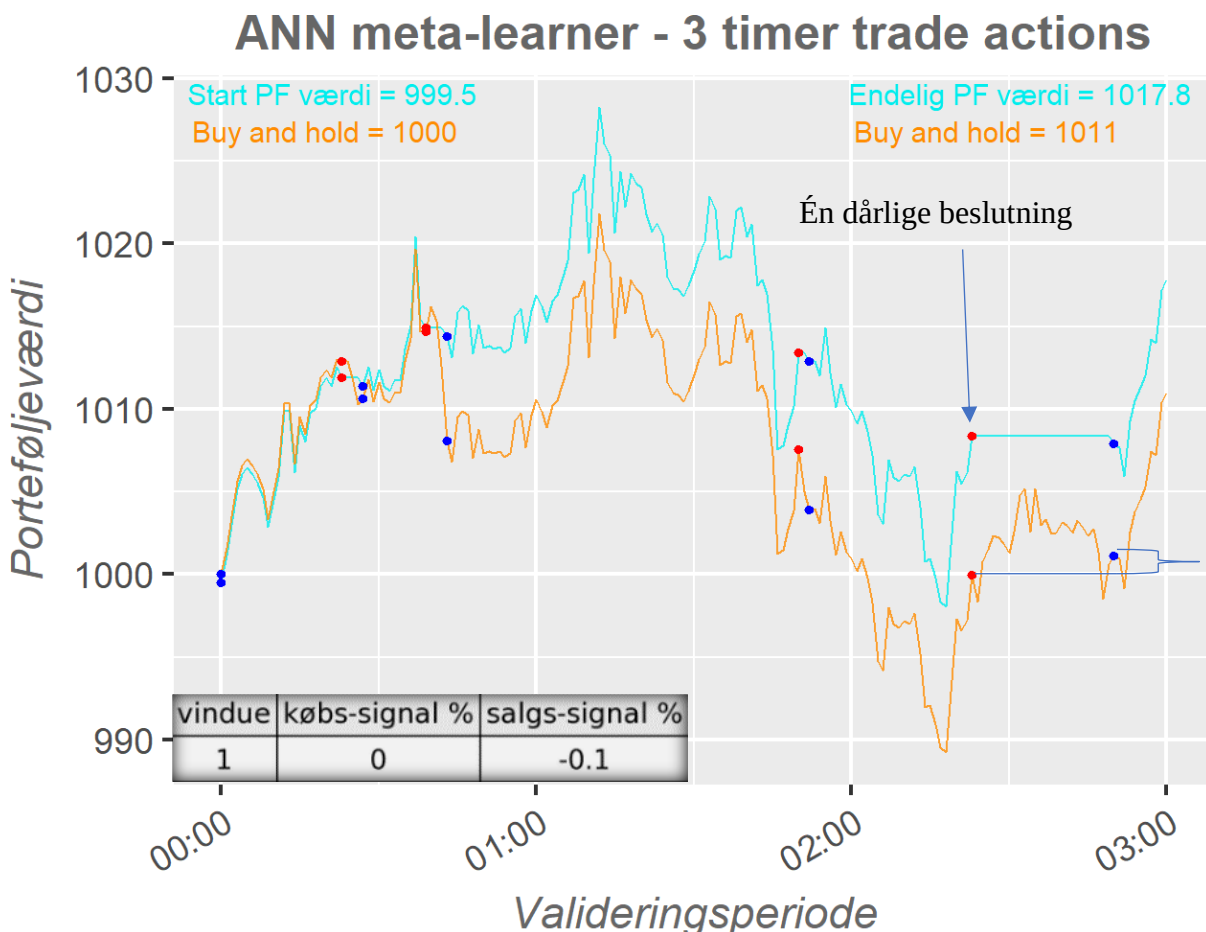
Strategien køber på de blå prikker og sælger på de røde. Bemærk at disse prikker er indikeret på begge linjer for nemmere at kunne sammenligne, men de påvirker selvfølgelig kun *ANN meta-learners* porteføljeværdi – nemlig den blå linje.

Det ses, at ANN-strategien, straks efter at have observeret den første *Close* pris, køber *BTC* for hele porteføljen, der inden købet var 1000 *USD*. Det resulterer i, at den bruger transaktionsomkostninger på 0.05% på købet, og porteføljeværdien bliver øjeblikkeligt 0.5 *USD* lavere, og får en værdi på 999.5 *USD*. Strategien holder *BTC* indtil *Close* prisen for 00:23 er observeret, hvor den får et salgssignal og sælger øjeblikket til denne *Close* pris. Porteføljeværdien er nu 1011.9 *USD*, hvorimod køb-og-hold strategiens porteføljeværdi er 1012.91 *USD*, da denne ikke har havt to transaktionsomkostninger på 0.05%.

Med de tilhørende optimale købs- og salgssignaler på hhv. 0% og -0.1% fortæller disse to handler, at vores ANN *meta-learner* prædikerer en stigning i *Close* prisen fra 00:00 til 00:15 og et fald på over 0.1% fra 00:23 til 00:38.

Det ses at strategien kunne have handlet på bedre tidspunkter, men den har dog overskud på alle lukkede handler, og genkøber altid billigere end forrige salgspris (husk at den orange linje jo er en skalering af *Close* prisen). Det ses samtidigt, at porteføljeværdien er steget med 1,69% efter den første time, hvorimod porteføljeværdien for køb-og-hold strategien er steget med 1,05% (selvom vi ikke har givet denne strategi nogen start transaktionsomkostning).

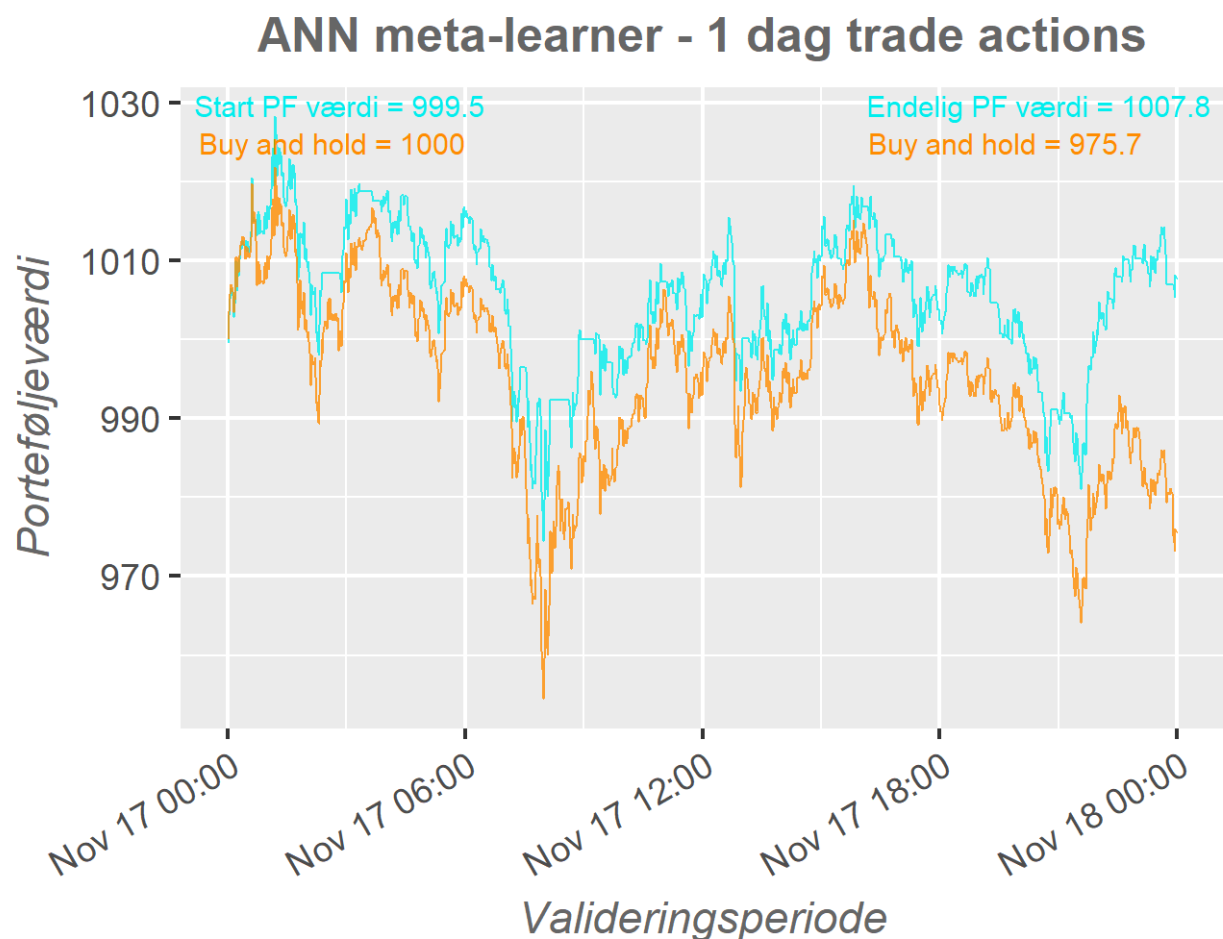
Vi øger nu tidshorisonten til at inkludere de næste 2 timer.



Sammenlignet med køb-og-hold, så træffer strategien nu en dårlig beslutning. Det vedrører nemlig de sidste to *trades*, hvor den sælger, men køber igen lidt dyrere. Dette er samtidig den eneste gang over de første tre timer, hvor den har holdt *USD* i over 10 minutter. Det sidste køb ser ellers ud til at være fornuftigt. Samtidig har porteføljeværdien for vores strategi været over køb-og-hold siden det tredje køb.

Det er en smule håbløst at fortsætte med at illustrere hver enkelt *trade* på denne måde, da disse hurtigt ville overdække hinanden, samt porteføljeudviklingen. Som det sidste i denne gennemgang illustreres dog udviklingen igennem den første dag.

Man kan ane hvordan strategien handler, ved at se på de korte, horisontale linjer, hvor dens portefølje er i *USD*. Næste dag starter altså med et overskud på 0,78%, hvorimod køb-og-hold begynder dagen med et underskud på 2,43%.



Der mindes om, at den samlede udvikling for *ANN meta-learner* strategien er at finde på en af de første sider i underkapitel 7.2.

Overvejelser omkring trading antagelser

Som vi har set, lader det til at vi er stand til at producere nogle strategier der slår køb-og-hold afkastmæssigt. Men lad os se lidt mere nøje på vores *trading* antagelser.

Vi benytter her den bedst præsterende strategi, nemlig *ANN meta-learner* strategien, i denne uddybning.

Vi husker først tilbage på vores *trading* antagelser.

- Strategien handler altid til *Close* prisen, inden næste *Close* pris observeres .
- Strategiens ordrer fyldes altid.

For at se hvorvidt dette er realistiske antagelser, er det oplagt at se på, hvor meget strategien har handlet for og hvordan antallet af ordrer forløber over de 10 vinduer.

ANN meta-learner	Antal transaktioner	Gns. transaktionsvolume	Samlet transaktionsvolume
1	741	1191	882533
2	353	2325	820597
3	575	5082	2922058
4	459	10498	4818396
5	702	25547	17933710
6	692	70337	48673405
7	630	180722	113854698
8	496	271830	134827647
9	908	396892	360377619
10	162	516554	83681776

Vi husker, at startværdien for strategiens portefølje er *1.000 USD*.

Igennem det første vindue eksekverer den 741 transaktioner (køb og salg) der medfører en samlet transaktionsvolume på *882.533 USD*. Det betyder, at dens gennemsnitlige transaktionsvolume er *1.191 USD*.

Givet den høje volume på *Bitfinex*, er det forholdsvis rimeligt at antage, at de fleste af strategiens ordrer med denne gennemsnitlige volume kunne blive fyldte, givet dens ordrer placeres i ordrebogen snarligt efter at *Close* prisen for minuttet er observeret.

Begynder vi derimod at bevæge os videre til de andre vinduer ses en kraftig stigning i den gennemsnitlige transaktionsvolume, som nogenlunde afspejler den stigende porteføljeværdi (husk, at strategien altid handler for hele porteføljeværdien).

I vindue 10 ses den gennemsnitlige transaktionsvolume at være på *516.554 USD* med en tilsvarende, samlet volume på *83.681.776 USD*.

Det er især den gennemsnitlige transaktionsvolume der vækker øjenbryn.

Det er nemlig meget svært at forestille sig, at samtlige af strategiens ordrer bliver fyldte, når de har en så stor volume.

Hvis vi kigger på et tilfældigt valgt udklip af en del af ordrebogen for *BTC/USD* på *Bitfinex*, taget den *05.05.2018 13:10*, bliver dette måske mere tydeligt. Den aflange figur er 1-minut volumen op til *13:10* (denne minut er lige begyndt, derfor kun en volume på *0.11 BTC*).

9547.60	0.4000000	31.4017975
9547.30	0.8681760	31.0017975
9545.80	0.5931175	30.1336215
9545.60	1.0560000	29.5405040
9545.20	0.0065082	28.4845040
9545.10	1.7711062	28.4779958
9543.90	0.4500000	26.7068896
9543.70	1.0000000	26.2568896
9543.30	1.4350100	25.2568896
9542.70	0.3497133	23.8218796
9540.00	14.0424633	23.4721663
9539.00	8.8762798	9.4297030
9538.60	0.5534232	0.5534232
9538.50 USD		
9538.50	2.8496909	2.8496909
9538.40	1.5508000	4.4004909
9538.30	1.2501512	5.6506421
9538.00	12.0200005	17.6706426
9537.90	0.0500000	17.7206426
9537.00	0.3000000	18.0206426
9536.90	0.5556462	18.5762888
9536.30	3.0000000	21.5762888
9536.00	0.5575469	22.1338357
9535.80	0.8000000	22.9338357
9535.50	1.0000000	23.9338357
9534.20	0.0500000	23.9838357
9533.80	1.0000000	24.9838357

Med den nuværende markedspris ville den gennemsnitlige ordre for vindue 10 svare til ca. $\frac{516554 \text{ USD}}{9538.50 \text{ USD/BTC}} = 54.15 \text{ BTC}$.

Som vist i figuren for ordrebogen, ville beløbet af en ordre af denne størrelse nogenlunde svare til beløbet af det samlede antal ordrer over et prisinterval på *9533.8* til *9547.6 USD/BTC*. Samtidig ser vi, at 1-minut volumen sjældent overskrider *50 BTC* for denne periode.



Med disse to observeringer taget i betragtning, lader det til at være meget usandsynligt, at alle af strategiens ordrer vil være fyldte inden næste *Close* pris. Det kunne samtidig tænkes, at jo større ordrene er, jo mere usandsynligt vil det være, at de fyldes til den *ønskede* pris (*Close* prisen).

Det er også muligt, at det ville være nemmere at opfylde en købsordre hvis prisen har en nedadgående trend, og vice versa for en salgsordre.

I den forstand kan det tænkes, at der i realiteten ville være større chance for at strategiens købsordrer bliver fyldte på "uheldige" tidspunkter, samt at dens salgsordrer *ikke* bliver fyldt på "afgørende" tidspunkter. Samlet set kan man forestille sig at dette vil have en negativ effekt på porteføljeværdien over tid.

Endeligt, da vi antager, at strategien altid handler til *Close* prisen, betyder dette i princippet, at det er usikkert hvorvidt strategiens ordre skal omtales som en *limit order* eller *market order*, og at der dermed kan være forskel på omkostningerne i form af hhv. *maker fee* og *taker fee*.

Hvis det sidste *ordermatch* i et minut (dvs. det *match* der afgør minuttens *Close* pris) er en købsordre der "æder" af den laveste salgsordre, kan der være resterende volumen tilbage i salgssordren. Hvis vores strategi f.eks. øjeblikkeligt køber til denne *Close* pris, så vil den også "æde" af denne salgsordre, og betale *taker fee* for denne transaktion. Som sagt i kapitel 2, så er denne type transaktionsomkostning ofte højere.

Til vores *walk-forward-optimization*, har vi konstrueret en funktion, *Everything*, der eksekverer hele vores samlede handlingsstrategi, sådan som vi har beskrevet den i dette kapitel. Den t mest relevante af dens input,

Lidt praktisk information omkring tid og computerkraft

I forbindelse med denne afhandling har vi bygget en funktion i *R*, der er i stand til at eksekvere en hel kørsel af *walk-forward-optimization* med "ét click", om man vil.

Den har været under opbyggelse i nogle måneder og indeholder modeller – udover de 8 modeller vi har redegjort for i dette kapitel – der også går igennem *walk-forward-optimization*. Disse 12 andre modeller er dog meget billige i tidsomkostninger da *grid-search* ikke bliver direkte anvendt på disse (de har *H2O* standardværdier for hyperparametrene). Som konsekvens er det generelt dårlige modeller der er tale om.

Se bilag for yderligere beskrivelse af denne funktion, som vi måske ikke så mærkeligt har kaldet *Everything*. Er man nysgerrig, kan man gøre sig mere klog på denne, og evt. køre den, eller indeksere datafilerne, der indeholder samtlige af vores resultater fra hvert vindue.

I dette afsnit ser vi lidt nærmere på nogle detaljer omkring tid og computerkraft fra vores kørsel af *walk-forward-optimization* vha. *Everything*.

Nedenfor ses et billede af *CPU utilization* fra *Amazon Web Service* for vores kørsel af de 10 vinduer.

Det tog 15 timer og 35 minutter at nå frem til de resultater vi har fremvist. Her ses det, at der efter hvert vindue ventes efter initialisering af *H2O* (30 sekunder), samt at vi "kun" allokerer 20 (20 modeller) ud af i alt 36 kerner på vores *Amazon c5.9xlarge*-server, til paralleliseringsfunktionen *foreach()*, der blandt andet bruges til at udregne optimale købs- og salgssignaler, samt porteføljeværdien af valideringsperioden for hvert vindue. De sidste 16 kerner bliver brugt til at tage imod det data, som vores funktion spytter ud. Hver gang *CPU utilization* kommer ned imod 60% (indikeret med den horisontale linje på billedet) er et vindue udregnet.

Det ses samtidigt at der er 2 spidser i *CPU utilization* for hvert vindue. Den første spids opstår når der udføres *grid-search* for vores *base-learners*, hvor udregningerne altså bliver forholdsvis intensive, og den anden opstår når vi laver *grid-search* for vores *meta-learners*, men dette *grid-search* varer jo ikke lige lang tid (husk *max_runtime_secs*).

De første 2 pile angiver de to spidser for de første vindue, mens de to næste pile angiver spidserne for det sidste vindue.

CloudWatch Monitoring Details

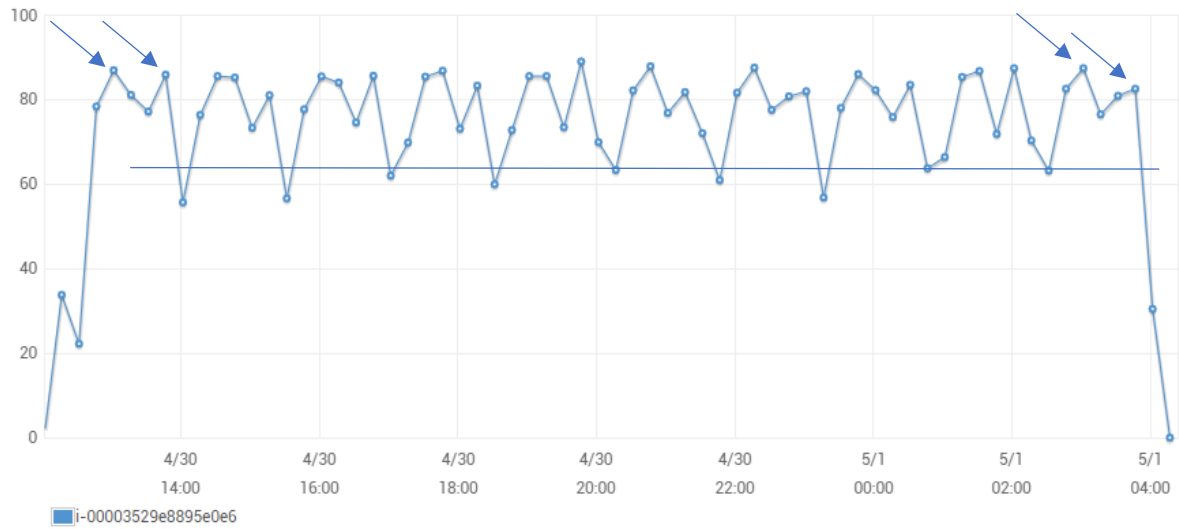


CPU Utilization (Percent)

Statistic: **Average**

Time Range: **Last 24 Hours**

Period: **15 Minutes**



i-00003529e8895e0e6

Close

8 Diskussion og perspektivering

I dette kapitel vil de resultater der er opnået i denne afhandling blive diskuteret. Endvidere bliver der perspektiveret til nogle af vores resultater, samt fremgangsmåder.

Prædiktioner af afkast

Vi valgte at gøre brug af *walk-forward-optimization*, der anvendes til optimering af *trading* strategier på de finansielle markeder. Denne tilgang gav vores statistiske modeller mulighed for at tilpasse sig over tid – både i form af nyt data, samt mulige ændringer i optimale hyperparametre. Da disse var et værktøj i hver sin adaptive *trading* strategi, omfattede denne metode også vores afkastmæssige resultater.

Vi argumenterede for, at markedstendenser kan ændre sig over tid og det derfor kan være fordelagtigt at opsplitte den samlede valideringsperiode i flere vinduer, og optimere modeller samt de tilhørende *trading* signaler, i hvert vindue.

Dette passede også godt ind med, at vi gerne ville benytte et forholdsvis stort *train* for vores modeller, samt et stort *validation*, uden at benytte muligvis forældet data i vores modeller. Ved at lade vores process glide igennem vinduerne i *walk-forward-optimization* blev vores ønske opfyldt.

Vi fandt, at udledningen af hyperparameterintervallerne generelt lod det til at være passende valgte for alle vores *base-learners*, samt de fleste af vores *meta-learners*. *Random Forest meta-learner* var nemlig den eneste af de 8 som ikke havde et bedre prædiktionsresultat end de to *benchmark-modeller*, *mean-* og *zero-prediction*, over de 10 vinduer. Det er muligt, at udledningen af denne models hyperparameterintervaller ikke blev udført grundigt nok fra vores side, da *Random Forest* typisk anses for at have en høj prædiktionskraft.

Da 3 af vores modeltyper havde et forholdsvis komplekst hyperparameterum, valgte vi at benytte *random-grid search* for at finde frem til passende hyperparameterintervaller for disse. Dette valg blev primært baseret på tidsomkostninger.

Vi så at *ANN base-learner* havde ustabile prædiktionsresultater, da den i to af vores vinduer havde det mindste *MAE*, mens den i to andre vinduer havde det højeste *MAE*, hvor selv vores 2 *benchmark* modeller præsterede bedre.

I denne sammenhæng kunne det være interessant at se, i hvor høj grad vores statistiske resultater havde ændret sig, hvis vi istedet for at lade hyperparametrene for vores statistiske modeller variere igennem de 10 vinduer, fandt optimale *værdier* for disse, der altså ikke skulle variere igennem vores *walk-forward-optimization*.

Igennem *walk-forward-optimization* viste de samlede prædiktionsresultater, at en *stacking* tilgang var fordelagtig i vores setup.

3 af de 4 bedst præsterende modeller var *meta-learners*, hvoraf de 2 bedste var *meta-learners*. *Elastic Net base-learner* var den dårligst præsterende af de 4 *base-learners*. Årsagen til dette kan muligvis findes i, at *Elastic Net* i vores setup, ikke tager højde for ulineære sammenhænge mellem respons og *features*, samt vekselvirkninger. Vi tog ikke højde for dette i vores *Elastic Net base-learner*, grundet vores forholdsvis høje antal *features*.

Da de 3 andre *base-learners*, per automatik, tager højde for dette, og disse klarede sig bedre end *Elastic Net base-learner*, er det rimeligt at antage, at det netop er nødvendigt at tage højde for ulineære sammenhænge mellem respons og *features*, og/eller vekselvirkninger, for at udføre tilfredsstillende prædiktioner af vores respons.

På den anden side, så fandt vi, at *EN meta-learner* præsterede bedre end alle vores *base-learners*. I den forstand tyder det altså på, at en lineær kombination af *prædiktioner* fra vores *base-learners* kan være gunstig for prædiktionskraften.

Samlet set var den bedst præsterende af vores 8 modeller, med hver sine hyperparameterintervaller, en *meta-learner*. I hele 6 ud af de 10 vinduer havde *ANN meta-learner* nemlig den laveste *MAE*, hvilket også medførte den laveste samlede *MAE*.

Det skal dog siges, at i vores setup, har tiden der bruges på *random-grid-search* for modeltyperne *RF*, *GBM* og *ANN*, en betydelig indflydelse på deres prædiktionskraft. Man kan derfor muligvis forestille sig scenarier, hvor vores *base-learners* ville have en lavere *MAE* end vores *meta-learners*.

Da vi som sagt fandt, at flere af vores *meta-learners* præsterede bedre end den bedste *base-learner*, ville en oplagt udvidelse af vores modellering være, at opbygge en *stacking* model med prædiktioner fra *meta-learners* som *features*.

Endvidere kan vi også forestille os, at anvendelsen af flere forskellige typer statistiske modeller – både som *base-* og *meta-learners* – kunne have forbedret vores statistiske resultater.

Vi valgte at benytte tekniske prisindikatorer af prisen for *BTC/USD*, samt volumen, som *features* i vores *base-learners*, og dermed indirekte også i vores *meta-learners*.

Da modellerne, ifht. vores *benchmarks* generelt viste sig at have en bedre prædiktionskraft på fremtidigt *Close* afkast, kunne det tyde på at tekniske prisindikatorer har en forklarende effekt på den fremtidige pris. Da disse indikatorer ikke er udgjort af andet end historiske værdier af prisen, samt volumen, kunne det derfor også tyde på, at den historiske pris for *BTC/USD* har en forklarende effekt på den fremtidige pris.

Som vi kom ind på tidligere i denne afhandling, så har skepsisen været stor vedrørende anvendelsen af teknisk analyse på de finansielle markeder.

I vores setup lader det altså til at have en forklarende effekt på *BTC/USD* markedet. I kapitel 2 kom ind på, at forklaringen kunne være de manglende muligheder for udførelsen af fundamental analyse på dette marked og at aktørerne derfor ikke har så meget andet at basere deres forventninger til prisen på.

Vi kan dog ikke sige noget om hvordan dette forholder sig over andre perioder eller andre markeder, men vi kan forestille os at det samme kan siges om andre kryptovaluta-mod-fiat-

markeder (f.eks. *Ethereum* imod dollaren).

I kapitel 2 kom vi også ind på, at kryptovalutamarkedet kunne være mere udsat for manipulation end de traditionelle markeder, da reguleringen ikke er på samme niveau. Hvis dette er tilfældet, så er det ikke helt til at vide i hvilken grad eller retning det påvirker vores modellers prædiktionskraft.

Der er et utal af forskellige måder vi kunne have udvidet vores *feature*-rum.

Vi kunne f.eks. have udvidet det til også at indeholde *social-media* data, da man kan forestille sig, at dette kunne have en forklarende effekt på prisen. F.eks. har den sociale nyheds- og diskussionsplatform, *Reddit*, et forum dedikeret til *Bitcoin*, der i skrivende stund har omtrentligt 820.000 læsere. Det ville være interessant, at se hvorvidt både antallet, samt *upvotes* (sammenligneligt med *likes* på *Facebook*) af indlæg på dette forum, sammen med vores nuværende *features*, ville øge prædiktionskraften.

I denne sammenhæng kunne man f.eks. også inddrage antallet af *Bitcoin* bemærkninger på *Twitter* den seneste minut som en *feature*.

For samtlige af de benyttede tekniske indikatorer, valgte vi at bruge nogenlunde standardværdier i deres *lags*. Vores ræsonnement var at de fleste af markedsaktørerne der vælger at handle ud fra de tekniske indikatorer også benytter disse standardværdier. Samt dette, udvidede vi nogle af indikatorerne til at bestå af korte *lags* og nogle med lange *lags*, for at fange mere kortsigtede og langsigtede tendenser. Endvidere valgte vi at gøre brug af 2 differencede serier af hver *feature* i håbet om, at udviklingen af vores *features* havde en vis forklaringsgrad.

Vi kunne selvfølgelig have udvidet dette til både at inkludere flere antal *lags*, samt flere forskellige differencede serier.

Den adaptive trading strategi

Vi valgte at konstruere en adaptiv *trading* strategi, som både skulle tage hensyn til, at de statistiske modeller der leverede prædiktionerne til strategien, samt strategiens købs- og salgssignaler, skulle kunne ændres over tid.

Dette blev som sagt udrettet igennem *walk-forward-optimization* metoden.

I vores *walk-forward-optimization* indeholdt hvert vindue 235 dage, hvoraf de 220 første dage, i form af *train* og *test*, var vinduets *in-sample* data, mens de resterende 15 dage, i form af *validation*, var *out-of-sample* data.

Vores *trading* resultater viste, at den endelige porteføljeværdi for alle 8 *trading* strategier var højere end for køb-og-hold strategien.

Den dårligst præsterende af disse var strategien der var baseret på prædiktioner fra *Elastic Net base-learner*. Da det viste sig, at dens samlede *MAE* også var forholdsvis høj, var dette

umiddelbart ikke så overraskende. Derimod, havde strategien med *Random Forest meta-learner* en højere samlet *MAE*, men opnåede en langt højere endelige porteføljeværdi. I vores setup er der altså noget der tyder på, at et lavt *MAE*-fejlmål ikke er altafgørende for en *trading* strategis præstation.

Den allerbedste af vores *trading* strategier viste sig at være *ANN meta-learner*, der samtidig også var den med det laveste fejlmål. Det viste sig generelt, at strategierne som var baseret på *meta-learners* opnåede en højere endelig porteføljeværdi end strategierne der var baseret på *base-learners*. 3 af de bedst præsterende strategier var nemlig baseret på prædiktioner fra *meta-learners*.

Ved at se på de afkastmæssige præstationer for den bedste *base-* og *meta-learner*, så vi tegn på at *walk-forward-optimization* metoden havde den ønskede effekt. Det lod til, at strategierne var i stand til at tilpasse sig de ændrede markedsforhold – dette var især synligt da vi sammenlignede vinduerne 4, 5 og 6 med vindue 3. Det var primært udviklingen i porteføljeværdien, samt hvordan de optimale *trading* signaler tilpassede sig, der indikerede dette.

Den samlede valideringsperiode indeholdt mange store udsving i prisen. Vi er tilfredse med dette valg af valideringsperiode, da vi var nysgerrige efter at se hvordan strategierne klarede sig, både for store negative, samt positive udsving i prisen.

Der er også tegn på, at vi kunne have forbedret strategierne i vores setup yderligere. Som vist i vores nærmere kig på hvordan *trading* signaler varierede fra vindue til vindue for den bedste *base-* og *meta-learner*, så vi, at de for det meste foretrak de lavere absolutte værdier af købs- og salgssignalerne. Ved at indsnævre deres intervaller fra $[0, 1]\%$ og $[-1, 0]\%$, med en skridtlængde på 0.1% , til f.eks. at være hhv. $[0, 0.3]\%$ og $[-0.3, 0]\%$, med en skridtlængde på 0.03% , er det muligt at disse nye intervaller for signaler ville føre til mere profitable strategier.

For at validere vores *trading* strategier, var det nødvendigt at opstille nogle antagelser for disse. Som vi kom ind på, så skulle de endelige porteføljeværdier tages med forbehold, da antagelserne nok ikke direkte kan oversættes til den virkelige verden.

Vi påviste bl.a. hvordan det reelt kan være sværere for vores strategier at få sine ordrer fyldte jo højere volumen de har, samt hvordan vores ellers faste transaktionsomkostninger kan variere.

Den eneste måde vi i realiteten kan svare på ovenstående, er igennem et *real-time trading* setup på en kryptobørs, som f.eks. *Bitfinex*.

For at komme nærmere på hvilken endelig strategi vi ville benytte i et *real-time trading* setup, ville vi umiddelbart vælge *ANN meta-learner* strategien, da denne både havde den bedste *afkastrate*, samt de laveste risikomål.

9 Konklusion

Indenfor rammerne af *walk-forward-optimization*, ville vi i denne afhandling undersøge hvorvidt det er muligt, at prædiktere fremtidigt afkast af kryptovalutaen *Bitcoin* ved hjælp af statistiske modeller, der er opbygget af tekniske prisindikatorer.

Vi ville endvidere undersøge, om disse prædiktioner kan benyttes i en *trading* strategi hvis præstation skal sammenholdes med en køb-og-hold strategi.

Endelig ville vi undersøge om en *stacking* tilgang af de statistiske modeller kan forbedre disse resultater.

For at kunne udføre dette, valgte vi at benytte *OHLCV* data fra kryptovalutabørsen *Bitfinex*. Ved at gøre brug af programmeringssproget *Python*, blev dette indsamlet igennem børsens *API*. Da vi benyttede en statistisk tilgang i vores *trading* strategi, kunne vi se nogle fordele ved at bruge det indsamlede data i dets originale 1-minut form.

Vi valgte at benytte en række tekniske prisindikatorer, der svarede til 41 forklarende variable. To differencede serier af hver af disse blev endvidere konstrueret, sådan at det endelige antal resulterede i 123 forklarende variable. Responsen blev sat til at være *Close* prisens afkast 15 minutter frem.

Der blev lagt en sidste hånd på vores forklarende variable, ved at tage højde for mulige stationaritetsproblemer. Samtlige af disse blev endvidere standardiseret.

4 forskellige statistiske modeltyper blev brugt som *base-learners* samt *meta-learners*.

Vi argumenterede for at finde passende hyperparameterintervaller for vores statistiske modeller, da vi foretrak at optimale værdier for hyperparametrene skulle have mulighed for at tilpasse sig igennem alle 10 vinduer i vores *walk-forward-optimization*, uden vores indblanding.

Først fandt vi passende hyperparameterintervaller for vores 4 *base-learners*. Derefter kunne *stacking* tilgangen benyttes til at konstruere vores 4 *meta-learners* og passende hyperparameterintervaller for disse blev derefter fundet.

Da 3 af vores modeltyper havde et forholdsvis komplekst hyperparameterum, valgte vi at benytte *random-grid search* for at finde frem til passende hyperparameterintervaller for disse. Dette blev udført med paralleliserings pakken *H2O*, i programmeringssproget *R*. Det vist sig også fordelagtigt at benytte kraftfulde, *cloud*-baserede servere til at eksekvere vores udregninger. Disse to værktøjer blev endvidere brugt i vores *walk-forward-optimization*.

Efterfølgende blev vores *walk-forward-optimization* indledt.

Vi lagde ud med at beskrive strukturen af den adaptive *trading* strategi, som *walk-forward-optimization* muliggjorde. Strategien indebar for hvert af de 10 vinduer:

- 1 Træning og optimering af statistiske modeller, samt prædiktion på *test* og *validation*.
- 2 Optimering af *trading* signaler, baseret på prædiktioner på *test*.

- 3 Validering af vores *trading* strategi, baseret på prædiktioner på *validation*, samt optimale *trading* signaler.

Først blev de 8 forskellige modeller valideret på uset data. Dette blev evalueret ud fra deres samlede *MAE* mål igennem de 10 vinduer. De statistiske resultater viste, at 3 af vores 4 bedst præsterende modeller var *meta-learners*. Den eneste model der ikke formåede at slå vores to *benchmark* modeller, *mean-* og *zero-prediction*, var *Random Forest meta-learner*.

Den bedste model ud fra det samlede *MAE*-mål var *Artificial Neural Network meta-learner*. Den bedste *base-learner* var *Random Forest base-learner*.

Prædiktionerne på det usete data fra samtlige modeller blev brugt i hver sin *trading* strategi. Hver af de 8 *trading* strategier indeholdt *trading* signaler som også skulle tilpasses for hvert vindue.

De afkastmæssige resultater fra de 8 forskellige *trading* strategier viste, at samtlige af de 8 *trading* strategier slog køb-og-hold strategien. 3 af vores bedst præsterende strategier viste sig at være baseret på *meta-learners*.

Den bedste var baseret på *Artificial Neural Network meta-learner*.

Den bedste *base-learner* strategi var baseret på *Random Forest base-learner*.

I vores risikoevaluering, fandt vi endvidere, at den bedste *meta-learner* igen slog den bedste *base-learner*.

Ud fra de resultater vi har opnået, kan vi altså svare således på vores problemformulering:

1. Det er, til en vis grad, muligt at prædiktere fremtidigt afkast af kryptovalutaen *Bitcoin* ved at bruge statistiske modeller med tekniske prisindikatorer som *features*. Vi konkluderer dette på baggrund af, at alle vores *base-learners* havde en lavere samlet *MAE* på uset data, end vores to *benchmark* modeller.
2. Givet vores *trading* antagelser, er det muligt at benytte prædiktioner fra en *base-learner* i en *trading* strategi, der slår en køb-og-hold strategi. Vi konkluderer dette på baggrund af, at samtlige af vores *base-learner* strategier havde en højere endelig porteføljeværdi end køb-og-hold strategien.
3. Da vi fandt, at 3 af vores 4 statistisk bedst præsterende modeller var *meta-learners*, samt at den bedst præsterende model var en *meta-learner*, konkluderer vi, at brugen af *stacking* i vores setup har forbedret de statistiske resultater. Endvidere fører en *stacking* tilgang til bedre samlede afkast, givet vores *trading* antagelser, da 3 af vores mest profitable strategier er baseret på *meta-learners*.

Bilag

Oversigt af filer i det digitale bilag

For at køre alt dette kræves programmeringssproget *R*.

Data mappen

Mappen der indeholder vores OHLCV data for BTC/USD.

Dækker 21.11.2016 00:07:00 til 15.04.2018 23:59:00

ResultsEachWindow mappen

Mappe der indeholder det data som resultaterne fra vores *walk-forward-optimization* bygger på. 10 filer – en fil for hvert vindue.

Indeholder et introduktions-script der beskriver hvordan hver fil (der læses ind som en stor liste i *R*) skal indekkes for at få fat i nogle af de mere interessante resultater.

Scripts

For at køre disse scripts, kræves *H2O*-pakken i *R* (vi har benyttet version 3.16.0.2, men vi kan forestille os at nyere versioner også fungerer).

AllFunctionsExceptEverythingFunction

Indeholder alle de funktioner, der er nødvendige for at køre scripts

Nogle af disse funktioner kan tage hensyn til, at man har *OHLCV* data med mange forskellige valutaer, men dette er overflødigt da der kun arbejdes med *BTC/USD*.

Everything

Den store funktion, der kan køre igennem *walk-forward-optimization*.

Kræver et Linux styresystem pga. af en paralleliseringspakke ("doMC")

EverythingFunction_RdyToRun_WithInputs

Output fra denne vil indeholde ALT det data vi baserer vores resultater fra Walk-Forward-Optimization på. Defineringsfilen "Everything" for funktionen viser at der er mange flere inputs end dem som vi skruer på i denne afhandling, dvs. dem som vi skruer på i "EverythingFunction_RdyToRun_WithInputs". Vi anbefaler at i kun skruer på disse parametre.

Som sagt, denne indeholder vores defaults værdier som input, men alt kan selvfølgelig ændres.

Oplagde interessante ting at ændre:

- De forskellige hyperparameterintervaller (eller faste værdier selvfølgelig) for modellerne, som f.eks. "ann_hyper_params" (ændr kun værdierne og ikke specificer nye hyperparametre fra H2O som vi altså IKKE har benyttet)
- Hvor mange *candles* frem skal der prædikteres: "forecasting_period"
- Skal de default 1 minut *candles* benyttes eller skal de transformeres til f.eks. 15 min, 1 time osv.: "candle_period"
- Hvorhenne skal den første obs. i train i allerførste vindue i *epoch* tid ligge: "start_point_epoch"
- Intervaller for trading signaler der skal benyttes: "sell_signals" og "buy_signals"
- Hvor længe skal der udføres *random-grid-search* (hvis denne er for lav, og data så stort at en model ikke når at blive trænet, f.eks. for ANN *base*, så opstår der fejl. Brug derfor tilstrækkelig lang tid): "max_runtime_secs"

For de følgende parametre skal man være påpasselig med at vinduerne ikke overskrider det data der rent faktisk arbejdes med.

- Hvor mange vinduer skal der være i *WFO*: "n_windows"
- Fordeling, og størrelse, af de forskellige data-splits:
 - "train_days"
 - "test_days"
 - "val_days"

Disse parametre er nok de mest interessante at ændre på.

Kap6_GridSearch_BaseLearners

Søgningen efter de passende hyperparameterintervaller for *base-learners*

Kap6_GridSearch_MetaLearners

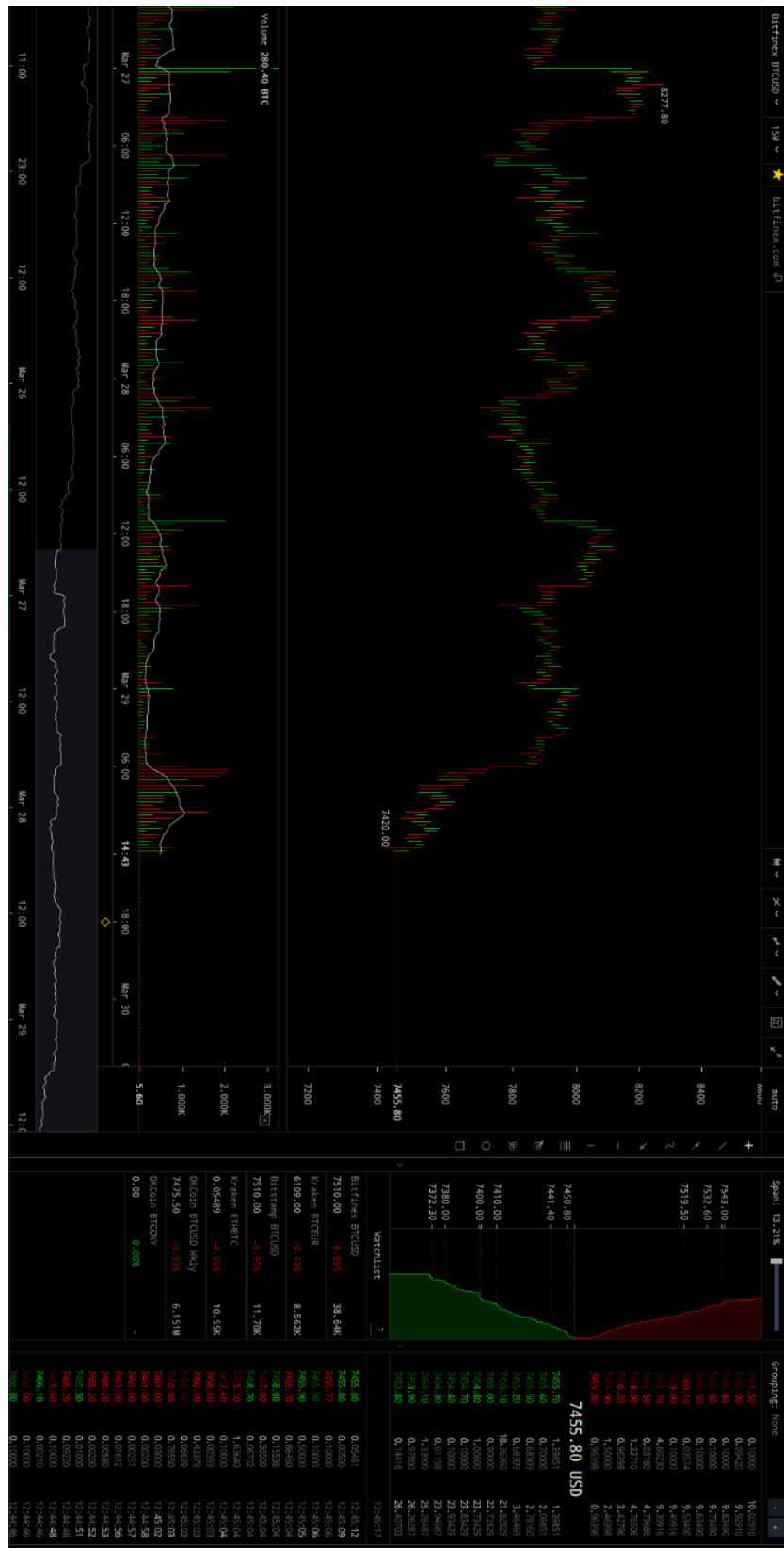
Søgningen efter de passende hyperparameterintervaller for *meta-learners*

Kap7_WFO(forholdvis_rodet)

Forholdsvis rodet script der kan illustrere mange af resultaterne vi viste i dette kapitel.

Vi råder dog til at se på de forskellige "final_return_#" filer fra de forskellige vinduer

104



Modelling

GLM

```
> glm_fit_1.coefs
      glm_fit_10model$coefficients_table$coefficients      glm_fit_base.coefs      glm_fit_base0model$coefficients_table$coefficients
Intercept      9.684994e-05      Intercept      8.797774e-05
BTC_USD_SMA_short_15      0.000000e+00      BTC_USD_SMA_short_15      1.594328e-03
BTC_USD_SMA_short_15_diff_1      0.000000e+00      BTC_USD_SMA_short_15_diff_1      1.286730e-03
BTC_USD_SMA_short_15_diff_5      0.000000e+00      BTC_USD_SMA_short_15_diff_5      -2.154061e-04
BTC_USD_SMA_long_50      0.000000e+00      BTC_USD_SMA_long_50      -6.294367e-04
BTC_USD_SMA_long_50_diff_1      0.000000e+00      BTC_USD_SMA_long_50_diff_1      1.558996e-05
BTC_USD_SMA_long_50_diff_5      0.000000e+00      BTC_USD_SMA_long_50_diff_5      -1.948569e-04
BTC_USD_RSI_14      0.000000e+00      BTC_USD_RSI_14      5.586614e-04
BTC_USD_RSI_14_diff_1      0.000000e+00      BTC_USD_RSI_14_diff_1      -1.134025e-04
BTC_USD_RSI_14_diff_5      0.000000e+00      BTC_USD_RSI_14_diff_5      -2.133360e-04
BTC_USD_MACD_price_10_26_9      0.000000e+00      BTC_USD_MACD_price_10_26_9      1.232669e-02
BTC_USD_MACD_price_10_26_9_diff_1      0.000000e+00      BTC_USD_MACD_price_10_26_9_diff_1      6.130836e-03
BTC_USD_MACD_price_10_26_9_diff_5      0.000000e+00      BTC_USD_MACD_price_10_26_9_diff_5      3.112677e-03
BTC_USD_MACD_signal_10_26_9      0.000000e+00      BTC_USD_MACD_signal_10_26_9      1.314506e-02
BTC_USD_MACD_signal_10_26_9_diff_1      0.000000e+00      BTC_USD_MACD_signal_10_26_9_diff_1      2.628524e-04
BTC_USD_MACD_signal_10_26_9_diff_5      0.000000e+00      BTC_USD_MACD_signal_10_26_9_diff_5      4.555356e-03
BTC_USD_SAR_0_025_0_05      0.000000e+00      BTC_USD_SAR_0_025_0_05      2.161444e-04
BTC_USD_SAR_0_025_0_05_diff_1      0.000000e+00      BTC_USD_SAR_0_025_0_05_diff_1      -6.083997e-06
BTC_USD_SAR_0_025_0_05_diff_5      0.000000e+00      BTC_USD_SAR_0_025_0_05_diff_5      -4.522370e-05
BTC_USD_Aroon_up_25      0.000000e+00      BTC_USD_Aroon_up_25      2.028449e-05
BTC_USD_Aroon_up_25_diff_1      0.000000e+00      BTC_USD_Aroon_up_25_diff_1      5.751635e-06
BTC_USD_Aroon_up_25_diff_5      0.000000e+00      BTC_USD_Aroon_up_25_diff_5      3.903079e-05
BTC_USD_Aroon_down_25      0.000000e+00      BTC_USD_Aroon_down_25      -2.086714e-06
BTC_USD_Aroon_down_25_diff_1      0.000000e+00      BTC_USD_Aroon_down_25_diff_1      -9.031500e-06
BTC_USD_Aroon_down_25_diff_5      0.000000e+00      BTC_USD_Aroon_down_25_diff_5      -2.316569e-05
BTC_USD_Aroon_oscillator_25      0.000000e+00      BTC_USD_Aroon_oscillator_25      1.365011e-05
BTC_USD_Aroon_oscillator_25_diff_1      0.000000e+00      BTC_USD_Aroon_oscillator_25_diff_1      1.033662e-05
BTC_USD_Aroon_oscillator_25_diff_5      0.000000e+00      BTC_USD_Aroon_oscillator_25_diff_5      4.127618e-05
BTC_USD_stochosc_FastK_14      0.000000e+00      BTC_USD_stochosc_FastK_14      9.922989e-06
BTC_USD_stochosc_FastK_14_diff_1      0.000000e+00      BTC_USD_stochosc_FastK_14_diff_1      -5.285503e-05
BTC_USD_stochosc_FastK_14_diff_5      0.000000e+00      BTC_USD_stochosc_FastK_14_diff_5      4.531727e-06
BTC_USD_stochosc_FastD_3      0.000000e+00      BTC_USD_stochosc_FastD_3      -7.123737e-06
BTC_USD_stochosc_FastD_3_diff_1      0.000000e+00      BTC_USD_stochosc_FastD_3_diff_1      1.098454e-05
BTC_USD_stochosc_FastD_3_diff_5      0.000000e+00      BTC_USD_stochosc_FastD_3_diff_5      -2.981380e-05
BTC_USD_stochosc_SlowD_3      0.000000e+00      BTC_USD_stochosc_SlowD_3      -3.373575e-05
BTC_USD_stochosc_SlowD_3_diff_1      0.000000e+00      BTC_USD_stochosc_SlowD_3_diff_1      6.110456e-05
BTC_USD_stochosc_SlowD_3_diff_5      0.000000e+00      BTC_USD_stochosc_SlowD_3_diff_5      3.927187e-05
BTC_USD_ADX_Dip_14      0.000000e+00      BTC_USD_ADX_Dip_14      -8.459518e-05
BTC_USD_ADX_Dip_14_diff_1      6.531648e-05      BTC_USD_ADX_Dip_14_diff_1      1.026355e-04
BTC_USD_ADX_Dip_14_diff_5      0.000000e+00      BTC_USD_ADX_Dip_14_diff_5      2.923081e-05
BTC_USD_ADX_Din_14      0.000000e+00      BTC_USD_ADX_Din_14      -4.167859e-05
BTC_USD_ADX_Din_14_diff_1      -2.625555e-05      BTC_USD_ADX_Din_14_diff_1      -8.248808e-05
BTC_USD_ADX_Din_14_diff_5      0.000000e+00      BTC_USD_ADX_Din_14_diff_5      4.489121e-05
BTC_USD_ADX_Dx_14      0.000000e+00      BTC_USD_ADX_Dx_14      -8.215966e-06
BTC_USD_ADX_Dx_14_diff_1      0.000000e+00      BTC_USD_ADX_Dx_14_diff_1      8.345738e-06
BTC_USD_ADX_Dx_14_diff_5      0.000000e+00      BTC_USD_ADX_Dx_14_diff_5      2.102191e-05
BTC_USD_ADX_14      0.000000e+00      BTC_USD_ADX_14      -2.189091e-05
BTC_USD_ADX_14_diff_1      0.000000e+00      BTC_USD_ADX_14_diff_1      5.679537e-06
BTC_USD_ADX_14_diff_5      0.000000e+00      BTC_USD_ADX_14_diff_5      9.585095e-05
BTC_USD_EMA_short_10      0.000000e+00      BTC_USD_EMA_short_10      -5.383353e-03
BTC_USD_EMA_short_10_diff_1      0.000000e+00      BTC_USD_EMA_short_10_diff_1      5.383353e-03
BTC_USD_EMA_short_10_diff_5      0.000000e+00      BTC_USD_EMA_short_10_diff_5      3.171026e-03
BTC_USD_EMA_long_21      0.000000e+00      BTC_USD_EMA_long_21      9.819911e-03
BTC_USD_EMA_long_21_diff_1      0.000000e+00      BTC_USD_EMA_long_21_diff_1      -9.819911e-03
BTC_USD_EMA_long_21_diff_5      0.000000e+00      BTC_USD_EMA_long_21_diff_5      -1.407051e-02
BTC_USD_BolBands_dn_20_2      0.000000e+00      BTC_USD_BolBands_dn_20_2      3.144279e-04
BTC_USD_BolBands_dn_20_2_diff_1      0.000000e+00      BTC_USD_BolBands_dn_20_2_diff_1      1.733433e-04
BTC_USD_BolBands_dn_20_2_diff_5      0.000000e+00      BTC_USD_BolBands_dn_20_2_diff_5      -1.660896e-04
BTC_USD_BolBands_mavg_20_2      0.000000e+00      BTC_USD_BolBands_mavg_20_2      5.698377e-04
BTC_USD_BolBands_mavg_20_2_diff_1      0.000000e+00      BTC_USD_BolBands_mavg_20_2_diff_1      2.856556e-04
BTC_USD_BolBands_mavg_20_2_diff_5      0.000000e+00      BTC_USD_BolBands_mavg_20_2_diff_5      -3.637595e-04
BTC_USD_BolBands_up_20_2      1.061140e-04      BTC_USD_BolBands_up_20_2      5.016551e-04
BTC_USD_BolBands_up_20_2_diff_1      0.000000e+00      BTC_USD_BolBands_up_20_2_diff_1      1.694871e-04
BTC_USD_BolBands_up_20_2_diff_5      0.000000e+00      BTC_USD_BolBands_up_20_2_diff_5      -3.159177e-04
BTC_USD_BolBands_pct_20_2      0.000000e+00      BTC_USD_BolBands_pct_20_2      5.973884e-05
BTC_USD_BolBands_pct_20_2_diff_1      0.000000e+00      BTC_USD_BolBands_pct_20_2_diff_1      3.195502e-06
BTC_USD_BolBands_pct_20_2_diff_5      9.261335e-05      BTC_USD_BolBands_pct_20_2_diff_5      1.029998e-04
BTC_USD_TDI_transformed_20_2      0.000000e+00      BTC_USD_TDI_transformed_20_2      1.461826e-04
BTC_USD_TDI_transformed_20_2_diff_1      0.000000e+00      BTC_USD_TDI_transformed_20_2_diff_1      7.017425e-05
BTC_USD_TDI_transformed_20_2_diff_5      0.000000e+00      BTC_USD_TDI_transformed_20_2_diff_5      -1.459488e-04
BTC_USD_TDI_transformed_d1_20_2      0.000000e+00      BTC_USD_TDI_transformed_d1_20_2      -7.477678e-04
BTC_USD_TDI_transformed_d1_20_2_diff_1      0.000000e+00      BTC_USD_TDI_transformed_d1_20_2_diff_1      -1.809706e-04
BTC_USD_TDI_transformed_d1_20_2_diff_5      0.000000e+00      BTC_USD_TDI_transformed_d1_20_2_diff_5      2.493360e-04
BTC_USD_ROC_1      0.000000e+00      BTC_USD_ROC_1      3.054870e-03
BTC_USD_ROC_1_diff_1      5.237996e-06      BTC_USD_ROC_1_diff_1      4.626528e-04
BTC_USD_ROC_1_diff_5      0.000000e+00      BTC_USD_ROC_1_diff_5      -2.307672e-04
BTC_USD_ROC_momentum_transformed_1      0.000000e+00      BTC_USD_ROC_momentum_transformed_1      3.042123e-03
BTC_USD_ROC_momentum_transformed_1_diff_1      5.164094e-06      BTC_USD_ROC_momentum_transformed_1_diff_1      4.788838e-04
BTC_USD_ROC_momentum_transformed_1_diff_5      0.000000e+00      BTC_USD_ROC_momentum_transformed_1_diff_5      1.685663e-04
BTC_USD_MFI_14      0.000000e+00      BTC_USD_MFI_14      -2.192340e-04
BTC_USD_MFI_14_diff_1      2.368215e-05      BTC_USD_MFI_14_diff_1      4.421396e-05
BTC_USD_MFI_14_diff_5      0.000000e+00      BTC_USD_MFI_14_diff_5      1.111744e-04
BTC_USD_OBV_SMA_OBV_SMA-VOL      0.000000e+00      BTC_USD_OBV_SMA_OBV_SMA-VOL      3.352493e-05
BTC_USD_OBV_SMA_OBV_SMA-VOL_diff_1      0.000000e+00      BTC_USD_OBV_SMA_OBV_SMA-VOL_diff_1      -2.139009e-05
BTC_USD_stochRSI_FastD_14      7.809700e-04      BTC_USD_stochRSI_FastD_14      -3.480237e-05
BTC_USD_stochRSI_FastD_14_diff_1      0.000000e+00      BTC_USD_stochRSI_FastD_14_diff_1      4.816184e-05
BTC_USD_stochRSI_FastD_14_diff_5      0.000000e+00      BTC_USD_stochRSI_FastD_14_diff_5      -1.249969e-04
BTC_USD_stochRSI_SlowD_14      0.000000e+00      BTC_USD_stochRSI_SlowD_14      4.984372e-05
BTC_USD_stochRSI_SlowD_14_diff_1      0.000000e+00      BTC_USD_stochRSI_SlowD_14_diff_1      -1.347655e-05
BTC_USD_SMA_very_short_5      0.000000e+00      BTC_USD_SMA_very_short_5      -9.205469e-04
BTC_USD_SMA_very_short_5_diff_1      -1.586137e-04      BTC_USD_SMA_very_short_5_diff_1      4.544056e-04
BTC_USD_SMA_very_short_5_diff_5      0.000000e+00      BTC_USD_SMA_very_short_5_diff_5      2.119205e-04
BTC_USD_SMA_very_long_200      0.000000e+00      BTC_USD_SMA_very_long_200      4.762928e-04
BTC_USD_SMA_very_long_200_diff_1      -8.430167e-04      BTC_USD_SMA_very_long_200_diff_1      1.407005e-05
BTC_USD_SMA_very_long_200_diff_5      0.000000e+00      BTC_USD_SMA_very_long_200_diff_5      -1.128544e-03
BTC_USD_MACD_price_50_130_45      0.000000e+00      BTC_USD_MACD_price_50_130_45      -2.082892e-03
BTC_USD_MACD_price_50_130_45_diff_1      0.000000e+00      BTC_USD_MACD_price_50_130_45_diff_1      -8.209242e-03
BTC_USD_MACD_price_50_130_45_diff_5      0.000000e+00      BTC_USD_MACD_price_50_130_45_diff_5      4.208257e-03
BTC_USD_MACD_signal_50_130_45      -1.345721e-04      BTC_USD_MACD_signal_50_130_45      -1.157155e-05
BTC_USD_MACD_signal_50_130_45_diff_1      0.000000e+00      BTC_USD_MACD_signal_50_130_45_diff_1      -5.480272e-05
BTC_USD_MACD_signal_50_130_45_diff_5      0.000000e+00      BTC_USD_MACD_signal_50_130_45_diff_5      9.947076e-04
BTC_USD_TDI_transformed_100_10      0.000000e+00      BTC_USD_TDI_transformed_100_10      4.604273e-05
BTC_USD_TDI_transformed_100_10_diff_1      0.000000e+00      BTC_USD_TDI_transformed_100_10_diff_1      1.369478e-04
BTC_USD_TDI_transformed_100_10_diff_5      0.000000e+00      BTC_USD_TDI_transformed_100_10_diff_5      -5.695900e-05
BTC_USD_TDI_transformed_d1_100_10      0.000000e+00      BTC_USD_TDI_transformed_d1_100_10      -2.623898e-04
BTC_USD_TDI_transformed_d1_100_10_diff_1      0.000000e+00      BTC_USD_TDI_transformed_d1_100_10_diff_1      2.917571e-04
BTC_USD_TDI_transformed_d1_100_10_diff_5      0.000000e+00      BTC_USD_TDI_transformed_d1_100_10_diff_5      -1.328425e-03
BTC_USD_BolBands_dn_100_10      0.000000e+00      BTC_USD_BolBands_dn_100_10      5.428267e-04
BTC_USD_BolBands_dn_100_10_diff_1      0.000000e+00      BTC_USD_BolBands_dn_100_10_diff_1      -2.073143e-06
BTC_USD_BolBands_dn_100_10_diff_5      0.000000e+00      BTC_USD_BolBands_dn_100_10_diff_5      3.138303e-04
BTC_USD_BolBands_mavg_100_10      0.000000e+00      BTC_USD_BolBands_mavg_100_10      1.584580e-03
BTC_USD_BolBands_mavg_100_10_diff_1      0.000000e+00      BTC_USD_BolBands_mavg_100_10_diff_1      1.156310e-05
BTC_USD_BolBands_mavg_100_10_diff_5      0.000000e+00      BTC_USD_BolBands_mavg_100_10_diff_5      1.386801e-03
BTC_USD_BolBands_up_100_10      0.000000e+00      BTC_USD_BolBands_up_100_10      4.920682e-04
BTC_USD_BolBands_up_100_10_diff_1      0.000000e+00      BTC_USD_BolBands_up_100_10_diff_1      5.452396e-06
BTC_USD_BolBands_up_100_10_diff_5      0.000000e+00      BTC_USD_BolBands_up_100_10_diff_5      7.947907e-05
BTC_USD_BolBands_pct_100_10      0.000000e+00      BTC_USD_BolBands_pct_100_10      3.511033e-05
BTC_USD_BolBands_pct_100_10_diff_1      0.000000e+00      BTC_USD_BolBands_pct_100_10_diff_1      -1.146861e-04
BTC_USD_BolBands_pct_100_10_diff_5      0.000000e+00      BTC_USD_BolBands_pct_100_10_diff_5      2.849831e-05
```

GRID SEARCH

GBM best random-grid-search

```
> model_grid_search_gbm4@summary_table[1:20,c(1:7,9)]
```

	col_sample_rate	learn_rate	learn_rate_annealing	max_depth	min_rows	ntrees	sample_rate	mae
1	0.8	0.032	0.995	45	90.0	75	0.2	0.004085440388070786
2	0.8	0.024	0.995	45	80.0	75	0.15	0.004086493184820965
3	0.6	0.024	0.995	50	70.0	75	0.2	0.004087529102040031
4	0.8	0.032	0.99	50	100.0	75	0.15	0.004087532717228769
5	0.8	0.02	0.995	35	100.0	75	0.2	0.00408809901554956
6	0.6	0.024	0.995	40	70.0	75	0.15	0.004088423553741081
7	0.6	0.028	0.99	30	110.0	75	0.2	0.004088636416142258
8	1.0	0.02	0.985	40	70.0	75	0.1	0.004089037413613877
9	0.4	0.032	0.985	30	100.0	75	0.1	0.004089168069076712
10	1.0	0.016	0.995	50	60.0	75	0.15	0.00408916826333938
11	0.4	0.016	0.995	45	60.0	75	0.2	0.0040906134858498325
12	0.2	0.028	0.99	30	60.0	75	0.1	0.004091122477493243
13	1.0	0.028	0.99	30	80.0	75	0.2	0.004091715219047302
14	0.6	0.028	0.99	30	70.0	75	0.15	0.004091734380080738
15	1.0	0.012	0.995	50	60.0	75	0.15	0.004091866026960481
16	0.2	0.024	0.995	40	100.0	75	0.1	0.004091909912311535
17	0.6	0.028	0.99	35	90.0	75	0.15	0.004092302089378981
18	1.0	0.028	0.985	50	80.0	75	0.2	0.004092376913128299
19	0.8	0.016	0.995	30	100.0	75	0.2	0.0040929384416131896
20	0.8	0.016	0.985	50	60.0	75	0.2	0.004093121945184722

ANN best random-grid-search

```
> model_grid_search_ann5@summary_table[1:20,c(1:8,11)]
```

	activation	epochs	epsilon	hidden	hidden_dropout_ratios	l2	loss	rho	mae
1	RectifierWithDropout	20.0	1.0E-9	[100, 100, 100]	[0.3, 0.3, 0.3]	1.0E-6	Absolute	0.99	0.004063757121481134
2	RectifierWithDropout	20.0	1.0E-9	[100, 300, 100]	[0.5, 0.5, 0.5]	1.0E-5	Absolute	0.99	0.004064902059788394
3	RectifierWithDropout	20.0	1.0E-9	[300, 200, 100]	[0.6, 0.6, 0.6]	1.0E-5	Absolute	0.99	0.004069701294503769
4	RectifierWithDropout	20.0	1.0E-9	[200, 200, 200]	[0.5, 0.5, 0.5]	1.0E-7	Absolute	0.8	0.004071772863562032
5	RectifierWithDropout	20.0	1.0E-9	[300, 200, 100]	[0.6, 0.6, 0.6]	1.0E-5	Absolute	0.9	0.004072495895144116
6	RectifierWithDropout	20.0	1.0E-9	[100, 100, 100]	[0.5, 0.5, 0.5]	1.0E-8	Absolute	0.99	0.004072784303006148
7	RectifierWithDropout	20.0	1.0E-8	[200, 200, 200]	[0.6, 0.6, 0.6]	1.0E-5	Absolute	0.99	0.004073276871676716
8	RectifierWithDropout	20.0	1.0E-8	[100, 100, 100]	[0.6, 0.6, 0.6]	1.0E-7	Absolute	0.8	0.00407376960649223
9	RectifierWithDropout	20.0	1.0E-9	[100, 100, 100]	[0.6, 0.6, 0.6]	1.0E-8	Absolute	0.9	0.004074005980703862
10	RectifierWithDropout	20.0	1.0E-10	[100, 100, 100]	[0.6, 0.6, 0.6]	1.0E-4	Absolute	0.99	0.004075225141387534
11	RectifierWithDropout	20.0	1.0E-10	[200, 200, 200]	[0.3, 0.3, 0.3]	1.0E-7	Absolute	0.8	0.004075555372534134
12	RectifierWithDropout	20.0	1.0E-9	[100, 300, 100]	[0.5, 0.5, 0.5]	1.0E-8	Absolute	0.9	0.0040769398977464335
13	RectifierWithDropout	20.0	1.0E-8	[200, 200, 200]	[0.4, 0.4, 0.4]	1.0E-6	Absolute	0.8	0.0040774982092473
14	RectifierWithDropout	20.0	1.0E-9	[200, 200, 200]	[0.6, 0.6, 0.6]	1.0E-4	Absolute	0.8	0.004077853387808983
15	RectifierWithDropout	20.0	1.0E-8	[100, 100, 100]	[0.3, 0.3, 0.3]	1.0E-5	Absolute	0.8	0.004078054863351884
16	RectifierWithDropout	20.0	1.0E-8	[200, 200, 200]	[0.5, 0.5, 0.5]	1.0E-5	Absolute	0.8	0.0040780726883124655
17	RectifierWithDropout	20.0	1.0E-8	[100, 300, 100]	[0.3, 0.3, 0.3]	1.0E-6	Absolute	0.8	0.004078451463706392
18	RectifierWithDropout	20.0	1.0E-10	[300, 200, 100]	[0.5, 0.5, 0.5]	1.0E-4	Absolute	0.8	0.004090991307990856
19	RectifierWithDropout	20.0	1.0E-10	[300, 200, 100]	[0.6, 0.6, 0.6]	1.0E-7	Absolute	0.99	0.004096773840602151
20	RectifierWithDropout	20.0	1.0E-10	[100, 100, 100]	[0.6, 0.6, 0.6]	1.0E-5	Absolute	0.8	0.004098436098994798

STACKING

META EN:

```
> meta_model_grid_glm3@summary_table[1:20,]
Hyper-Parameter Search Summary: ordered by increasing mae
  alpha  lambda  model_ids  mae
1 [0.25] [0.005] meta_model_glm_grid3_model_68 0.004086616130059974
2 [0.6] [0.002] meta_model_glm_grid3_model_12 0.004086621225658704
3 [0.4] [0.003] meta_model_glm_grid3_model_29 0.004086625017892273
4 [0.3] [0.004] meta_model_glm_grid3_model_48 0.0040866288519443945
5 [0.2] [0.006] meta_model_glm_grid3_model_88 0.004086636726005306
6 [0.15] [0.008] meta_model_glm_grid3_model_129 0.004086644806507982
7 [0.1] [0.012] meta_model_glm_grid3_model_212 0.00408666274430053
8 [0.45] [0.003] meta_model_glm_grid3_model_30 0.004086966731029041
9 [0.1] [0.011] meta_model_glm_grid3_model_191 0.004086999523101129
10 [0.15] [0.009] meta_model_glm_grid3_model_150 0.004087029443993937
11 [0.65] [0.002] meta_model_glm_grid3_model_13 0.004087073914717759
12 [0.55] [0.002] meta_model_glm_grid3_model_11 0.0040871346146948186
13 [0.1] [0.013] meta_model_glm_grid3_model_233 0.0040871926115776044
14 [0.7] [0.002] meta_model_glm_grid3_model_14 0.004087746188786205
15 [0.35] [0.004] meta_model_glm_grid3_model_49 0.004087775934187255
16 [0.2] [0.007] meta_model_glm_grid3_model_109 0.004087825427946622
17 [0.1] [0.014] meta_model_glm_grid3_model_254 0.004087938834858945
18 [0.15] [0.007] meta_model_glm_grid3_model_108 0.00408801282090101
19 [0.35] [0.003] meta_model_glm_grid3_model_28 0.004088125408248806
20 [0.05] [0.02] meta_model_glm_grid3_model_379 0.0040885338822976985
```

META RF:

```
> meta_model_grid_rf2@summary_table[1:20,]
Hyper-Parameter Search Summary: ordered by increasing mae
  col_sample_rate_per_tree max_depth min_rows mtries nbins ntrees sample_rate  model_ids  mae
1 1.0 4 10.0 2 30 100 0.9 meta_model_rf_grid2_model_3 0.0041530786757492675
2 1.0 8 30.0 1 50 100 0.8 meta_model_rf_grid2_model_12 0.004156167116328067
3 1.0 8 30.0 1 40 100 0.8 meta_model_rf_grid2_model_35 0.004156619149627831
4 1.0 8 30.0 1 50 100 1.0 meta_model_rf_grid2_model_20 0.004157937200229528
5 1.0 8 5.0 1 50 100 0.8 meta_model_rf_grid2_model_10 0.004158223785829755
6 1.0 8 10.0 1 50 100 0.9 meta_model_rf_grid2_model_21 0.004158228333653735
7 1.0 12 5.0 1 50 100 0.9 meta_model_rf_grid2_model_27 0.004186989021672711
8 1.0 12 5.0 1 40 100 1.0 meta_model_rf_grid2_model_29 0.004187454172527267
9 1.0 8 5.0 2 30 100 0.9 meta_model_rf_grid2_model_31 0.004187985006814217
10 1.0 8 30.0 2 40 100 0.9 meta_model_rf_grid2_model_2 0.004188686257880682
11 1.0 8 30.0 2 50 100 1.0 meta_model_rf_grid2_model_34 0.004190210929377323
12 1.0 4 30.0 4 40 100 1.0 meta_model_rf_grid2_model_17 0.004195426586958434
13 1.0 4 10.0 4 40 100 1.0 meta_model_rf_grid2_model_32 0.004195426586958434
14 1.0 8 10.0 3 40 100 0.9 meta_model_rf_grid2_model_33 0.004198984839857848
15 1.0 8 30.0 3 30 100 0.9 meta_model_rf_grid2_model_28 0.0041991795206410415
16 1.0 20 30.0 1 30 100 0.9 meta_model_rf_grid2_model_16 0.004201239428952928
17 1.0 8 30.0 4 30 100 0.9 meta_model_rf_grid2_model_1 0.0042031072787625315
18 1.0 20 30.0 1 50 100 0.9 meta_model_rf_grid2_model_22 0.00420489825604078
19 1.0 16 10.0 1 30 100 1.0 meta_model_rf_grid2_model_15 0.00420492012110166
20 1.0 12 10.0 2 50 100 0.8 meta_model_rf_grid2_model_7 0.0042099281887953005
```

META GBM:

```
> meta_model_grid_gbm3@summary_table[1:20,]
Hyper-Parameter Search Summary: ordered by increasing mae
col_sample_rate learn_rate learn_rate_annealing max_depth min_rows ntreesh sample_rate model_ids mae
1 1.0 0.007 0.984 4 130.0 150 0.92 meta_model_gbm_grid3_model_7 0.004085832930175053
2 1.0 0.006 0.988 5 110.0 150 0.96 meta_model_gbm_grid3_model_10 0.0040872940889968779
3 1.0 0.005 0.99 5 120.0 150 0.92 meta_model_gbm_grid3_model_3 0.004087516498658137
4 1.0 0.01 0.972 5 110.0 150 0.9 meta_model_gbm_grid3_model_21 0.004088024410163023
5 1.0 0.008 0.978 5 140.0 150 0.94 meta_model_gbm_grid3_model_18 0.004088184656288224
6 1.0 0.006 0.984 5 110.0 150 0.94 meta_model_gbm_grid3_model_1 0.004088418322116542
7 1.0 0.009 0.976 6 120.0 150 0.9 meta_model_gbm_grid3_model_20 0.004088858726636244
8 1.0 0.006 0.982 5 130.0 150 0.98 meta_model_gbm_grid3_model_8 0.004089355173252125
9 1.0 0.009 0.972 6 120.0 150 1.0 meta_model_gbm_grid3_model_15 0.004090057709697995
10 1.0 0.008 0.98 7 120.0 150 0.9 meta_model_gbm_grid3_model_27 0.004090131062629708
11 1.0 0.007 0.982 8 110.0 150 0.9 meta_model_gbm_grid3_model_4 0.004091453663708785
12 1.0 0.01 0.986 6 120.0 150 0.94 meta_model_gbm_grid3_model_16 0.004091838275638113
13 1.0 0.006 0.984 9 130.0 150 0.98 meta_model_gbm_grid3_model_14 0.004092839588718445
14 1.0 0.009 0.98 9 110.0 150 1.0 meta_model_gbm_grid3_model_12 0.0040928724997912665
15 1.0 0.008 0.97 7 100.0 150 0.92 meta_model_gbm_grid3_model_29 0.004092907420902503
16 1.0 0.007 0.97 5 130.0 150 0.98 meta_model_gbm_grid3_model_24 0.004093024055682295
17 1.0 0.01 0.974 10 140.0 150 0.92 meta_model_gbm_grid3_model_6 0.004093131526453987
18 1.0 0.008 0.986 9 140.0 150 0.9 meta_model_gbm_grid3_model_9 0.004093314960412004
19 1.0 0.01 0.98 10 140.0 150 0.9 meta_model_gbm_grid3_model_11 0.004093752670353225
20 1.0 0.007 0.978 10 100.0 150 1.0 meta_model_gbm_grid3_model_5 0.0040938633684538895
```

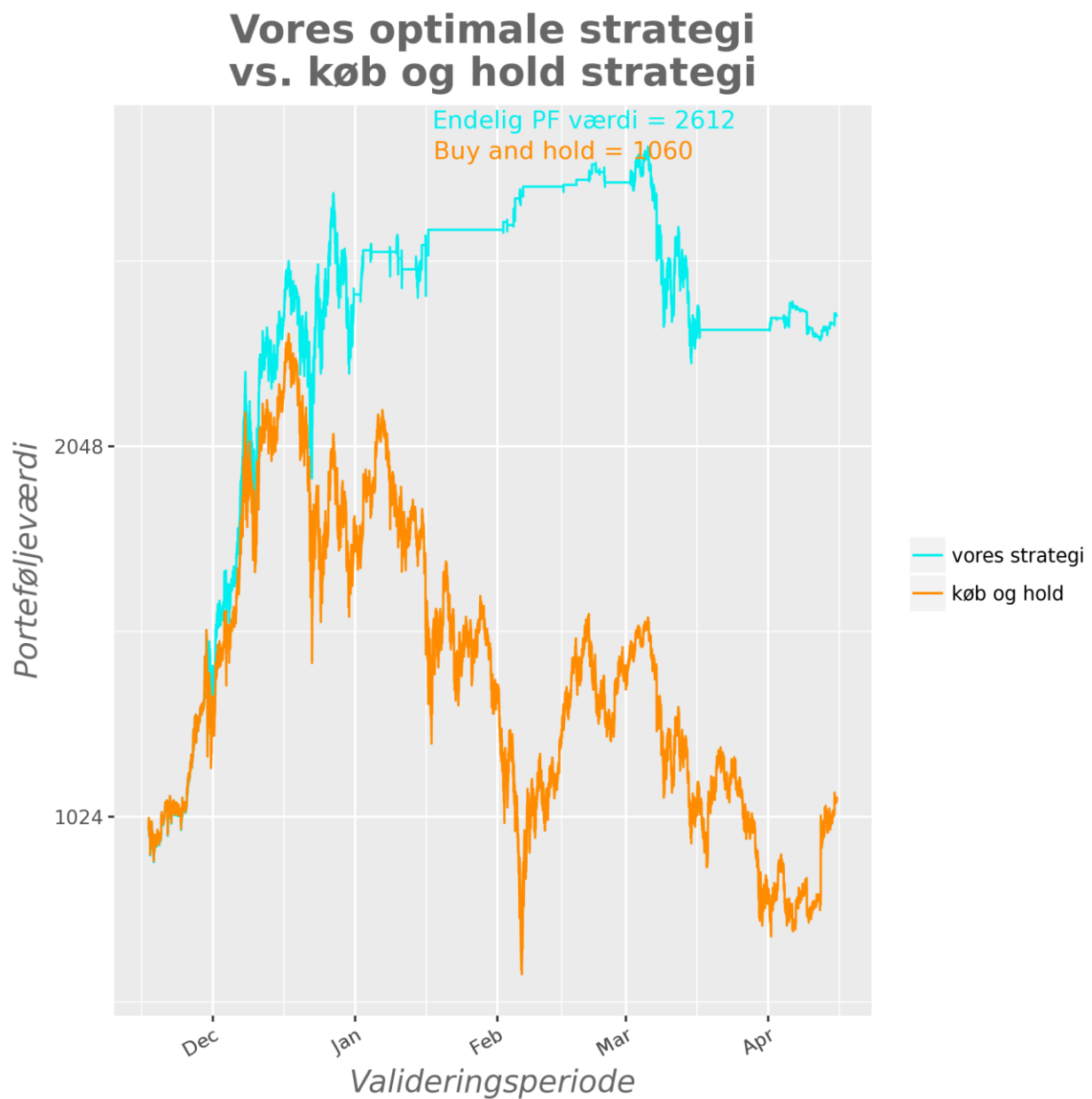
META ANN:

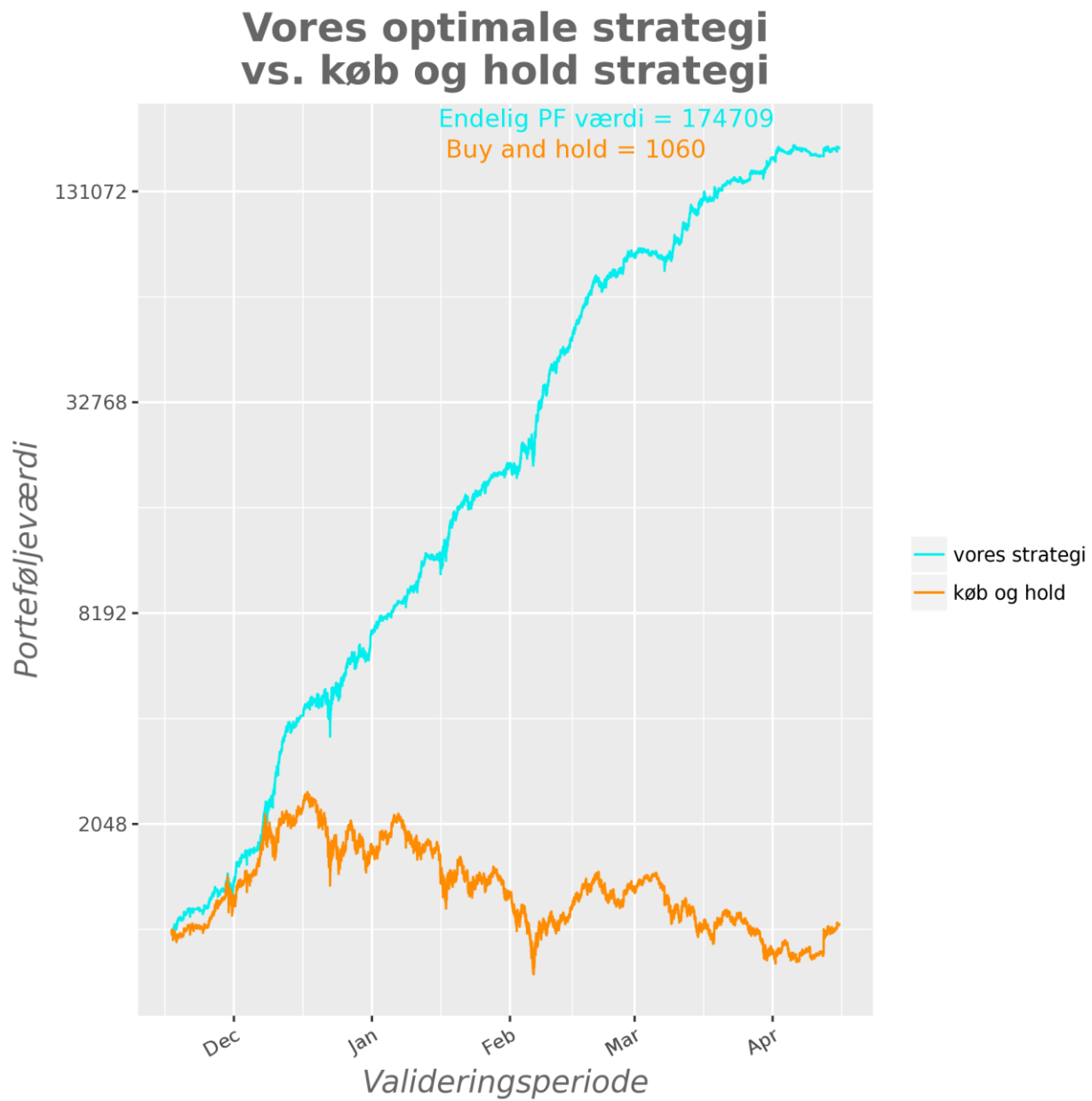
```
> meta_model_grid_ann_final4@summary_table[1:20,]
Hyper-Parameter Search Summary: ordered by increasing mae
activation epochs epsilon hidden hidden_dropout ratios l2 loss rho model_ids mae
1 RectifierWithDropout 1.0 1.0E-8 [80, 40, 20] [0.5, 0.5, 0.5] 1.0E-4 Absolute 0.99 meta_model_ann_grid_final4_model_14 0.004069525048389494
2 RectifierWithDropout 2.0 1.0E-10 [50, 100, 50] [0.3, 0.3, 0.3] 2.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_48 0.004070074307689585
3 RectifierWithDropout 1.0 1.0E-8 [50, 100, 50] [0.5, 0.5, 0.5] 2.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_39 0.004072564517511994
4 RectifierWithDropout 2.0 1.0E-10 [80, 40, 20] [0.3, 0.3, 0.3] 4.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_18 0.0040729177113128295
5 RectifierWithDropout 1.0 1.0E-9 [300, 40, 25] [0.5, 0.5, 0.5] 4.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_10 0.004073678448807051
6 RectifierWithDropout 1.0 1.0E-10 [300, 40, 25] [0.5, 0.5, 0.5] 6.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_19 0.004073786128704451
7 RectifierWithDropout 2.0 1.0E-9 [50, 100, 50] [0.5, 0.5, 0.5] 2.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_30 0.004073851114152466
8 RectifierWithDropout 1.0 1.0E-9 [50, 100, 50] [0.7, 0.7, 0.7] 2.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_3 0.004074260980196992
9 RectifierWithDropout 2.0 1.0E-10 [80, 40, 20] [0.5, 0.5, 0.5] 1.0E-4 Absolute 0.99 meta_model_ann_grid_final4_model_17 0.004074548020587056
10 RectifierWithDropout 1.0 1.0E-8 [200, 30, 15] [0.5, 0.5, 0.5] 4.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_26 0.004074915463742262
11 RectifierWithDropout 4.0 1.0E-8 [50, 100, 50] [0.7, 0.7, 0.7] 4.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_25 0.00407504351835628
12 RectifierWithDropout 2.0 1.0E-8 [200, 30, 15] [0.3, 0.3, 0.3] 8.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_21 0.004077115111425141
13 RectifierWithDropout 2.0 1.0E-9 [80, 40, 20] [0.5, 0.5, 0.5] 4.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_2 0.0040776208652733535
14 RectifierWithDropout 2.0 1.0E-8 [200, 30, 15] [0.7, 0.7, 0.7] 8.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_20 0.004077724024442105
15 RectifierWithDropout 1.0 1.0E-8 [200, 30, 15] [0.5, 0.5, 0.5] 2.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_5 0.004078096325927771
16 RectifierWithDropout 2.0 1.0E-9 [200, 30, 15] [0.7, 0.7, 0.7] 8.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_34 0.004078408556731279
17 RectifierWithDropout 4.0 1.0E-8 [50, 100, 50] [0.5, 0.5, 0.5] 4.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_23 0.004079445348647925
18 RectifierWithDropout 5.0 1.0E-10 [200, 30, 15] [0.7, 0.7, 0.7] 4.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_31 0.004079841941958816
19 RectifierWithDropout 2.0 1.0E-8 [300, 40, 25] [0.5, 0.5, 0.5] 1.0E-4 Absolute 0.99 meta_model_ann_grid_final4_model_22 0.00408019256341226
20 RectifierWithDropout 4.0 1.0E-8 [80, 40, 20] [0.5, 0.5, 0.5] 2.0E-5 Absolute 0.99 meta_model_ann_grid_final4_model_16 0.004080361314957412
```

Walk-forward-optimization

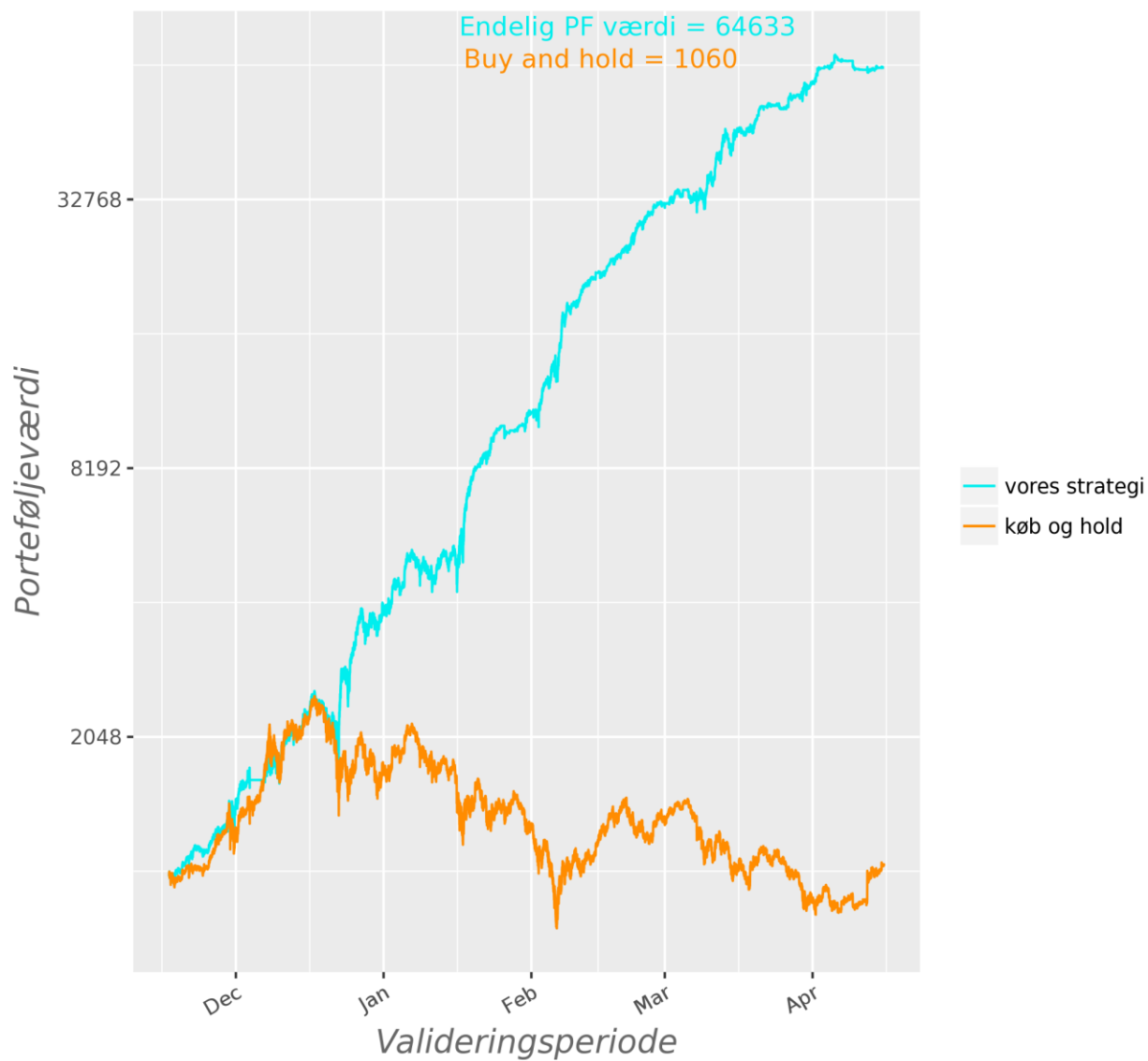
Resultater fra walk-forward-optimization

EN base-learner porteføljeværdi

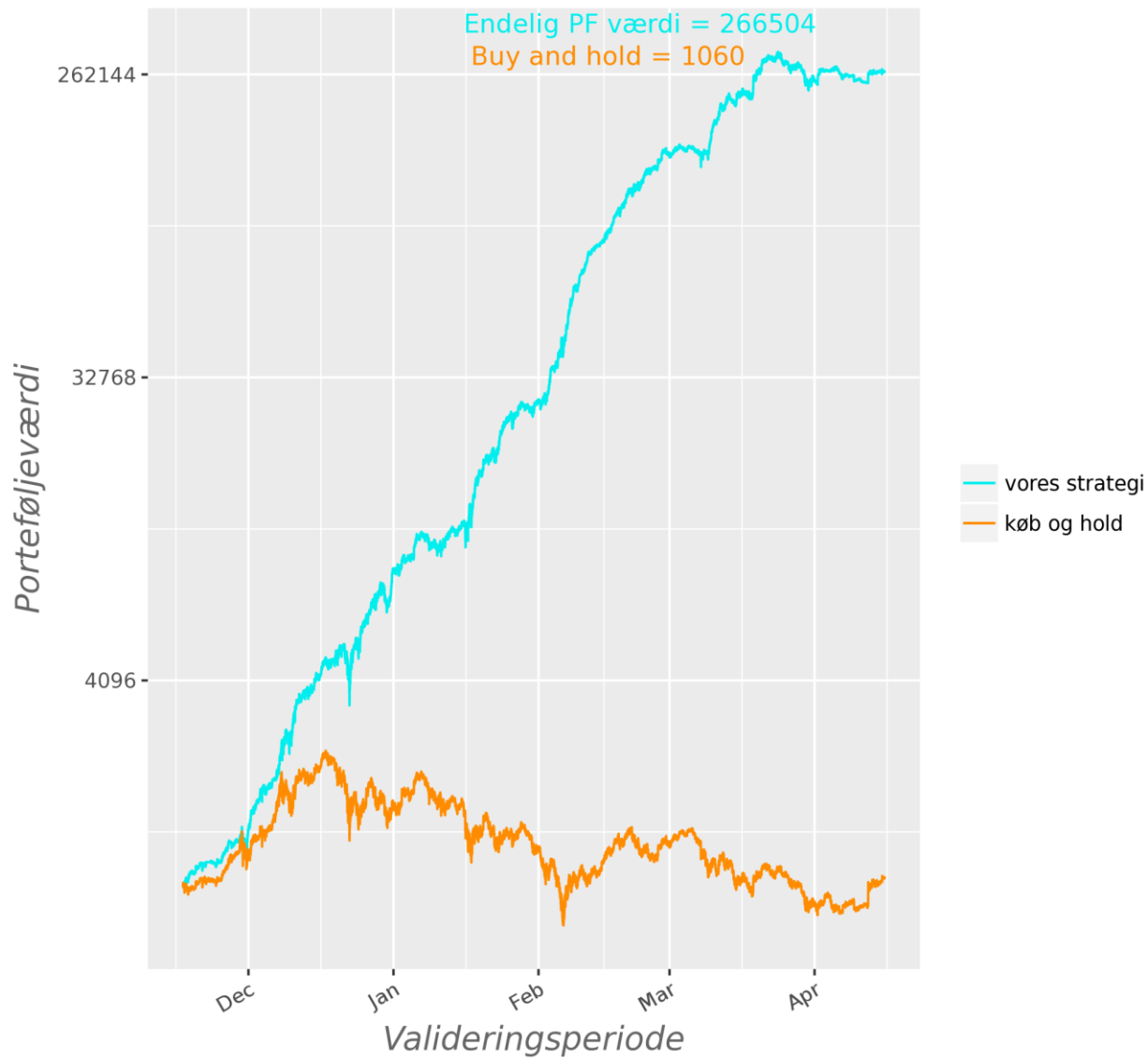


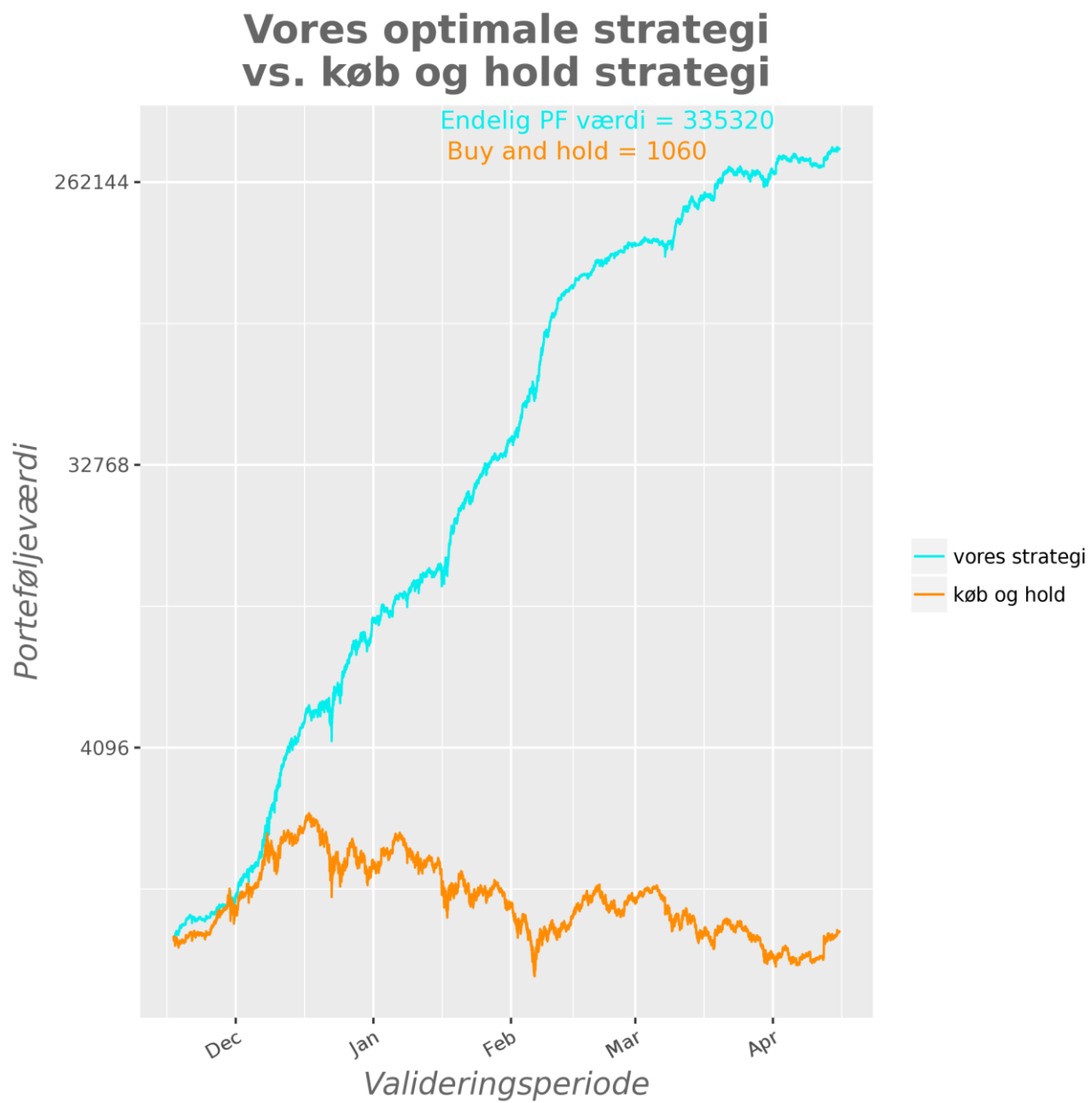


Vores optimale strategi vs. køb og hold strategi

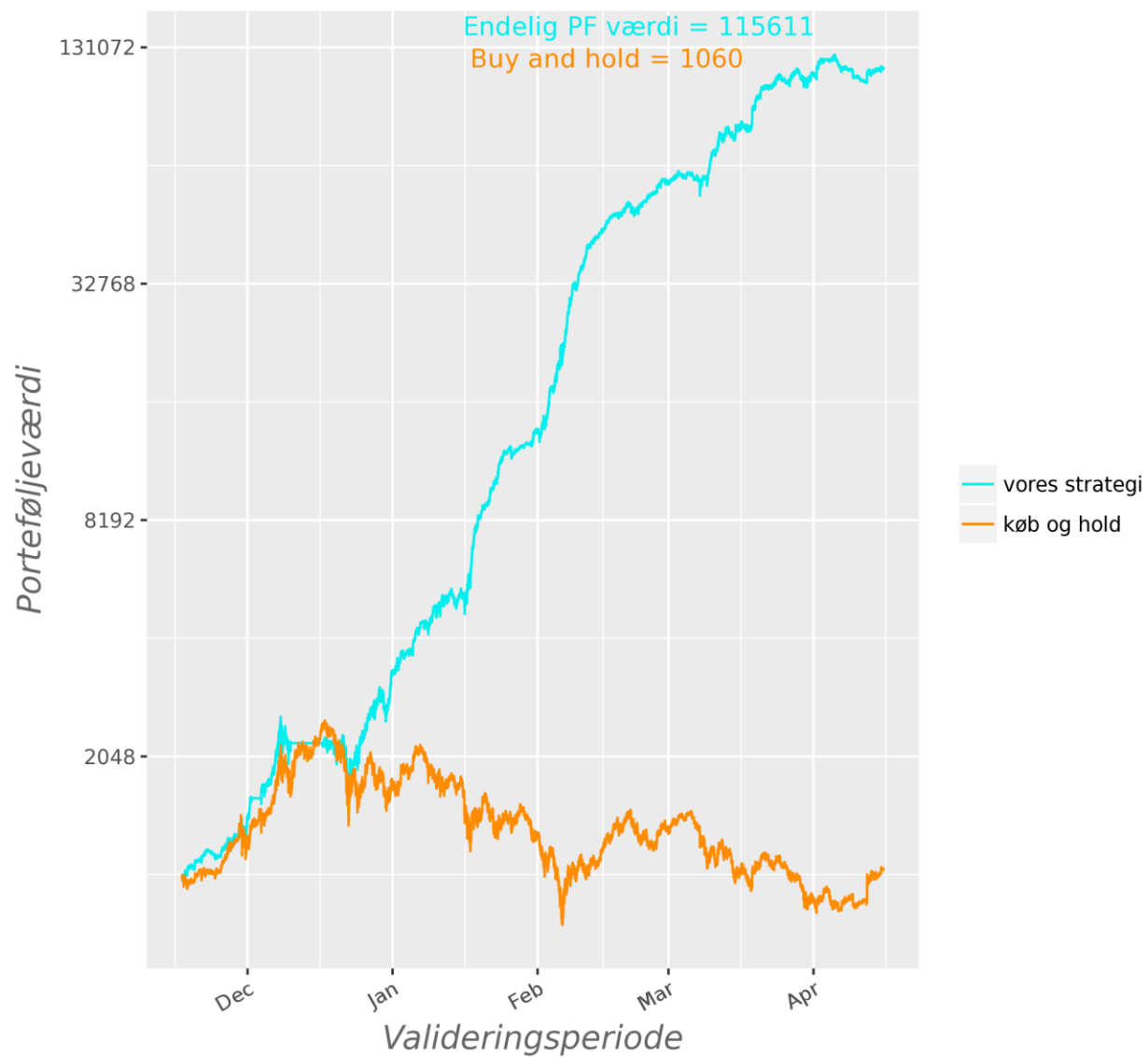


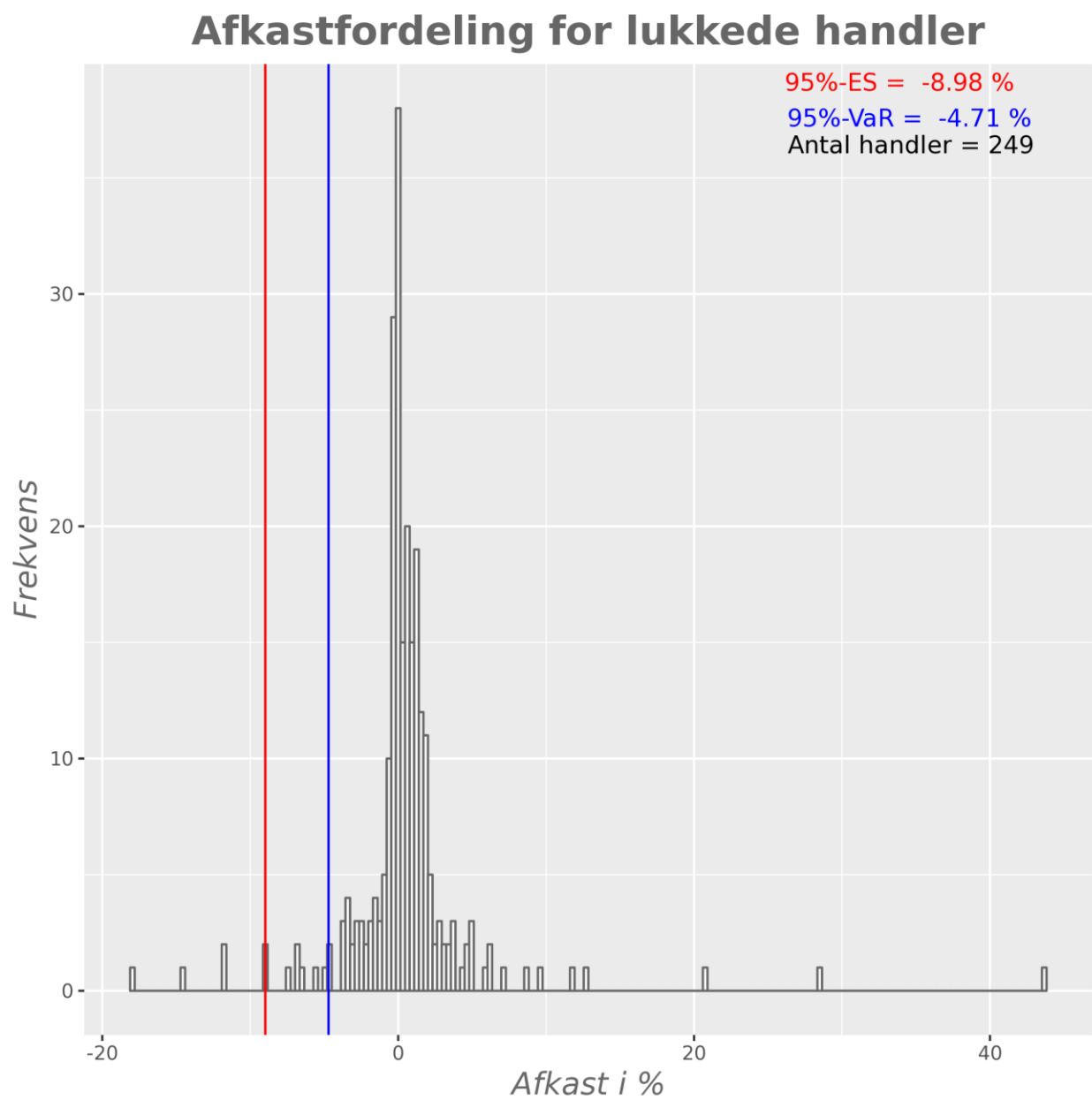
Vores optimale strategi vs. køb og hold strategi



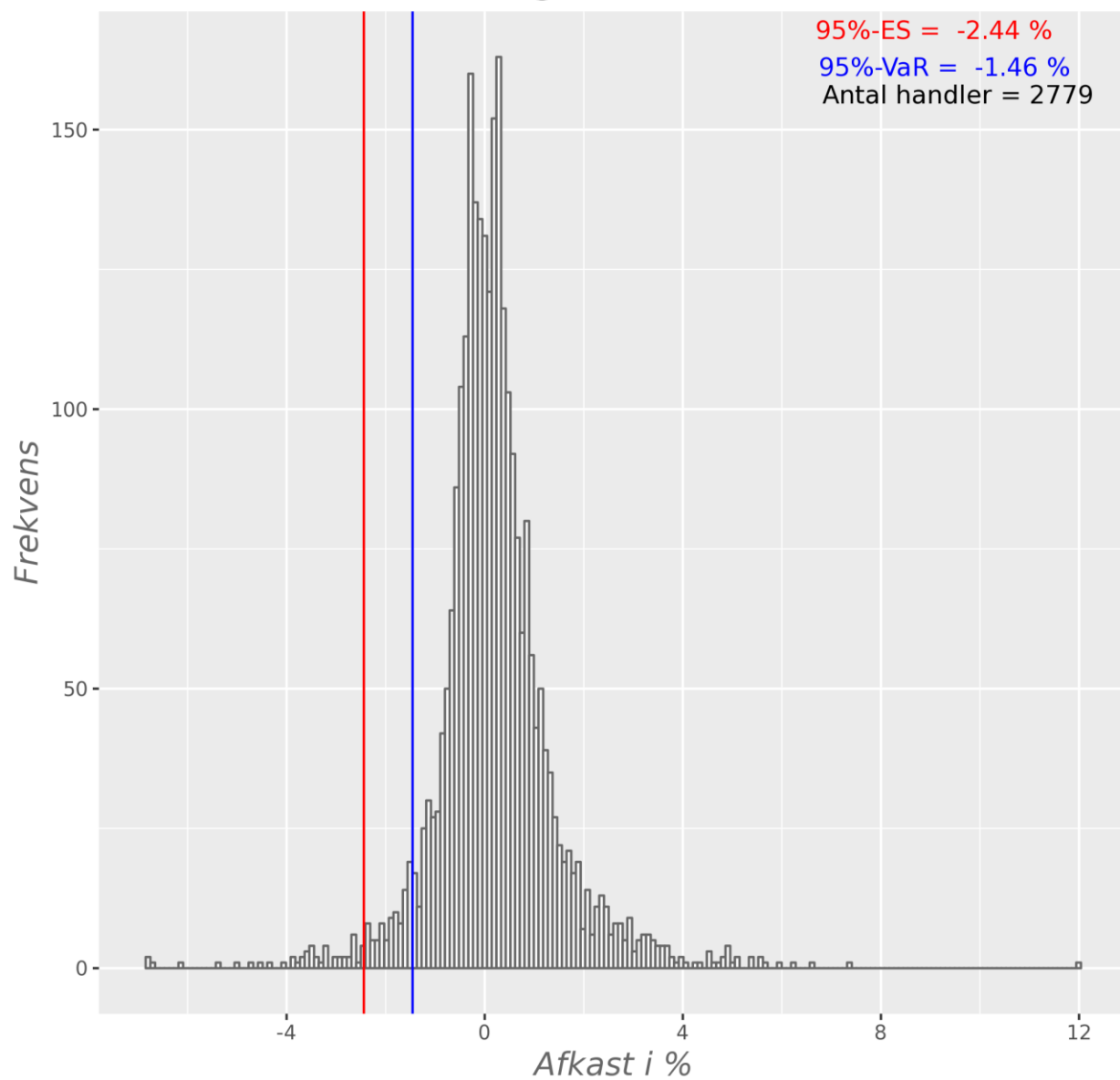


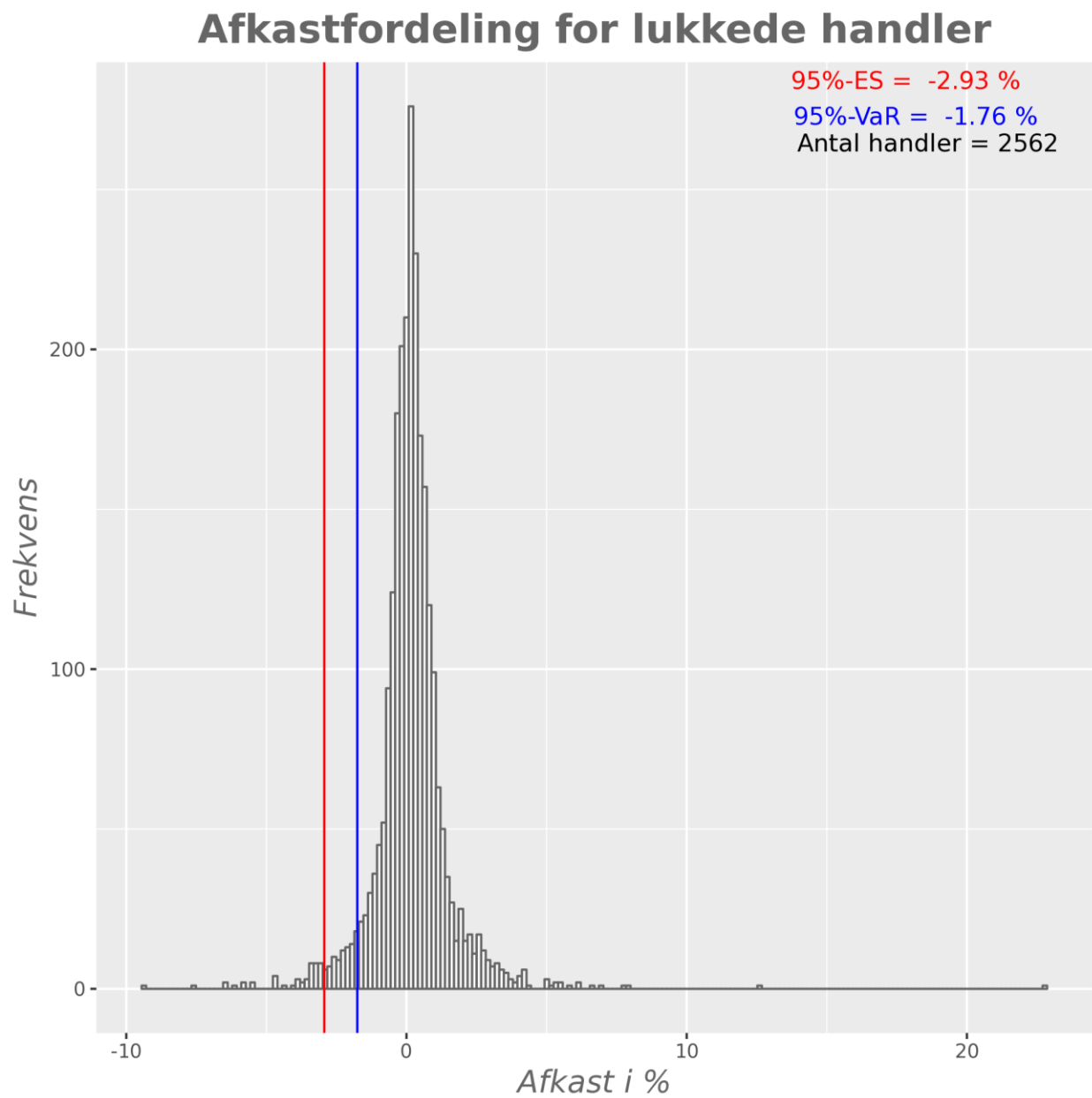
Vores optimale strategi vs. køb og hold strategi





Afkastfordeling for lukkede handler





Afkastfordeling for lukkede handler

