NTNU
Det skapende universitet

Maskinlæring for predikering av live odds

Christian Hunstad
Daniel Reinholdt

Master i datateknologi
Innlevert: juni 2017
Hovedveileder: Trond Aalberg, IDI
Medveileder: Ole Markus With, Sportradar

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknologi og informatikk

Forord

Dette er en masteroppgave utført ved NTNU våren 2017 i samarbeid med bedriften Sportradar. Sportradar startet høsten 2016 utviklingen av en *Data Lake* og ønsker i den anledning å skape ytterligere verdi av data som er tilgjengelig. Vi skal ved hjelp av maskinlæring prøve å forbedre en av Sportradars tilgjengelige tjenester, kalkulering av *Live Odds*. Ved å bruke en del av den store mengden historiske sportsdata Sportradar eier, vil vi i denne oppgaven forsøke å lage maskinlæringsmodeller som skal kunne predikere pågående fotballkamper i et spesifikt tippemarked så nøyaktig som mulig.

Anerkjennelse

Vi vil rette en stor takk til Trond Aalberg ved NTNU for hans veiledning gjennom dette prosjektet. Samtidig vil vi takke Sportradar for muligheten de har gitt oss, samt Ole Markus With og Joakim Dahlstedt for deres tilgjengelighet og effektivitet når enn det ble nødvendig med veiledning i deres systemer.

Oppsummering og konklusjon

Dette prosjektet har hatt som mål å forbedre måten kalkulering av odds gjøres på for et spesifikt tippemarked i fotball. Ved å bruke historisk data samlet inn av Sportradar under fotballkamper, har vi laget maskinlæringsmodeller som forsøker å predikere vinneren av 15-minutts intervaller i fotballkamper i Premier League. 15-minutts intervaller er et tippemarked hvor en kan predikere hjemmeseier, uavgjort eller borteseier i et intervall i en fotballkamp. Med andre ord så deler en opp en fotballkamp i seks mindre kamper. I tillegg til å samle inn statistikker under fotballkamper, tilbyr også Sportradar oddsforslag til tippeselskaper. Ved å utvikle maskinlæringsmodeller som predikerer like nøyaktig eller bedre enn hva Sportradar gjør, vil prosessen med å bestemme odds/sannsynligheter kunne automatiseres i større grad enn hva som gjøres idag.

For å finne frem til de beste maskinlæringsmodellene har vi gjort eksperimenter hvor tre forskjellige algoritmer har blitt brukt til å trene opp modellene; Multinomial Logistic Regresion, Random Forest, og Support Vector Machine. Vi viser til tre eksperimenter innenfor hver algoritme, hvor modellene trenes opp med forskjellige typer attributter som er relevante for en fotballkamp. Resultatene viser at modellene som har blitt trent opp med flest attributter yter best. Av disse modellene er det modellen trent opp med algoritmen Support Vector Machine som scorer best basert på de evalueringsteknikkene tatt i bruk i denne rapporten.

Vi har gjennom tre eksempelkamper vist hvilke sannsynligheter de beste modellene gir for hvert utfall innenfor de forskjellige intervallene, og sammenlignet de korresponderende sannsynligheter Sportradar har utarbeidet. I eksempelet tjener 2 av 3 modeller bedre enn Sportradars sannsynligheter, hvis en tar utgangspunkt i en lik fordeling av innsats på hvert utfall.

Summary and conclusion

The goal of this thesis has been to improve the way calculation of odds in a specific soccer betting market is done. By using historical data collected by Sportradar during soccer matches, we have made several machine learning models with the goal of predicting the outcome of 15 minutes intervals of matches in the Premier League. 15 minutes intervals is a betting market where the bettor can predict home, away or draw within a interval in a soccer match. In other words you split a soccer game into 6 smaller games. In addition to collecting data during soccer matches, Sportradar offers odds in a vast majority of betting markets for bookmakers to use. By developing machine learning models that predicts as accurate or better then what Sportradar does, we hope to improve the process of setting odds and automating it in a bigger way then what is done today.

In order to find the best machine learning models we have conducted experiments using three different machine learning algorithms in the training of our models; Multinomial Logistic Regression, Random Forest, and Support Vector Machine. In the thesis we are showing three experiments within each algorithm, where we are training our models using different number of features. The results show that the models using the most amount of features is performing best. Of all the models trained with the dataset containing the most amount of features, the model trained using Support Vector Machine has the best score using the evaluation metrics outlined in this thesis.

Through a set of example matches, we show how the probabilities the machine learning models are giving compare to the probabilities Sportradar are offering. In this example, two out of three models are out-earning Sportradar given an equal amount of bets on each outcome.

Innhold

Denne siden er med hensikt tom.

Kapittel 1

Introduksjon

1.1 Bakgrunn og motivasjon

Analysering av sportsdata har blitt gjort i flere år med forskjellige mål for øyet. Baseballaget *Oakland Athletics* bygde før 2002 sesongen opp laget sitt ved hjelp av dataanalyser for å finne undervurderte spillere. Laget vant den amerikanske ligaen og seieren er en av de mest berømte triumfene innenfor baseball.

Basketball og baseball er de idrettene som har hatt størst suksess med anvendelse av vitenskapelige teknikker. Den største grunnen til at fotball, som er den idretten dette prosjektet vil fokusere på, ikke er like godt representert innenfor Sportsanalyser er en historisk mangel på offentlig tilgjengelig data [13]. De siste årene har det derimot blitt en økt tilgjengelighet av detaljert data innenfor fotball hos flere aktører. En slik aktør er Sportradar, som vil være kilden til data brukt i dette prosjektet.

Selv med den økte tilgjengeligheten rundt sportsdata er det fortsatt utfordringer når det kommer til å predikere og analysere fotball. Det kan argumenteres for at idretter som baseball er lettere å analysere enn fotball, ettersom hvert *play* i baseball har et startpunkt og sluttpunkt med et fokus på kampen mellom kasteren og slagmann. På grunn av dette kan en bryte ned en baseballkamp mer systematisk enn en fotballkamp. Å vite hva slags statistikker som er utslagsgivende for resultatet i en fotballkamp er en utfordring både fordi statiske analyser er yngre enn

for andre idretter, men også fordi forskjellige lag har forskjellige tilnærminger til kampene. Som et eksempel henger utfallet av fotballkamper ofte ikke sammen med ballinnehav. Underdog vil som regel i en kamp godta at motstanderen triller mye ball, og legge seg lavt og kompakt i banen for så å kjøre kontringsangrep når muligheten byr seg. Cupkamper som i f.eks «FA Cupen» inneholder ofte store overraskelser hvor lag fra lavere divisjoner vinner over noen av de beste lagene i verden.

Å kunne gjøre statistiske analyser og predikere fotballkamper er ikke bare interessant for trenere og lag, men også for tippemarkedet. Tippeselskaper setter sin lit til eksperter for å sette odds på de forskjellige markedene. Rangeringen ekspertene gir lag og spillere er «black-box» i den forstand at eksperter ikke presist kan belyse kriteriene de bruker [13]. Gjennom eksperimentering med forskjellig treningsdata for å lage maskinlæringsmodeller håper vi å kaste lys over hvilke statistikker som har størst korrelasjon til utfallet av en fotballkamp.

I tillegg til den økte tilgjengeligheten av fotballdata de siste årene, har også data blitt mer detaljert. Dette gir muligheten til å ikke bare predikere sluttresultater mer nøyaktig, men også andre tippemarkeder. De fleste kjenner til markeder som går ut på å predikere hvorvidt en kamp ender som hjemmeseier, uavgjort, eller borteseier. Men det finnes haugevis av andre markeder som f.eks resultat innenfor visse tidsintervaller, hvilke spillere som scorer, hva stillingen blir osv.

1.2 Sportradar

Sportradar ble grunnlagt i år 2000, den gang under navnet Market Monitor AS, hvor selskapet leverte verktøy til spillbransjen basert på teknologi utviklet som en del av masteroppgave ved NTNU. Teknologien som ble utviklet overvåket odds fra rundt 300 forskjellige sider, og skapte dermed en tjeneste som tippeselskaper kunne bruke for å verifisere at sine odds var på linje med andre tippeselskaper rundt om i verden [1]. Idag tilbyr Sportradar diverse tjenester til over 800 bedrifter i mer enn 80 land. Utover sine oddstjenester tilbyr de også *live* oppdateringer av sportsarrangementer, avdekking av kampfiksing, og mye mer. Kundene til Sportradar er medie bedrifter, tippeselskaper, idrettsforbund samt statlige myndigheter.

[1] https://www.ercim.eu/publication/Ercim_News/enw61/bazilchuk.html

1.3 *Live Odds*

En av tjenestene Sportradar tilbyr er oddsforslag til tippeselskaper. Kalkulering av *Live Odds* er per dags dato en blanding av automatisert og manuelt arbeid. Algoritmene brukt i den automatiserte delen tar i liten grad hensyn til historisk data, men kalkulerer odds utfra blant annet hvor lang tid det er igjen i kampen, stilling og opprinnelig styrkeforhold. Likevel finnes det mange andre faktorer som påvirker en kamp og det brukes derfor titalls personer (traders) til å observere én kamp. Disse sørger for at *live odds* gjenspeiler hendelser utover det som tas i betraktning i algoritmene. Ved å fokusere på fotball ønsker vi å forbedre denne modellen ved hjelp av maskinlæring. Sportradar gir oss tilgang til deres «Data Lake» som blant annet inneholder kamprefera-ter til flere hundre tusen fotballkamper i XML-form. En ER-modell av skjemadefinisjonene til disse referatene er å finne som vedlegg. Tippemarkedet vi tar utgangspunkt i kan beskrives slik; «hvilket lag scorer flest mål i et gitt 15 minutts intervall», med andre ord deler vi opp en kamp i seks perioder. Dette markedet vil ha tre mulige utfall; hjemmeseier, uavgjort, og borteseier.

1.4 Forskningsspørsmål

Målet for dette prosjektet vil være å utvikle en prediksjonsmodell som klarer å predikere bedre *live odds* innenfor et spesifikt tippemarked en hva som tilbys pr. dags dato. Gjennom arbeidet utført i dette prosjektet ønsker vi å besvare følgende spørsmål:

- Hvordan kan en best mulig sette sammen hendelser fra fotballkamper for å lage et *feature set*?
- Hvordan presterer maskinlæringsalgoritmer når det kommer til predikering av utfall i et spesifikt tippemarked innenfor fotball?
- Er det mulig å forbedre *live odds* i et spesifikt tippemarked kun ved hjelp av maskinlæring?

1.5 Tilnærming

Vi vil fokusere på å predikere kamper i engelsk Premier League. For å lage datasett har vi utviklet programmer i Java som henter og parser XML-filer og JSON-filer fra Sportradars *Data Lake*. Disse programmene kommuniserer også med deres MySQL databaser. Programmene transformerer data som vi ønsker å ha med i trening av maskinlæringsmodeller til CSV-filer. Vi gjør så eksperimenter ved å lage maskinlæringsmodeller som bruker tre forskjellige algoritmer; Multinomial Logistic Regression, Random Forest, og Support Vector Machine. Utviklingen av maskinlæringsmodeller blir gjort ved hjelp av klassebiblioteker i programmeringsspråket R, samt Amazons maskinlæringstjeneste i skyen. Ved å evaluere disse modellene kan vi ta utgangspunkt i de med best resultater og sammenligne sannsynlighetene våre modeller gir med Sportradars *live odds*.

1.6 Struktur

Strukturen til dette prosjektet er delt inn på følgende måte:

Kapittel 2: Maskinlæring gir en innføring i teorien rundt maskinlæring som er relevant for denne masteroppgaven. Vi tar for oss ulike metoder som brukes for å oppnå tilfredsstillende maskinlæringsmodeller.

Kapittel 3: State of the art vil gi en oversikt over tidligere vitenskapelig arbeid som er gjort innenfor samme domene som i dette prosjektet. Den vil også gi eksempler på annet arbeid våre utvalgte algoritmer har blitt brukt til.

Kapittel 4: Treningsdata går nærmere inn på hvordan datasettet blir til, samt *scopet* til prosjektet.

Kapittel 5: Eksperimenter vil gi en oversikt over forskjellige iterasjoner av treningssett, samt evalueringer av maskinlæringsmodeller som blir trent opp med forskjellige algoritmer. De maskinlæringsmodellene som yter best vil videre bli brukt til å sammenligne sannsynligheter med de Sportradar tilbyr i en eksempelkamp.

Kapittel 6: Diskusjon og oppsummering vil gi en konklusjon om hvorvidt maskinlæringsmodeller kan overta det manuelle arbeidet med å sette odds for det spesifikke tippemarkedet. Kapiellet vil besvare forskningsspørsmålene. Vi vil også diskutere styrker og svakheter med funnene våre, og hvordan en eventuell videreutvikling bør struktureres.

Kapittel 2

Maskinlæring

Maskinlæring går inn under grenen kunstig intelligens og går ut på at datamaskiner skal lære fra og utvikle atferd basert på empiriske data. Ved bruk av statistiske modeller, matematisk optimalisering og algoritmer kan maskinen finne komplekse mønstre i et datasett og ta intelligente beslutninger basert på oppdagelsene.

2.1 Forskjellige typer læring

Maskinlæring kan deles inn i tre hovedtyper læring [22]:

- Unsupervised learning
- Reinforcement learning
- Supervised learning

For *Supervised learning* er målet å lære koblingen mellom input og utfall. *Unsupervised learning* gis kun input, og målet er å finne interessante mønstre i datasettet. Dette blir ofte kalt «knowledge discovery». *Reinforcement learning* går ut på å interagere med et dynamisk miljø og lære av observasjoner. Alle handlinger har en effekt på miljøet og læringsalgoritmen blir veiledet basert på tilbakemeldinger i from av belønninger fra miljøet. I praksis er ikke alltid skille mellom disse læringstypene så skarp. Eksempler på dette kan være at deler av datasettet er ukategorisert

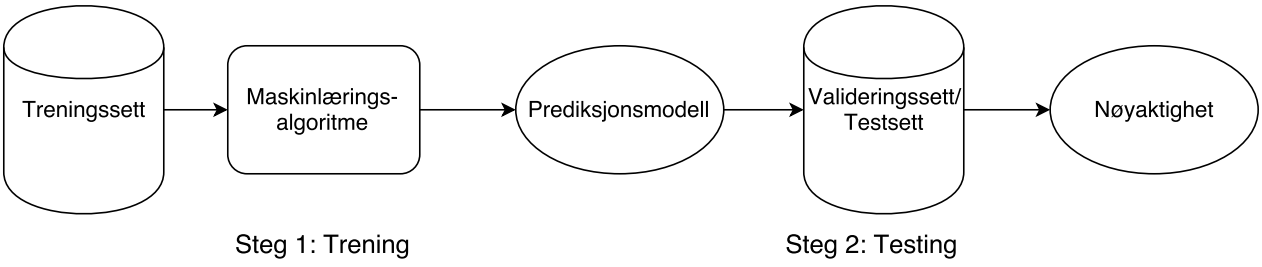
eller feilkategorisert og en blir nødt til å bruke en blanding av *supervised* og *unsupervised* læring (*semi-supervised*). Siden vi ønsker å predikere ved å bruke historisk data til å klassifisere ny data innenfor tre kategorier vil vi i vårt prosjekt fokusere på *supervised learning*.

2.2 Supervised learning

I maskinlæring blir *Supervised learning* også kalt klassifisering eller induktiv læring [14]. Denne typen læring er parallell med hvordan mennesker bruker erfaringer til å tilegne seg ny kunnskap for å forbedre evnen til å utføre oppgaver. Oppgaven med *supervised learning* er: gitt et treningssett med N input-output par

$$(x_1, y_1), (x_2, y_2), ...(x_N, y_N),$$

hvor hver Y_j ble generert av en ukjent funksjon $y = f(x)$, å oppdage en funksjon h som er tilnærmet lik den virkelige funksjonen f [22]. Læringsoppgaven kalles binær klassifisering når y kun har to mulige utfall og *multiclass* klassifisering dersom det er flere utfall. Dersom y er et tall og en ønsker å predikere hvilken verdi y skal ta (f.eks predikere morgendagens temperatur) kalles læringsoppgaven regresjon. Figur 2.1 illustrer de to grunnleggende læringsstegene i *Supervised learning*. I steg 1, en læringsalgoritme bruker treningsdata til å generere en klassifiseringsmodell. I steg 2, den genererte modellen blir testet ved bruk av testdata for å måle nøyaktigheten.



Figur 2.1: Grunnleggende læringssteg: trening og testing [14]

En læringsoppgave innebærer ofte flere iterasjoner av disse læringsstegene før en ender opp

med en modell som har tilfredstillende nøyaktighet. Høy grad av *randomness* i dataene eller begrensninger i læringsalgoritmene kan medføre at modellen aldri oppnår tilfredsstillende nøyaktighet [14].

Et eksempel på *supervised learning* er arbeidet med å klassifisere epost som søppelpost eller legitim epost. Det finnes mange teknikker for å gjøre dette, hvor en av de mest vanlige er å kategorisere det tekstlige innholdet i en epost. I treningen av en modell kan meldingene bli delt opp i ord og brukt som features, sammen med avsender, for så å bli merket som søppelpost eller legitim. Modellen vil kunne gi en sannsynlighet for at en epost er søppelpost eller ikke ved å få det tekstlige innholdet i en epost og avsenders adresse som input. En av de største utfordringene innenfor dette feltet er å bedømme hvilken sannsynlighet som er tilfredsstillende å filtrere med ettersom en aldri ønsker at legitim epost skal bli klassifisert som søppelpost.

Discriminative og Generative

Algoritmer som brukes for å løse klassifiseringsoppgaver faller generelt inn under to kategorier: *discriminative* og *generative*. *Generative* modellerer felles sannsynlighetsfordeling som gir $P(x, y)$, der x er input og y er utfallet. Prediksjoner gjøres ved å bruke Bayes teorem [1] til å kalkulere $P(y|x)$ for så å velge mest sannsynlig utfall. *Discriminative* modellerer betinget sannsynlighetsfordeling som gir $P(y|x)$ og gjør at en kan predikere y gitt x direkte [19]. *Generative*-læring vil gi en rikere modell med kunnskap om hvordan dataene ble generert. Dette gjør at modellen har flere gode egenskaper som f.eks å generere syntetisk data, sette betingede avhengigheter mellom parameterene og håndtere manglende data. *Discriminative*-læring bryr seg ikke om hvordan dataene er generert og er kun opptatt med å predikere et utfall y gitt input x. Siden en *Generative*-modell er rikere er den mer kostbar å modellere og det har blitt demonstrert at *Discriminative*-modell er overlegen fordi den kun fokuserer på den aktuelle oppgaven maskinen trenger å løse [10]. Med mindre en har et spesielt behov for noen av egenskapene en *Generative*-modell tilbyr, så vil det være lønnsomt å bruke en *Discriminative*-modell.

[1] Bayes' teorem er angitt matematisk som følgende ligning $P(x, y) = \frac{P(y|x)P(x)}{P(y)}$, der x og y er hendelser og $P(y) \neq 0$

Datasett

Gitt en situasjon hvor en har mye data tilgjengelig vil den beste fremgangsmåten være å dele datasettet i tre deler [9] : treningssett, valideringssett, og testsett. Treningssettet er dataene som brukes for å trene modellen sånn at den kan oppdage potensielle prediktive relasjoner. Valideringssettet brukes for å estimere prediksjonsfeil for modellvalg og testsettet brukes for å validere den endelige modellen en har valgt. Ideelt bør testsettet kun tas i bruk ved slutten av dataanalysen. Det er ikke en generell regel for hvor mange observasjoner som bør være i hver av delene fordi det avhenger av størrelsen på datasettet og *signal-to-noise* forhold. En typisk splitt kan være 50 prosent til treningsettet, 25 prosent til valideringssettet og 25 prosent til testsett [9]. Måten en deler datasettet kan gjøres på få måter:

- Velg tilfeldige instanser fra datasettet for å sette sammen et treningssett og bruk gjenværende instanser til testing.
- Dersom data er samlet inn over tid, kan den eldste delen brukes til treningssett og den nyere delen brukes til testsett. I mange applikasjoner vil denne fremgangsmåten være best egnet fordi den gjenspeiler hvordan klassifikatoren bruker tidligere hendelser for å kategorisere fremtidige data.

Input-output parene vi mater til modellen vår kan defineres slik $D = (x_i, y_i)$. Her er D treningssettet, x input data, og y utfallet [18]. Hver input x_i representerer i vårt tilfellet en rekke attributter, mens y_i representerer utfallet. Ettersom vi ønsker å predikere utfallet av et intervall i en kamp, altså hjemmeseier, uavgjort, eller borteseier, vil y_i være en kategorisk variabel.

Feature set

Features er variabler eller egenskaper som definerer et problem, et domene eller en verden som skal observeres [15]. En samling av *features* med deres verdier danner en flat datafil som beskriver en applikasjon hvor hver linje beskriver en forekomst eller et mønster. Ved *supervised learning* kan en definere en forekomst som et par $(x, f(x))$, der x er input og definert av en N-dimensjonal *feature* vektor. $f(x)$ gir en av de predefinerte klassene (kategoriene) [15].

Ved å ta utgangspunkt i ovenstående eksempel med klassifisering av epost har vi i tabell 2.1 et forenklet eksempel på *feature* data som kan brukes for å løse oppgaven.

Avsender	Antall ord	Grad av søppelpost-ord	URL	Kategori
Ola@epost.no	104	Høy	Ja	Søppelpost
Kari@epost.no	331	Lav	Nei	legitim post

Tabell 2.1: Eksempel på *feature* data

Her vil "Avsender", "Antall ord", "Grad av søppelpost-ord" og "URL" tilsvare x og $f(x)$ gir "Kategori". *Features* kan ha diskrete eller kontinuerlige verdier, eller være komplekse [15]. Diskrete verdier kan deles inn i ordinale verdier og nominale verdier. Alle *Features* i tabell 2.1 er eksempler på diskrete verdier, hvor "Grad av søppelpost-ord" er ordinal fordi verdiene har en naturlig rangering (høy, medium, lav) mens "URL" er eksempel på en nominal verdi fordi det ikke er en naturlig måte å rangere verdiene på. Kontinuerlige *Features* kan ha verdier fra domenet til de reelle tallene som innebærer at antall mulige verdier er uendelig. Komplekse *Features* kan f.eks være komplekse tall på formen $a + ib$. Features som "Grad av søppelpost-ord" kan igjen være et resultat av andre maskinlæringsmodeller eller statistiske utregninger.

2.2.1 Maskinlæringsalgoritmer

Det finnes et stort antall *supervised learning* algoritmer som kan brukes til å bygge opp prediktive analyseløsninger. En læringsalgoritme består av en «loss» funksjon og en optimaliseringsteknikk, og har som oppgave å lære "vektingen" for modellen. «loss» funksjonen er straffen modellen får når estimatet modellen har regnet ut ikke samsvarer med det faktiske resultatet. En optimaliseringsteknikk forsøker å minimere tapet. De forskjellige algoritmene bruker forskjellige «loss» funksjoner og forskjellige optimaliseringsteknikker. Hver algoritme har sin egen stil eller induktive bias og det er ikke alltid mulig å vite hvilken algoritme som egner seg best for et spesifikt problem. Vi er derfor nødt til å eksperimentere med flere forskjellige algoritmer for å se hvilke som gir tilfredstillende resultater. Som nevnt tidligere ønsker vi å predikere hvilket lag scorer flest mål i et gitt 15 minutts intervall, altså hjemmeseier, uavgjort, eller borteseier. Dette

gjør at vår problemstilling går inn under kategorien *multiclass* klassifisering. Vi kan begrense oss til å eksperimentere med algoritmer som er ment til å løse den typen problemer. Siden vi kun er interessert i å predikere et utfall y, gitt input x vil vi bruke en læringsalgoritme som er *Discriminative*. Algoritmene vi velger å eksperimentere med som går under denne kategorien er:

- Random Forest
- Multinomial Logistic Regression
- Support Vector Machine

Selv om forskjellige algoritmer kan være bedre enn andre med tanke på ytelse, nøyaktighet og kompleksitet er resultatene avhengig av treningssett. Et bedre treningssett kan slå en bedre algoritme og det er derfor viktig at vi finner beste kombinasjon av algoritme og treningsett.

Random Forest

Introduksjonen til Random Forest i den formen den har idag ble gjort i 1998 av Leo Breiman [3]. Random Forest er en ensemble-algoritme som vil si at den bruker flere læringsalgoritmer for å oppnå bedre prediktiv ytelse. Algoritmen gjør dette ved å bygge mange klassifiseringstrær. Treningsalgoritmen til Random Forest bygger på den generelle teknikken «bootstrap aggregating» også kjent som «bagging»:

- Gitt et standard treningsett **D** av størrelse n, bagging genererer m nye treningssett **D**i, hver med størelse n', valgt ut fra datasettet **D** tilfeldig og med bruk av erstatning.

Et sett med trær blir generert av treningssettet **D**i. Treningssettet for hvert tre inneholder kopier av de opprinnelige postene. Tilfeldig utvalg med erstatning sikrer at ca. $n/3$ av postene ikke er inkludert i treningssettet sånn at disse er tilgjengelig som OOB (out-of-bag) data og brukes til å estimere klassifiseringserror og viktighetsgraden til variabler etterhvert som trærne genereres.

I den generelle «bagging» teknikken er settet av attributter som behandles ved en node alle attributtene som ikke er brukt i overordnede noder. I Random Forest er settet med attributter som behandles ved hver node et tilfeldig subset av attributtene, dette er kjent som *feature bagging*.

Prediksjoner på usette poster x' blir gjort ved å aggregere alle trærs prediksjoner, eller regne ut gjennomsnittet hvis det er snakk om regresjon. Dette løser problemet med *overfitting* som ofte er tilfelle med vanlige beslutningstrær. I Random Forest algoritmen er klassifisering avhengig av to ting [4]:

- korrelasjonen mellom trærne; lavere korrelasjon mellom trærne gir mer varianskansellering når trærne skal predikere og gir derfor lavere feilrate.
- Styrken til hvert tre; høyere nøyaktighet på hvert tre gir bedre individuelle predikeringer og derfor lavere feilrate.

Support Vector Machine

Support Vector Machine(SVM) har siden 1990 tallet vært en av de mest populære klassifiseringsalgoritmene og er fortsatt anerkjent som en algoritme med høy ytelse som krever liten tuning. SVM lignende sin nåværende form ble først introdusert i 1992 av Boser, Guyon, og Vapnik [2]. Algoritmen gjør det spesielt bra når en har høy-dimensjonal data og kan optimaliseres for å hindre overtilpassing av treningssettet og dermed oppnå høy nøyaktighet [14]. SVM bruker *hyperplanes* konseptet; i geometrien er *hyperplanes* i et n-dimensjonalt rom V et subrom av dimensjon n-1 [2]. *Hyperplanes* fungerer som en grense mellom kategoriene der postene på hver side av grensen representerer forskjellige kategorier. En SVM-modell representerer alle postene i treningssettet som punkter i et rom, kartlagt slik at eksempler på de forskjellige kategoriene er delt ved et gap som er så bredt som mulig. Nye eksempler blir satt inn i det samme rommet og kategorisert basert på hvilken side av gapet de havner.

Det er to strategier som kan brukes når en benytter SVM til å løse *multi-class classification* oppgaver: *One Versus One* (OVO) og *One Versus All* (OVA) [8]. Gitt en klassifiseringsoppgave med c klasser. OVO oppretter $c(c-1)/2$ binære SVM-klassifikatorer for hvert unike par av klasser. Hver SVM-klassifikator tar *samples* fra en klasse som positiv og *samples* fra en annen klasse som negativ. Klassifisering av en ny post gjøres i henhold til *Max-Wins voting*. OVA oppretter c binære SVM-klassifikatorer og hver klassifiserer en klasse (positiv) mot alle andre klasser (negativ).

[2] Wolfram Math World, Hyperplanes, "http://mathworld.wolfram.com/Hyperplane.html"

Klassifisering av en ny post gjøres i henhold til *Winner-Takes-All*.

I tilfeller hvor data ikke kan separeres lineært brukes SVM med *kernel trick*. Det vil si at input data transformeres fra sitt orginale rom til et annet rom (ofte et rom med mye høyere dimensjon). SVM kan da splitte positive og negative datapunkter lineært i et flerdimensjonalt rom [14].

Multinomial Logistic Regression

Multinomial Logistic Regression er en klassifiseringsmetode som generaliserer logistisk regresjon til å brukes for problemer med mer enn to mulige verdier. Denne generaliseringen skjedde takket være Gurland, Lee, og Dahm (1960), Mantel (1966), og Theil (1969) [23]. Denne formen for logistisk regresjon blir brukt til å predikere kategorisk plassering til en avhengig variabel basert på flere uavhengige variabler. De uavhengige variablene kan enten være dikotome eller kontinuerlige. Som i binær logistisk regresjon, bruker algoritmen «maximum likelihood estimation» for å evaluere sannsynligheten til den kategoriske plasseringen en avhengig variabel har. Optimaliseringsteknikken som blir brukt i anvendelsen av denne algoritmen i dette prosjektet er «stochastic gradient descent»(SGD). SGD gjør sekvensielle passeringer over treningsdata, og oppdaterer vekten av *features* ved hver passering for å minimere tapet som blir regnet ut av den logistiske «loss» funksjonen.

Output til multiclass klassifiseringsalgoritme er et sett med sannsynligheter for hvert mulig utfall. I binær klassifisering brukes «F1-score» for å evaluere hvor nøyaktig modellen kan predikere. Ved å bruke gjennomsnittet av «F1 Score» til de forskjellige klassene kan vi bruke denne målingsmetoden for vår modell [1]. «F1-score» er definert i formel 2.5.

Macro gjennomsnittlig «F1-score» defineres som:

$$Macro \quad average \quad F1 \quad Score = \frac{1}{K} \sum_{k=0}^{k} F1 \quad Score \quad for \quad class \quad K \tag{2.1}$$

2.2.2 Evalueringsmetoder

For å vite hvor godt en klassifiseringsmodell presterer er en nødt til å måle nøyaktigheten. Det er flere fremgangsmåter som kan benyttes for å måle hvor nøyaktig og robust en modell er. Før vi forklarer fremgangsmåtene vil vi introdusere begreper som brukes for å presenterer resultatene av en klassifikator:

- True Positive (TP): antall korrekte klassifiseringer av et positivt eksempel
- False Negative (FN): antall feilklassifiseringer av et positivt eksempel
- False Positive (FP): antall feilklassifiseringer av et negativt eksempel.
- True Negative (TN): antall korrekte klassifiseringer av et negativt eksempel

Disse begrepene brukes i en *confusion matrix* (Tabell 2.2) og med den kan en visualisere prestasjonen til en algoritme.

Alle Eksempler	Klassifisert positiv	klassifisert negativ
Positive eksempler	TP	FN
Negative eksempler	FP	TN

Tabell 2.2: confusion matrix

Accuracy

Nøyaktighet kan måles ved å dele alle riktige klassifiserte eksempler med totalt antall eksempler:

$$\text{Accuracy} = \frac{\sum TP + \sum TN}{\text{Alle Eksempler}} \tag{2.2}$$

Det er også vanlig å bruke *error rate* som evalueringsmetode. Denne er definert som $1 - Accuracy$. Det finnes tilfeller hvor *Accuracy* og *error rate* ikke gir et pålitelig resultat f.eks når størrelsen på datasettet er veldig lite eller når en har et veldig ubalansert datasett. I disse tilfellene er en nødt til å bruke andre evalueringsmetoder.

Cross-validation

I tilfelle hvor størrelsen på datasettet er lite kan en bruke *k-fold cross-validation* eller *leave-one-out validation*. *k-fold cross-validation* går ut på å dele datasettet opp i k disjunkte subsett med lik størrelse. Hvert av subsettene brukes hver sin gang som testsett og de gjenværende $k-1$ subsettene settes sammen og brukes som treningssett. Nøyaktigheten (formel 2.2) regnes ut for hvert testsett og gjennomsnittet av disse k nøyaktighetene vil være det gjeldene estimatet. Populære verdier for k er 5 og 10. Disse verdiene for k er statistisk sannsynlig å gi et estimat som er nøyaktig. Det er viktig å merke seg at beregningstiden øker med antall subsett k, hvor eksempelvis $k = 5$ vil gi 5 ganger lenger beregningstid.

Ved bruk av *leave-one-out validation* brukes kun ett av datapostene til testing og resten brukes som treningssett. Dersom datasettet er på størrelse m vil prosessen tilsvare *m-fold cross-validation* og denne fremgangsmåten brukes kun hvis en har veldig lite data tilgjengelig.

Precision

Et ubalansert datasett vil si at en klasse er veldig underrepresentert. I disse tilfellene er en ofte spesielt interessert i å predikere hvilke dataposter som tilhører den underrepresenterte klassen. Bruk av formel 2.2 vil i de tilfellene være et upålitelig estimat fordi den ikke gir en god indikasjon på om modellen fungerer som ønsket. En kan oppnå høy nøyaktighet uten at modellen har predikert datapostene fra den underrepresenterte klassen riktig. *Precision* vil i dette tilfelle være et bedre estimat fordi den måler hvor godt modellen predikerer en gitt klasse. *Precision* er antall riktige klassifiseringer av en gitt klasse delt på totalt antall instanser klassifisert til den gitte klassen, og uttrykkes [14]:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.3}$$

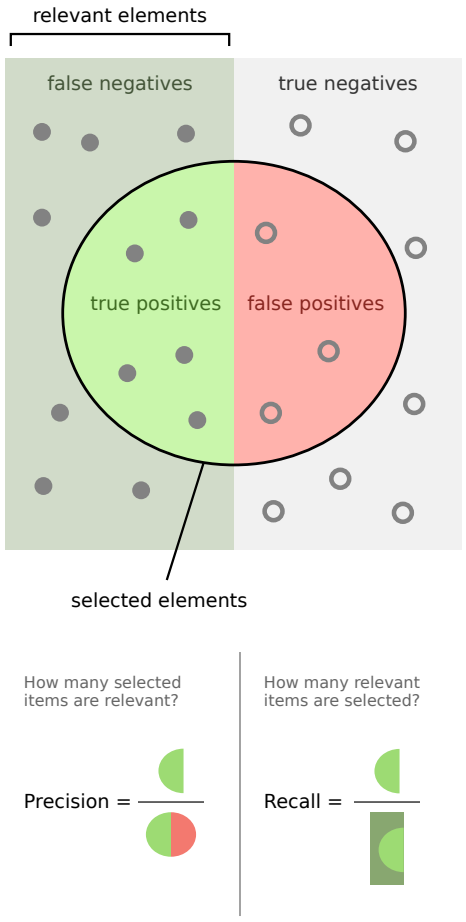
Precision er visualisert i figur 2.2.

Recall

Recall er antall riktige klassifiseringer av en gitt klasse delt på totalt antall instanser av den gitte klassen, og uttrykkes [14]:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.4}$$

Recall er visualisert i figur 2.2.



Figur 2.2: Precision og Recall visualisert. Bilde er hentet fra: https://en.wikipedia.org/wiki/Precision_and_recall

.

F-score

Precision og *Recall* er to gode måter å måle hvor godt en modell presterer med hensyn til individuelle klasser. Likevel er det vanskelig å bruke disse to målingene når en skal sammenligne to klassifikatorer. Dette kommer av at de to målingene ikke er funksjonelt relaterte. En modell kan ha høy *Precision* og lav *Recall* eller omvendt. Dersom en ønsker å bruke disse målingene til å sammenligne klassifikatorer er det vanlig å bruke F-score (også kalt F1-score) som er det harmoniske gjennomsnittet [3] av *Precision* og *Recall*. F1-score defineres som [14]:

$$F1score = \frac{2 * precision * recall}{precision + recall} \tag{2.5}$$

2.2.3 Feature engineering

Feature Engineering er en viktig prosess av det å bygge prediktive analyseløsninger. Dette er fordi prosessen skal hjelpe til med å finne relevante *features* som bør brukes i en maskinlæringsalgoritme. Med teknologiske fremskritt blir det enklere og billigere å samle store mengder data. Dette gjør at en fort kan ende opp med store *feature set*. For hver *feature* som brukes i en maskinlæringsalgoritme øker beregningstid og kompleksitet. Det er derfor viktig å finne en balanse mellom kompleksitet og fordelene med den endelige nøyaktigheten til en modell. *Feature Engineering* kan deles opp i to prosesser [21]:

- Feature selection
- Feature validering

Der den førstnevnte er statistikkintensiv og gir empirisk bevis på hvorfor en viss *feature* eller et sett med *features* er viktig for maskinlæringsalgoritmen. Den andre handler mer om å bruke domenekunnskapene en har for å sikre at *features* som brukes gir mening og gir riktig innsikt f.eks med tanke på hva en bedrift er ut etter.

[3] Det harmoniske gjennomsnittet av to tall er definert ved at forholdet mellom det største tallet minus gjennomsnittet og gjennomsnittet minus det minste tallet, skal være lik forholdet mellom det største og minste tallet. https://no.wikipedia.org/wiki/Harmonisk_gjennomsnitt

Feature selection

Målet med *Feature selection* er å identifisere hvilke *features* i et datasett som er korrelert med eller kan predikere en gitt klasse. Disse blir satt sammen til et subsett av *features* som kan brukes til å trene en modell. *Feature selection* kan deles inn i tre grupper [21]:

- Filter: metoden går ut på å rangere *features* basert på deres naturlige egenskaper. Dette kan f.eks være korrelasjon eller varians. Metoden er generelle og finner ”beste” *features* uavhengig av maskinlæringsmetoden en ønsker å bruke.
- Wrapper: metoden går ut på å teste flere forskjellige kombinasjoner med *features* for å finne det beste subsettet. Kombinasjoner av *features* testes med en maskinlæringsmodell, også referert til som prediksjonsmodell, og metoden tar derfor hensyn til maskinlæringsalgoritmen en ønsker å bruke når den tester subsettene. Evalueringsmetoder blir tatt i bruk for å regne nøyaktigheten og subsettet med høyest nøyaktighet blir brukt til modellering. Metoden kan være beregningsmessig dyr og har risiko for *overfitting* av modellen.
- Embedded: metoden er en videreutvikling av wrapper metoden. Metoden introduserer en straffefaktor til evalueringskriteriene for modellen. Dette gjøres for å styre modellen mot lavere kompleksitet. Algoritmene prøver å balansere mellom kompleksiteten og nøyaktigheten av modellen.

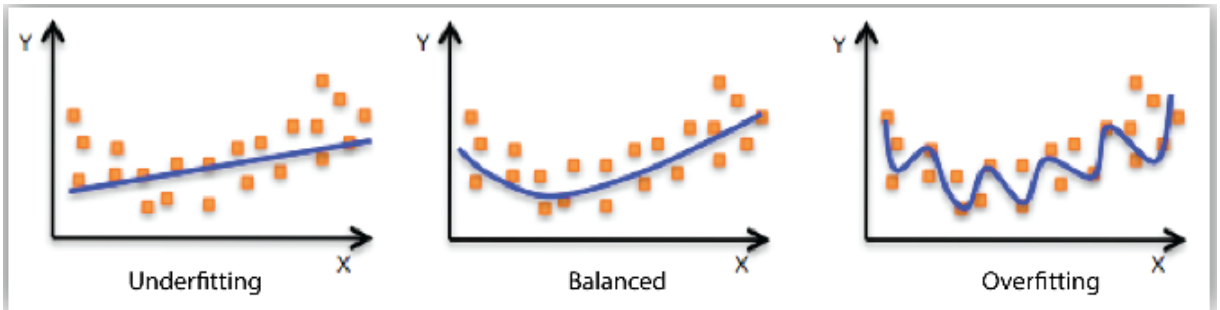
2.2.4 Problemområder med klassifisering

Bias og varians

Når en prediksjonsmodell har høy varians så representerer den treningssettet godt, men har en tendens til å overtilpasse seg støy og underrepresenterte data i treningssettet [21]. Dette medfører at den representerer testdata dårlig. Prediksjonsmodeller med høy bias er ofte enklere og overtilpasser seg ikke treningssettet, men har en tendens til å ikke fange opp viktige regelmessigheter i treningssettet og ender derfor opp med å være undertilpasset [21]. En modell med lav bias må derfor være ”fleksibel” slik at den kan tilpasse data godt. Men hvis modellen er for flek-

sibel, vil den tilpasse hvert treningsdatasett annerledes, og dermed ha høy varians. Lav bias eller lav varians oppnås ofte på bekostning av hverandre og et vanlig problem er å finne en balanse mellom disse.

Figur 2.3 visualiserer hvordan en typisk undertilpasset, balansert, og overtilpasset modell vil se ut.



Figur 2.3: Tre forskjellige tilpasninger av modell. Bilde er hentet fra: http://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

Kompleksitet og datamengde

Prestasjonen til en prediksjonsmodell er avhengig av kompleksiteten til den virkelige funksjonen f for klassifiseringsproblemet og hvor mye data en har tilgjengelig. Hvis f er enkel kan en læringsalgoritme med høy bias og lav varians kunne lære det fra en liten mengde treningsdata. Er f kompleks vil det mest sannsynlig være bruk for veldig stor mengde treningsdata og en læringalgoritme med lav bias og høy varians.

Støy

Når en skal lage en prediksjonsmodell så finnes det situasjoner hvor dataverdiene ikke kan modelleres. Det kan være at det er målefeil eller tilfeldige svingninger i dataene, dette kalles stokastisk støy. En annen situasjon er at læringsproblemet er for komplekst og en har for høy kompleksitet i dataene som skal modelleres, dette kalles deterministisk støy. Stokastisk og deterministisk støy fører ofte til at modellen blir overtilpasset.

Kapittel 3

State of the art

3.1 Tidligere arbeid

Maskinlæring brukes til utallige ting i dagens samfunn. Noen eksempler på dette kan være:

Web søk tar i stor grad bruk maskinlæring. Google har redifinert seg selv som et maskinlæringsselskap, og søkeforslag samt søkeresultater i deres søkemoterer er i stor grad et resultat av maskinlæring. Et eksempel på dette er når google spør deg om du egentlig mente å søke på noe annet(«Did you mean ..»). Dette er blant annet et resultat av maskinlæringsalgoritmer som tar utgangspunkt i søkeord som er gjort med få sekunders mellomrom.

Anbefaling av produkter er et veldig populært område å bruke maskinlæring. Netflix tar for eksempel bruk en rekke algoritmer innenfor både *supervised* og *unsupervised* læring. Historisk sett baserte Netflix seg på kundenes anmeldelser av filmer for å gi anbefalinger. Men på grunn av de store mengdene data som de nå sitter på kan de gjøre anbefalinger ved å bruke informasjon om når et medlem ser på filmer/serier, hvor på skjermen denne personen trykket, hvilke anbefalinger medlemmet ikke tok til seg osv.

Språkoversetting blir bare bedre og bedre takket være maskinlæring. Ved hjelp av den økende mengden tekster som er tilgjengelig, lærer maskinlæringsalgoritmene seg hvordan ord henger sammen og oversetter mer nøyaktig enn tidligere tjenester.

Dette er bare noen få eksempler på bruksområder til maskinlæring. Videre vil vi ta et dypere dykk innenfor arbeidet som er gjort innenfor maskinlæring i fotball, før vi kikker på noen anvendelser av algoritmene som vi kommer til å bruke i våre eksperimenter.

3.2 Domenerelatert arbeid

I «Applying machine learning to event data in soccer» [12] forsøker forfatteren å predikere utfallet av fotballkamper ved bruk av hendelser i kamper (eksklusiv mål). I tillegg forsøker han å klassifisere hvilke lag en sekvens av gitte hendelser (pasninger, skudd, taklinger etc.) tilhører, med andre ord for å se om lag har spillestil som kan gjenkjennes av statistikker. Datasetet som blir brukt er samlet inn av Opta [1] , og er mer detaljert enn hva som er tilgjengelig i Sportradars kampreferater. For predikeringen av utfallet i kampene ble det brukt tre forskjellige modeller som tok i bruk tre forskjellige *feature sets*. Disse settene deles inn i åpenbare *features*, ikke-åpenbare *features*, samt en blanding av de to. Åpenbare *features* er f.eks skudd på mål og hvilke lag som spiller, mens to eksempler på ikke-åpenbare *features* er forskjellen av antall innlegg mellom to lag og forskjellen på den gjennomsnittlige skuddavstanden mellom to lag. Avhandlingen fungerer som en bra grobunn for fremtidig forskning ettersom de blant annet viser hvilke fem *features* som er viktigst i hver modell.

«Predicting outcome of soccer matches using machine learning» [24] gir også et innblikk i hvilke *features* som kan prøves ut i eksperimenteringen av forskjellige modeller. Som et eksempel bruker de en formel for å regne ut målforskjell kontra det å ha med stilling som to forskjellige *features* (mål for hjemmelag og bortelag). Form, konsentrasjon, motivasjon og historie er også *features* de har forsøkt å bruke. Noen av disse er interessante, selv om vi ikke er overbevist om at bruken av motivasjon og konsentrasjon vil være features som kan regnes ut ved hjelp av en formel.

I «Machine learning for soccer analytics» [13] var et av målene å finne en god måte å aggregere *rating* for fotballag og undersøke hvor nært disse *ratingene* kan komme til å predikere utfallet av en kamp. Å klassifisere utfallet av en kamp ved å bruke lagets *rating* fra samme kamp resulterte i

[1] http://www.optasports.com/

en nøyaktighet på 90 prosent. Jo flere historiske *ratings* fra kamper som ble tatt med, jo mer sank nøyaktigheten. Dette er ikke interessant på et annet nivå enn å verifisere at *ratingen* som blir gitt til lagene er korrekte. Men når de forsøkte å predikere fremtidige kamper ved bruk av lagets attributter oppnådde de kun en nøyaktighet på 54,4 prosent. Dette var beste resultatet etter utviklingen av to modeller, hvor den ene modellen hadde 8 *features* mens den andre modellen hadde 27 *features*, og data fra de 7 siste kampene ble brukt. De konkluderer med at grunnen til de dårligere resultatene er vanskeligheten med å predikere uavgjort.

Det som går igjen i tidligere arbeid er at det er vanskelig å predikere fremtidige kamper, men å klassifisere et utfall basert på kampens attributter er ofte vellykket og viser at attributtene er betydningsfulle. Vi skal i vårt prosjekt predikere kamper i 15-minutts intervaller, et tippemarked tilbudt av Sportradar. I prediksjon av fullstendige kamper er det uavgjort som er mest vanskelig å predikere, mens i vårt tilfelle vil uavgjort være det som forekommer oftest.

3.3 Algoritmerelatert arbeid

Videre har vi kikket på forskningsartikler med varierende tematikk, hvor algoritmene som brukes i våre eksperimenter er anvendt, for å vise hvor bredt bruksområde maskinlæring har. Selv om disse forskningsartiklene har et stort temasprang, kan de relateres til vårt arbeid. Fremgangsmåten som blir brukt i utviklingen av *feature sets* i alle artiklene tilsvarer tilnærmingen vi har brukt i vårt prosjekt. I artikkel 3.3.1 brukes *multiclass*-klassifisering hvor de er interessert i sannsynlighetsfordelingen mellom hvert av utfallene. I artikkel 3.3.2 forsøker de å forbedre eksisterende løsninger ved å gå mer spesifikt inn i et domenet, hvor de ønsker å predikere basert på nylig generert data. I artikkel 3.3.3 sammenligner de algoritmer i utviklingen av prediksjonsmodeller ved å bruke ikke-lineær data.

3.3.1 Identifisering av faktorer assosiert med anemi hos barn fra 6-59 måneder i nordøstlige stater i India [6]

I denne forskningsartikkelen kikker forfatterene på hvilke faktorer som har størst betydning for utvikling av forskjellige grader av anemi hos barn i de nordøstlige statene i India. Ifølge WHO har India den største utbredredelsen av anemi i alle aldersgrupper blant land i sør-Asia. 70 prosent av barn mellom 6 og 59 måneder i India har en grad av anemi.

Forfatterene tar i bruk Multinomial Logstic Regression til å kartlegge hvilke faktorer som har størst betydning for de ulike gradene av anemi. Graden av anemi blir delt inn i fire kategorier; alvorlig anemi, moderat anemi, mild anemi, og ingen anemi.

De lager fire forskjellige modeller hvor de for hver iterasjon legger til flere faktorer som kan spille inn. Dette er en fremgangsmåte vi også følger i vårt prosjekt. De forskjellige faktorene som blir tatt med i modellene er følgende:

- Mors karakteristikker
  - Antall barn født
  - Alder ved giftemål
  - Religion
  - Leseferdighet
- Alder på barn
- Bosted
- Levestandard til husstand
- Barnets kjønn

Resultatene fra analysene viser at utbredelsen av anemi varierer signifikant mellom statene, samt hvilken religion moren har. Det er også en signifikant assosiasjon mellom graden av anemi til et barn og alderen moren hadde da hun ble gift. Det viser seg interessant nok at utbredelsen av anemi er lav blant barn av mødre som ikke kan lese eller skrive, selv om assosiasjonen ikke er

statistisk betydelig. Det var ikke funnet statistisk betydelig assosiasjon mellom graden av anemi og husstandens levestandard, kjønn, leseferdigheter, antall barn født, samt alder til barnet.

På bakgrunn av disse funnene og modellene som er utarbeidet kan de regne ut sannsynligheten for å ha forskjellige grader av anemi. Et av forskningsspørsmålene var å finne ut av sannsynligheten for å være på et nivå eller høyere av anemi for barn med typisk bakgrunn. Resultatene viser eksempelvis at sannsynligheten for at et barn med gjennomsnittlig bakgrunnskarakteristikker har alvorlig grad av anemi er 12,47 prosent.

3.3.2 Support Vector Machine tilnærming for å gi en prognose om oversvømmelse under tyfoner [11]

Ved tyfoner så er nøyaktige prognoser av vannstand essensielle for advarsler mot oversvømmelser og eventuelle evakueringer. I denne artikkelen forsøker de å bruke SVM til å kartlegge oversvømmelser en til seks timer før de skjer.

Forskningsområde er lagt til Chiayi i Taiwan. Denne byen har en ujevn fordeling av nedbør og er utsatt for oversvømmelser. I denne artikkelen er en vannstand i et område over 0.3 M å regne som oversvømt.

Det blir laget to modeller som tar i bruk forskjellige input. Den første modellen tar i bruk foregående intensitet på nedbør samt høyde på vannstand, mens den andre modellen legger til den kumulative nedbøren og meldte vannstanden. På samme måte som i foregående artikkel blir modellene utviklet iterativt, hvor *features* blir lagt til foregående modell.

Resultatene viser at *features* de har valgt å ta med forbedrer tidligere prognoser, og at tilnærmingen klarer å predikere nøyaktige prognoser med 1 til 6 timers forvarsel.

3.3.3 Predikering av organisk karbon i jord ved en intensivt håndtert gjenvinningssone i øst-Kina [25]

For å mitigere global oppvarming bør jord håndteres til å beslaglegge mer organisk karbon og utgi mindre drivhusgasser. I denne forskningsartikkelen forsøker de å predikere SOC (Soil organic carbon) ved å bruke Random Forest og Multiple linear regression modeller. De vil også identifisere hvilke faktorer som har størst påvirkning på SOC i en intensivt håndtert gjenvinningssone.

I resultatene av predikeringene viste det seg at Random Forest utklasset Multiple linear regression. Forfatterene konkluderer med at dette sannsynligvis er fordi Random Forest er overlegen i å håndtere ikke-lineær data og hierarkiske forhold.

3.4 Oppsummering

Gjennomgangen av tidligere arbeid har vist at algoritmene som vi tar i bruk har et bredt bruksområde. De fleste forskningsartiklene har en metodologi hvor de utvikler modeller med få *features* til å begynne med for så å legge til flere og flere. Dette er en fremgangsmåte også vi vil bruke i våre eksperimenter.

De sportsspesifikke artiklene vi har sett på har gitt oss en god oversikt over hvilke type *features* som er hensiktsmessige å ta med, selv om våre eksperimenter vil belyse andre aspekter innenfor predikering av fotball enn hva som er gjort tidligere på grunn av det spesielle tippemarkedet vi fokuserer på.

Når en skal utvikle maskinlæringsmodeller har en mange muligheter hva gjelder teknologi. Vi vil videre kikke på de mest populære løsningene.

3.5 Teknologi

De siste ti årene har man sett en økning av maskinlæringsbiblioteker og tjenester som gjør det mer tilgjengelig å sette seg inn maskinlæring. Dette gjør at vi kan få innsikt mer effektivt og raskere lage applikasjoner som tar i bruke maskinlæring på egen maskinvare eller skybaserte tjenester. Utviklingen av maskinlæringsmodeller i skyen gir økt tilgang til datakraft, god visualisering av dataene og forslag til innstillinger som kan forbedre prediksjonsmodellen. Skybaserte tjenester har dog et begrenset antall med maskinlæringsalgoritmer og vi ønsker derfor å eksperimentere med avansert analyse av data ved bruk av skytjeneste og maskinlæringsbiblioteker.

3.5.1 Skyløsninger

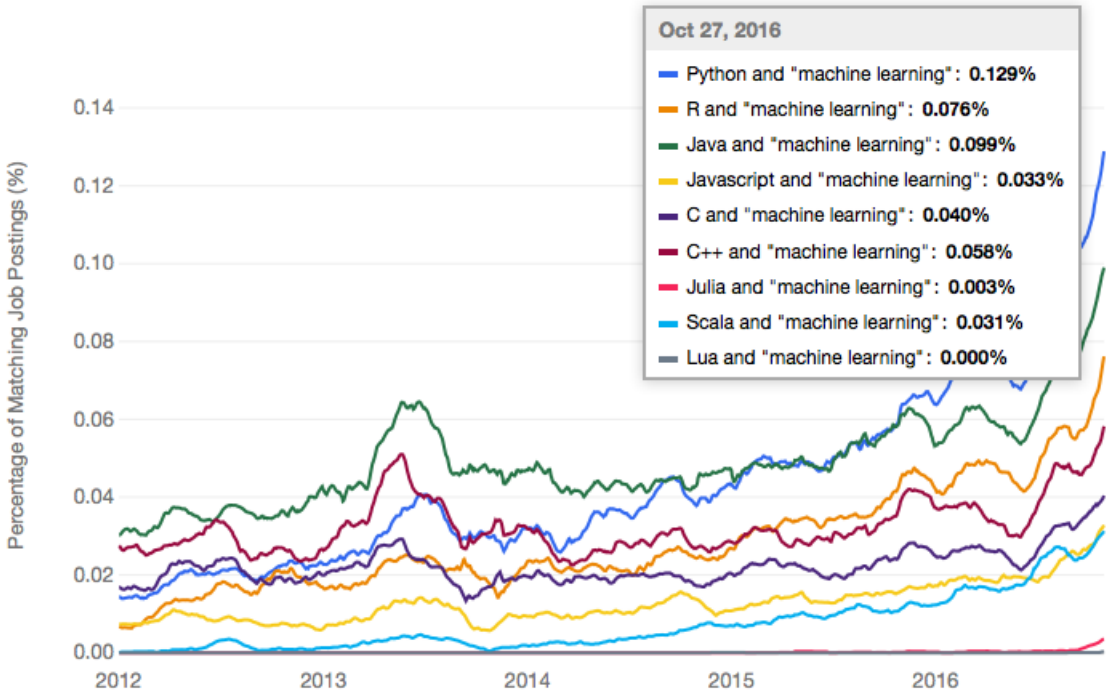
Leverandører som Google Cloud, Microsoft Azure, og Amazon Web Services(AWS) leverer alle maskinlæringstjenester i skyen. Google tilbyr for eksempel både forhåndstrente modeller for trivielle problemer, såvel som muligheten til å skreddersy egne maskinlæringsmodeller ved bruk av deres «neural-net» baserte maskinlæringstjeneste. Alle de forskjellige tjenestene har en rekke opplæringseksempler som baserer seg på typiske problemer maskinlæring kan løse.

Microsoft Azure har en rekke forskjellige algoritmer som de tilbyr å trene opp modeller med, og har på lik linje med AWS et godt visuelt grensesnitt som gjør treningen av modeller lettere. Disse tjenestene har også API'er for en rekke programmeringsspråk slik at modellene kan bli brukt til predikering i applikasjoner.

Vi vil i våre eksperimenter bruke AWS for eksperimentering med algoritmen Multinomial Logistic Regression. AWS tilbyr også utvikling av modeller med vanlig logistisk regresjon samt lineær regresjon. For eksperimentering med Random Forest og SVM vil vi programmere modeller ved bruk av biblioteker.

3.5.2 Programmeringsspråk

Det finnes mange programmeringsspråk som egner seg godt til å bygge prediktive analyseløsninger. Jean-Francois Puget fra IBM fant en oversikt over hvilke ferdigheter arbeidsgivere søker og lagde en avansert indikator (figur 3.1) på hvordan ferdighetenes popularitet utvikler seg med tiden [20].



Figur 3.1: Det mest populære språket for maskinlæring og utviklingen over tid [20]

I figur 3.1 kan vi se at Python er den mest etterspurte ferdigheten etterfulgt av Java og deretter R. Den viser at Java sin popularitet over R minsker og at Python sin popularitet over Java øker. KDNuggets er et av de ledende nettstedene innenfor bl.a. maskinlæring, datautvinning og big data. De gjennomførte en spørreundersøkelse i 2015 og resultatene [2] viste at de mest brukte programmeringsspråk innenfor analyse, datautvinning, datavitenskap-oppgaver var R etterfulgt av Python. Vi valgte derfor å se nærmere på Python og R.

[2] http://www.kdnuggets.com/polls/2015/r-vs-python.html

3.5.3 Python

Python er kjent for å kunne brukes til mange generelle formål og det har blitt mer og mer populært å bruke innenfor maksinlæring. Det er et «open-source software» som vektlegger produktivitet og lesbarhet. Det er derfor en enkel syntakst og oppgaver kan løses med få linjekoder. Det har et stort «community», men det er litt spredd fordi Python kan brukes til mange forskjellige formål. Det finnes flere biblioteker som muliggjør statistiske utregninger og som gir gode visualiseringer. Eksempler på populære maskinlærings-bibliotek er: Tensorflow, scikit-learn, Theano og Pylearn2.

3.5.4 R

Et åpent og gratis programvare-miljø for statistisk og grafisk utregning kalt R kan brukes til å utvikle applikasjoner som tar i bruk maskinlæring. Fordelen med å bruke R er blant annet at det har et rikt økosystem med pakker og brukere. Via CRAN-Repository [3] får en tilgang til mange maskinlæringsalgoritmer, og avanserte implementeringer som gjør at en raskt kan komme i gang med utviklingen. Visualiserte data kan ofte bli forstått mer effektivt enn rå tall alene. Det sies at R og visualisering er en perfekt match.

3.5.5 Vårt valg

Vi hadde et ønske om å utforske skyløsninger. Disse tjenestene er ikke gratis, og ettersom vi har fått tilgang til Sportradar sin Amazon konto valgte vi å utnytte dette. Denne maskinlæringstjenesten har bare en av de tre algoritmene vi ønsker å bruke i vårt prosjekt, Multinomial Logistic Regression. Etter eksperimentering i både R og Python besluttet vi å bruke R til å utvikle med algortimene SVM og Random Forest. Både R og Python var gode alternativer, men vi endte opp med R sine biblioteker fordi de var mer egnet til å jobbe med kategoriske variabler. Når en jobber med kategoriske variabler i Python må en bruke «One-hot-encoding». Dette går ut på å splitte kategoriske variabler med n mulige variabler til n binære kolonner [4]. Dette kan skape proble-

[3] https://cran.r-project.org/web/views/MachineLearning.html
[4] http://www.kdnuggets.com/2015/12/beyond-one-hot-exploration-categorical-variables.html

mer ved bruk av Random Forest fordi over halvparten av variablene som brukes til å bygge opp klassifiseringstrær er *encoding* av samme informasjon. Ikke-kategoriske variabler vil mest sannsynlig få høyere viktighetsgrad enn kategoriske variabler og kategoriske variabler vil sjeldent bli brukt til splitting høyt oppe i klassifiseringstrærne. Dette kan medføre at vi får en dårligere prediksjonsmodell [7].

3.6 Kalkulering av *Live Odds* i fotball

Ettersom vi skal forsøke å forbedre sannsynlighetene ved forskjellige utfall i fotballkamper kan det være hensiktsmessig å se på hvilke metoder som kan brukes idag. Et eksempel er bruken av en Poisson-fordeling, kombinert med målhistorikk, for å beregne sannsynligheten for at et lag scorer x antall mål i en fotballkamp. Ved å kombinere sannsynlighetene kan det regnes ut en odds for utfallene: hjemmeseier, borteseier og uavgjort. For å kunne bruke en Poisson-fordeling er vi nødt til å finne hvor mange scoringer som er forventet av hjemmelaget og bortelaget. For å finne forventet scoringer av hjemmelaget trenger vi å finne hjemmelagets angrepsstyrke. Angrepsstyrken er definert som:

$$\text{Home team attack strength} = \frac{\text{home team average home goals}}{\text{Average home goals in a given league}} \tag{3.1}$$

Videre må vi finne bortelagets forsvarsstyrke. Forsvarsstyrke er definert som:

$$\text{Away team defence strength} = \frac{\text{Away team average conceded away goals}}{\text{Average home goals in a given league}} \tag{3.2}$$

Etter å ha regnet ut Angrepsstyrken til hjemmelaget og forsvarsstyrken til bortelaget kan vi finne forventet antall scoringer av hjemmelaget. Forventet antall scoringer er definert som:

$$\text{Expected score} = \text{attack strength} * \text{defence strength} * \text{Average home goals in a given league} \tag{3.3}$$

Samme fremgangsmåten kan brukes for å finne forventet antall scoringer til bortelaget, men vi bytter da ut scoringer på hjemmebane med scoringer på bortebane, baklengsmål på bortebane med baklengsmål på hjemmebane og gjennomsnittlig hjemmescoring i en liga med gjennomsnittlig bortescoring i en liga. Når vi har funnet *expected home score* og *expected away score* kan disse brukes i Poisson-fordeling:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \tag{3.4}$$

hvor

- λ er forventet antall scoringer
- x er antall mål

Etterhvert som tiden går i en fotballkamp vil forventet antall scoringer minske. Gjenværende forventning av mål kan regnes ut [5]:

$$\text{Remaining goal Expectation} = \text{Initial expectation} * \text{Proportion of time remaining} \wedge 0{,}85 \tag{3.5}$$

Vi har i vårt prosjekt fått direkte tilgang på odds utarbeidet av Sportradar. Uten disse resurssene er dette en måte å kalkulere omtrentlige odds i fotballkamper for å sammenligne sannsynlighetene gitt av de utviklede prediksjonsmodellene.

[5] https://www.pinnacle.com/en/betting-articles/soccer/how-to-calculate-live-odds-for-soccer-betting

Kapittel 4

Datasett

Datasettet vi bruker i dette prosjektet er samlet inn og håndtert av Sportradar. I løpet av fotballkamper har Sportradar utplassert forskjellige *scouts* for å dokumentere hendelser som skjer i kamper. Disse dataene blir gjort tilgjengelig via diverse API-er i form av XML-dokumenter for Sportradars kunder. Høsten 2016 begynte Sportradar med utviklingen av en *Data Lake* hvor blant annet denne dataen blir lagret. Vi har fått tilgang til denne *Data Laken* samt deres relasjonelle databaser for å kunne utføre use caset som det ville blitt gjort internt.

4.1 Scope

Vi tar utgangspunkt i et spesifikt tippemarked når vi skal forsøke å forbedre dagens oddsmodeller. Sportradar tilbyr for fotball mer enn 200 markeder, hvor noen få eksempler kan være vinner (hjemme, borte, uavgjort), over/under et visst antall mål, korrekt stilling, resultat ved pause osv.

Markedet vi fokuserer på er hvilke lag som scorer flest mål i 15-minutts intervaller i en kamp. Sagt på en annen måte blir det som å dele opp en fotballkamp i seks forskjellige kamper og kikke på vinneren av hvert intervall. Ved å utføre eksperimenter og bruke tidligere beskrevede evalueringsmetoder vil vi ta utgangspunkt i den beste algoritmen og modellen, for så å sammenligne sannsynlighetene på de forskjellige tidspunktene i eksempelkamper med de som blir regnet ut av Sportradar.

4.2 Data Lake

Data Lake er et system med en samling av data som har forskjellig skjema og strukturelle former. Data kanaliseres fra alle tenkelige datakilder og lagres i råformat på et felles lagringsområde. Data transformeres ikke før det skal brukes til et spesifikt formål. Denne tilnærmingen kalles «schema-on-read» og gjør at data kan brukes til kjente og foreløpig ukjente analyseformål, i motsetning til f.eks datavarehus som har tilnærmingen «schema-on-write». *Data Lake* er basert på rimelig lagringsteknologi som kan skalere og håndtere eksponentiell vekst av multistrukturert data. Dette studiet blir et eksempel på et typisk bruksområde for en *Data Lake*, hvor det tas utgangspunkt i en liten del av den store mengde data som er lagret for å generere ny verdi. Sportradar har benyttet seg av Amazon S3 som plattform for sin *Data Lake*. Amazon S3, Amazon Simple Storage Service, er designet for å la brukere lagre uendelige mengder data. S3 er ikke et filsystem, men en tjeneste for å lagre objekter hvor dataen blir gjort tilgjengelig gjennom et API som kan bli aksessert hvor som helst. De forskjellige type objektene blir lagret i såkalte *buckets*. En *bucket* er en logisk lagringsenhet i S3 som består av data og metadata som beskriver denne dataen.

4.3 Datastruktur

Hvor detaljert en kamp blir dekket varierer, og representeres ved «coverage level». Dette er en av grunnene til at fokuset vårt vil ligge på de to siste sesongene av engelsk Premier League hvor deknkingsgraden er av høyeste rang. Vi vil ta i bruk referatene fra hver kamp og bruke *Event data* for utarbeidelse av de forskjellige datasettene. En *event* vil i XML-form eksempelvis se slik ut:

```
<event id="21" type="shot_on_target" time="2016-04-02" y="42" x="85" team="home" match_time="15"/>
```

Figur 4.1: Hendelse i kamp

Som en kan lese av figur 4.1 inneholder en *event* informasjon som type, koordinater for hvor på banen det skjedde, hvilket lag det gjelder, samt når i kampen hendelsen skjedde. I tillegg til denne informasjonen kan også noen *events* beskrive hvilke spillere som er involvert. Ved å

gå gjennom disse referatene for hver kamp vi ønsker å ha med i vårt datasett, kan vi dele opp statistikkene til å gjelde for de forskjellige intervallene.

I tilegg til kampreferatene som er tilgjengelige i XML-form, så blir *live odds* utarbeidet av et annet system, hvor alle oddsendringer blir representert i JSON-filer. Disse JSON-filene inneholder alle oddsendringer for alle markeder i en kamp, og ettersom Sportradar tilbyr over 200 markeder, er dette veldig store filer å manøvrere seg gjennom.

Et marked blir representert som et JSON-objekt med tre forskjellige arrays for hvert utfall. Disse arrayene inneholder alle oddsendringer i kampen til et gitt tidspunkt i epoch tid, samt stillingen. Figur 4.2 viser et eksempel på tre oddsendringer for at hjemmelaget skal vinne kampen på tre forskjellige tidspunkter i kampen.

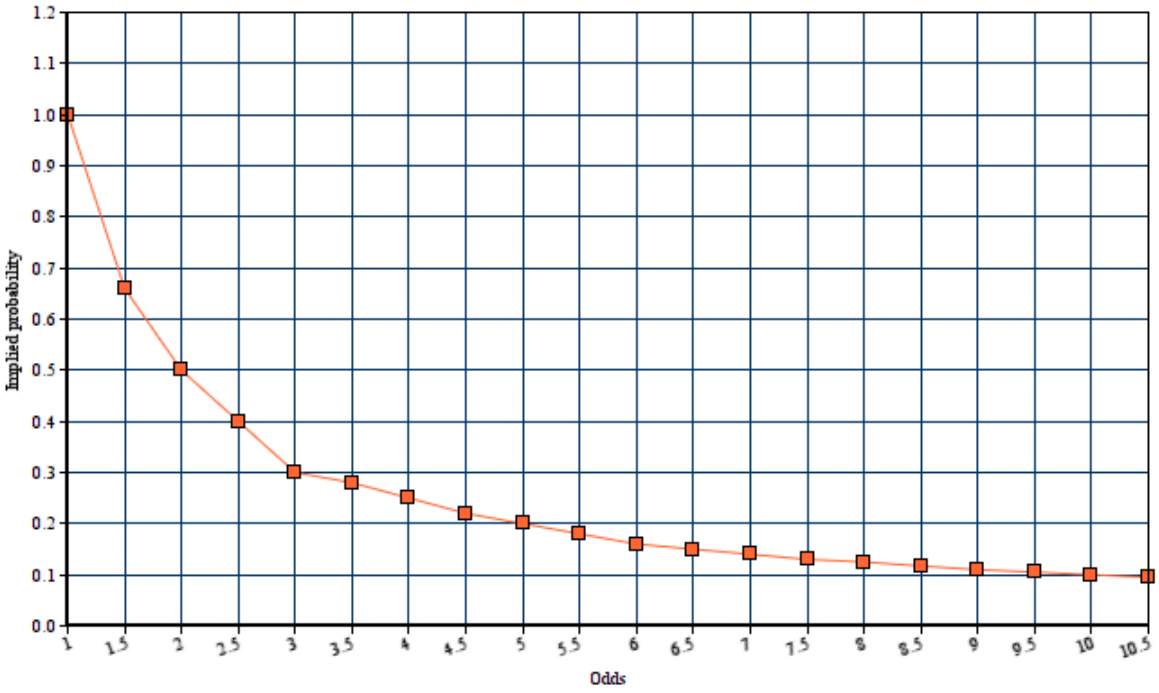

```
- {
      value: 1.65,
      clearedScore: "0:0",
      matchStatus: "FIRST_HALF",
      createdAt: 1184696284000
  },
- {
      value: 1.55,
      clearedScore: "1:0",
      matchStatus: "FIRST_HALF",
      createdAt: 1184696956000
  },
- {
      value: 2.7,
      clearedScore: "1:1",
      matchStatus: "FIRST_HALF",
      createdAt: 1184697166000
  },
```


Figur 4.2: Oddsendringer for hjemmeseier

Denne dataen vil bli brukt i evalueringen av sannsynlighetene for de forskjellige utfallene etter vi har funnet modellen og algoritmen som yter best.

I våre datasett vil oddsene som er satt av Sportradar før kamp for hjemmeseier og borteseier etter full tid bli brukt som et mål på styrkeforholdet mellom de to lagene.

Ettersom forholdene mellom odds og sannsynlighet ikke er lineær, som vist i figur 4.3, velger vi å transformere odds til sannsynlighet før den settes inn i treningssettet. Dette er spesielt viktig skulle en ønske å prosessere disse faktorene som kategoriske variabler ved å gruppere nærliggende variabler i «bins» under utarbeidelsen av maskinlæringsmodeller.



Figur 4.3: Graf som viser forholdet mellom odds og sannsynlighet

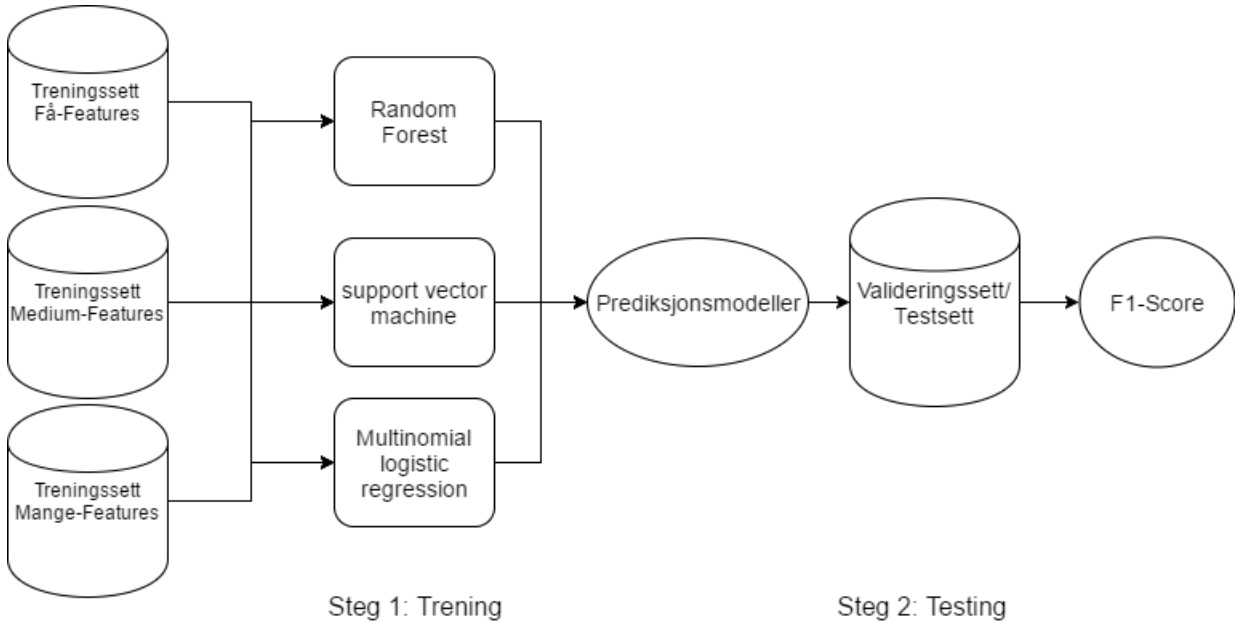
Kapittel 5

Eksperimenter

5.1 Oppsett av eksperiment

Dataene vi bruker for å gjennomføre eksperimentene blir ekstrahert fra XML-filer som finnes i Sportradars *Data Lake*. Ved å bruke Java transformerer vi dataene fra XML til CSV-format, ettersom biblioteker og skyløsninger har innebygget støtte for denne datakilden. Deretter kjører vi maskinlæringsalgoritmer på disse filene. Modellene med Multinomial Logistic Regression blir laget med Amazon Machine Learning som er en tjeneste laget for å utvikle prediktive applikasjoner. Denne tjenesten tilbyr tre typer modeller: binær klassifisering, *multiclass* klassifisering, og regresjon. Eksperimenter med bruk av Random Forest og SVM programmerer vi i R. Grafer og tabeller vil bli konstruert ved hjelp av programmering og vil derfor være forskjellige ut i fra programmeringsspråk og algoritme som brukes.

Ved å videreutvikle figur 2.1 fra avsnitt 2.2 har vi lagd en oversikt over hvordan eksperimentene vil bli gjennomført:



Figur 5.1: Viser gjennomføring av Eksperimentene

Vi vil lage treningssett hvor vi har et med få *features*, et med medium *features* og et med mange *features*. Treningssettene brukes så til å trene opp prediksjonsmodeller med maskinlæringsalgoritmene: Random Forest, Support Vector Machine og Multinomial Logistic Regression. Et evalueringssett vil bli brukt til å evalurere og tune modellene dersom det er nødvendig. For hver av prediksjonsmodellene vil vi regne ut F1-score slik at vi kan se hvilke algoritmer og *feature set* som presterer best sammen. Modellene med best resultat vil senere bli brukt til å sammenligne odds gitt med Sportradar basert på et testsett som hverken er brukt i treningen eller evalueringen.

Treningssett

De forskjellige resultatene vi presenterer innenfor de forskjellige algoritmene vil korrespondere til utkastene av treningssett. Som nevnt definerer vi et treningssett som $D = (x_i, y_i)$. y_i er utfallet og vil i alle treningssettene bli angitt som det samme, en kategorisk attributt. y_i tar verdien 1, X, eller 2 som henholdsvis representerer hjemmeseier, uavgjort, eller borteseier.

Ettersom vi ønsker en modell som kan predikere utfallet av en pågående kamp, trenger vi et treningssett som registrerer hendelser og statistikk i kamper. Selv om vi ikke har detaljert data

som er eldre enn starten av 2016, vil ikke nødvendigvis det være noe hindring. I et forsøk på å lage modeller som kjenner igjen forskjellige lags styrker og kjennetegn, er det mulig at data som er flere år gammel ikke vil være optimalt i treningen av slike modeller, ettersom fotball hele tiden er under utvikling og lagene skifter ut spillere med høy frekvens. Fordi det ikke er noe eldre data er det vanskelig å konkludere med at ikke et større datasett ville vært bedre. Vi har derimot laget noen modeller som blir trent opp med data fra flere forskjellige ligaer for å få en pekepinn på om det kunne vært hensiktsmessig, noe resultatene tyder på at det ikke er. Disse grafene kan finnes i vedlegg A. Vi tror derimot å ha med mer data fra samme liga enn hva som er tilgjengelig kunne vært bra, slik at en får flere kamper hvor de samme lagene møter hverandre og en enda bedre generalisering av viktige mønstre i den spesifikke ligaen.

Optimalt sett tror vi at det hadde vært best med data fra 2 hele sesonger, mens vi kun har tilgang data som tilsvarer en og en halv sesong. Resultatene vist at det er viktig å ha med data fra begge sesongene fordi enkelte lag har over og underprestert. For eksempel vant nyopprykkede Leicester 2015/2016 sesongen, mens regjerende mester Chelsea havnet midt på tabellen. I 2016/2017 sesongen har det normalisert seg hvor Chelsea ligger i toppen, mens Leicester kjemper på nedre halvdel av tabellen. x_i er attributtene i treningssettene som forandrer seg fra utkast til utkast. Det er mange ting som kan påvirke hvor godt et treningssett er. Noe av det viktigste er å ha en passelig mengde treningseksempler. For lite data fører til at modellen ikke finner klare mønstre, mens for mye data kan føre til overtilpasning.

Selv om vi har eksperimentert med mange forskjellige versjoner av treningssett vil vi ta utgangspunkt i tre forskjellige utkast, ettersom dette er de med mest variasjon seg imellom når det kommer til antall *features*. Det er viktig å merke seg at ”match-tid” kun har verdi fra 0 til 5, som skal representere de seks forskjellige 15-minuttsintervallene i en kamp.

Statistikkene som har skjedd opp til et tidspunkt i kampen blir brukt til å predikere hva utfallet blir i neste intervall. Derfor vil eksempelvis alle features untatt styrkeforhold og hvilke lag som spiller ha verdien 0 ved predikering av første intervallet(0-15 min). Tankegangen er at en skal stå på starten av hvert intervall og bruke den informasjonen som er tilgjengelig så langt i kampen til å predikere utfallet i neste intervall. Styrkeforholdet er som nevnt i foregående kapittel sannsynlighetene Sportradar gir lagene for hjemmeseier og borteseier i hele kampen(det tradisjonelle

tippemarkedet ”1x2”), hvor favoritten til å vinne kampen vil ha den høyeste verdien mellom 0 og 1 av de to lagene.

Ved å opprette datakilde under Amazon sine maskinlæringstjenester får en tilgang på masse statistikker for datasettet. Disse statistikkene er å finne i vedlegg C. Tabell 5.1 gir en oversikt over hvilke *features* som er med i de forskjellige modellene, og hvilken korrelasjon de har til utfallet.

	Få features	**Medium Features**	**Mange features**
Hjemmelag	0.01028	0.01013	0.01015
Bortelag	0.01296	0.01286	0.01298
Interval i match	0.00746	0.00727	0.00751
Mål hjemmelag	0.00324	0.00338	0.00351
Mål bortelag	0.00241	0.00232	0.00252
Styrke hjemmelag	n/a	0.04775	0.04825
Styrke bortelag	n/a	0.05043	0.0508
Skudd utenfor mål hjemmelag	n/a	n/a	0.00341
Skudd utenfor mål bortelag	n/a	n/a	0.00519
Hjørnespark hjemmelag	n/a	n/a	0.00467
Hjørnespark bortelag	n/a	n/a	0.00629
Skudd på mål hjemmelag	n/a	n/a	0.00495
Skudd på mål bortelag	n/a	n/a	0.00232
Frispark hjemmelag	n/a	n/a	0.0059
Frispark bortelag	n/a	n/a	0.00687
Gule kort hjemmelag	n/a	n/a	0.00262
Gule kort bortelag	n/a	n/a	0.00702
Røde kort hjemmelag	n/a	n/a	0.00143
Røde kort bortelag	n/a	n/a	0.00378

Tabell 5.1: Oversikt over features med i hver modell

Under vil gå gjennom hvert av treningssettene og ta utgangspunkt i en eksempelkamp for å vise hvordan typisk treningsdata ser ut. Alle settene er like store med kamper fra Premier League og består av 2347 eksempler. Eksempelkampen er en kamp mellom Arsenal og Watford hvor Arsenal vinnner 4-0.

Få features

Hjemmelag	Bortelag	Match tid	Mål H	Mål B	Utfall
Arsenal	Watford	0	0	0	1
Arsenal	Watford	1	1	0	x
Arsenal	Watford	2	1	0	1
Arsenal	Watford	3	2	0	1
Arsenal	Watford	4	3	0	x
Arsenal	Watford	5	3	0	1

Tabell 5.2: Treningsseksempler for en vilkårlig kamp

Medium features

Hjemmelag	Bortelag	Match tid	Mål H	Mål B	Styrke H	Styrke B	Utfall
Arsenal	Watford	0	0	0	0.741	0.111	1
Arsenal	Watford	1	1	0	0.741	0.111	x
Arsenal	Watford	2	1	0	0.741	0.111	1
Arsenal	Watford	3	2	0	0.741	0.111	1
Arsenal	Watford	4	3	0	0.741	0.111	x
Arsenal	Watford	5	3	0	0.741	0.111	1

Tabell 5.3: Treningsseksempler for en vilkårlig kamp

Mange features

Hjemmelag	Bortelag	Match tid	Mål H	Mål B	Styrke H	Styrke B	Skudd på mål H	Skudd på mål B
Arsenal	Watford	0	0	0	0.741	0.111	0	0
Arsenal	Watford	1	1	0	0.741	0.111	3	0
Arsenal	Watford	2	1	0	0.741	0.111	4	1
Arsenal	Watford	3	2	0	0.741	0.111	6	1
Arsenal	Watford	4	3	0	0.741	0.111	11	2
Arsenal	Watford	5	3	0	0.741	0.111	12	3

Skudd utenfor mål H	Skudd utenfor mål B	Cornere H	Cornere B	Gule kort H	Gule kort B	Røde kort H	Røde kort B
0	0	0	0	0	0	0	0
1	0	3	0	0	0	0	0
1	0	6	0	0	0	0	0
1	0	6	0	0	0	0	0
2	2	7	1	0	0	0	0
2	2	8	1	0	1	0	0

Frispark H	Frispark B	Utfall
0	0	1
0	1	x
4	1	1
6	1	1
7	3	x
9	4	1

Tabell 5.4: Treningsseksempler for en vilkårlig kamp

5.1.1 Multinomial Logistic Regression

Utarbeidelsen av modeller med Multinomial Logistic Regression som algoritme blir gjort med Amazon sin maskinlæringstjeneste. For å opprette en datakilde innenfor denne tjenesten må datasettet være i en CSV-fil eller ligge i deres datavarehussytem Redshift. Under opprettelsen av datakilde velger en hva slags datatype de forskjellige attributtene skal være; kategorisk, numerisk, tekst, eller binær. En angir også hvilken attributt som er *target* hvis en ønsker å bruke datakilden til å trene maskinlæringsmodeller.

Feature processing

Ved hjelp av en *data recipe* som angis før en trener opp modeller har en i Amazon mulighet til å velge hvilke attributter som skal bli brukt i læringsprosessen, samt transformere dem på flere forskjellige måter. De forskjellige transformasjonene som er tilgjengelig er:

- N-gram
- Orthogonal Sparse Biagram
- Lowercase

- Remove punctuation
- Quantile Binning
- Normalization
- Cartesian Product

Vi tar i våre eksperimenter bruk *quantile binning* og *cartesian product*, ettersom prøving og feiling har vist at det gir best resultater.

Quantile binning

Prosessoren i denne transformasjonen tar imot to inputs, en numerisk variabel og et «bin» nummer, og sender en kategorisk variabel tilbake. Ved å gruppere observerte numeriske verdier sammen ønsker en å finne ikke-lineære sammenhenger mellom disse verdiene og *target* attributten. *Bin* nummeret representerer hvor mange *bins* de observerte verdiene skal grupperes i, og hver gruppe av verdier vil representere en kategorisk verdi.

Denne transformasjonen blir gjort på alle numeriske verdier med varierende antall *bins*. Passende antall «bins» er funnet ved å utvikle flere forskjellige modeller hvor det kun er denne transformasjonen som endres på, samt studering av statistikkene i datasettet.

Cartesian product

Kartesisk produkt genererer permutasjoner av to eller flere tekst eller kategoriske variabler for å lage et kartesisk produkt. Dette brukes når det er sannsynlig at to attributters forhold til hverandre påvirker utfallet. Amazon bruker en lineær algoritme som ikke fanger opp forholdet mellom variabler, derfor vil denne transformasjonen være hensiktsmessig å bruke i flere av tilfellene. Et eksempel på bruksområde kan være sannsynlighet for hjemmeseier og borteseier for å fremstille et styrkeforhold.

Innstillinger

I tillegg til å angi en oppskrift ved oppsett av maskinlæringsmodeller i Amazon, finnes det også visse innstillinger som kan gjøres. Standardinnstillingene er tilstrekkelig for de fleste modellene, men avhengig av om resultatene er utilfredsstillende grunnet undertilpasning eller overtilpasning kan det være hensiktsmessig å endre på disse parameterene. De forskjellige parameterene som kan endres på er:

- Maksimal modellstørrelse
- Maksimal antall passeringer over treningsdataen
- Valg om å blande rekkefølge på treningsdata
- Reguleringstype og styrke

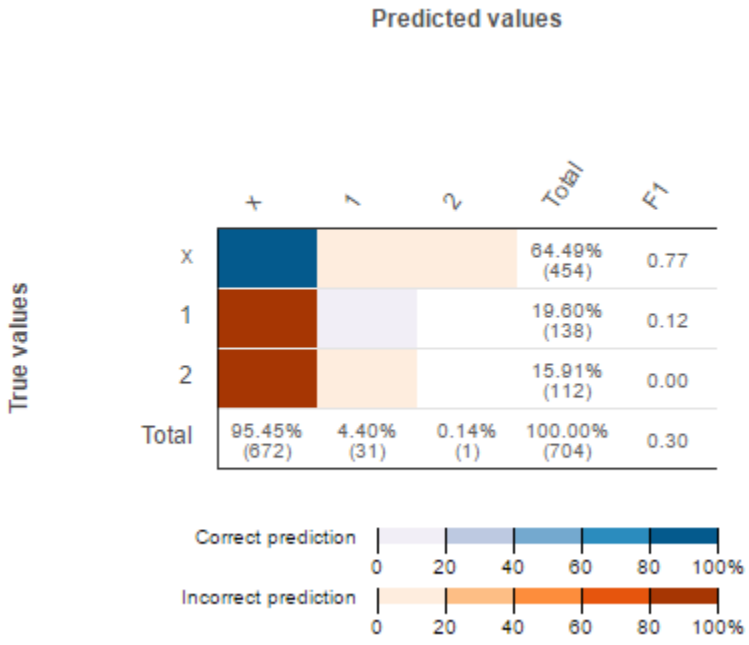
Standardinnstillingene setter en maksimal modell størrelse på 100MB, 10 passeringer over treningsdata, automatisk *shuffling* av treningsdata, samt en mild L2 regulering. Hvor ikke annet er oppgitt i eksperimentene blir disse innstillingene brukt.

Under vil vi gå gjennom de beste modellene for hvert utkast av treningssettene, og forklare hvilke oppskrifter og innstillinger som er brukt for å trene modellene samt resultatene.

Eksperiment med få features

Oppskrift transformerer mål ved hjelp av *quantile binning* til å ligge i seks forskjellige kategorier, og lager et kartestisk produkt av hjemmemål og bortemål.

Resultatene gir en gjennomsnittlig F1-score på 0.296. Figur 5.2 viser resultatene i form av en *confusion matrix*.



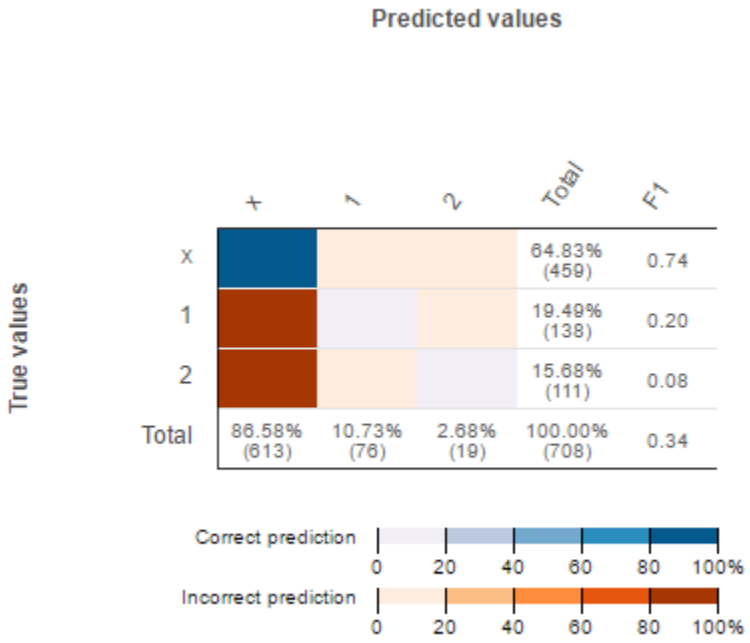
Figur 5.2: Confusion matrix for modell trent opp av treningssett med få *features*

Modellen predikerer riktig på uavgjort i 95,59 prosent av tilfellene. Den har klart å predikere 7,25 prosent av hjemmeseiere, men ingen borteseiere.

Eksperiment med medium features

I dette treningssettet legger vi til sannsynligheten for hjemmeseier og borteseier for hele kampen gitt før kampstart som et tegn på styrkeforholdet mellom lagene. Oppskriften er lik som i foregående eksempel, hvor vi i tillegg legger til et kartesisk produkt mellom sannsynlighetene for hjemmeseier og borteseier. Vi øker også antall passeringer som gjøres over treningsdata fra 10 til 100.

Resultatene gir en gjennomsnittlig F1-score på 0.339. Figur 5.3 viser resultatene i form av en *confusion matrix*.



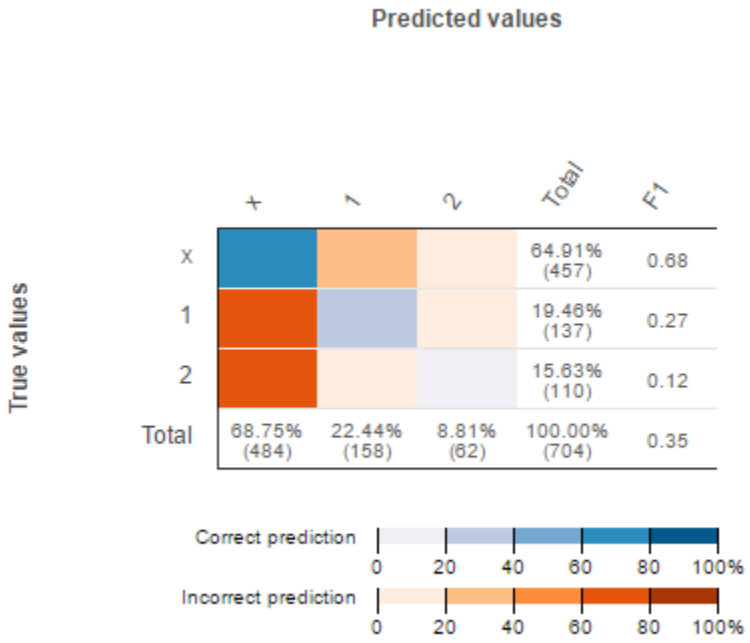
Figur 5.3: *Confusion matrix* for modell trent opp av treningssett med medium *features*

Denne modellen predikerer uavgjort riktig i 86,71 prosent av tilfellene som altså er noe lavere enn i foregående modell. Men i motsetning til foregående modell forbedrer den prosentandelen av riktige predikeringer både for hjemmeseier og borteseier, til henholdsvis 15,22 prosent og 4,5 prosent.

Eksperiment med mange features

I dette treningssettet la vi til mange *features* som korresponderer til statistikker i en kamp. Med så mange forskjellige *features* måtte det mange forsøk med forskjellige oppskrifter og innstillinger til før vi kan finne modellen som yter best. Oppskriften har med de samme *features* som i foregående modeller, men legger også til kartesisk produkt mellom hjemmelag og sannsynligheten for hjemmeseier, samt bortelag og sannsynligheten for borteseier. Også her er antall passeringer over treningsdata økt til 100.

Resultatene gir en gjennomsnittlig F1-score på 0.354. Figur 5.4 viser resultatene i form av en *confusion matrix*.



Figur 5.4: *Confusion matrix* for modell trent opp av treningssett med mange features

Modellen predikerer uavgjort riktig i 69,58 prosent av tilfellene. Igjen er dette nedgang ifra foregående modell, mens riktig prediksjoner av hjemmeseier og borteseier har økt til henholdsvis 29,20 prosent og 9,09 prosent.

Oppsummering

Resultatene forbedrer seg mellom hver iterasjon. Antall riktige predikeringer av uavgjort går ned mellom hver iterasjon, mens antall riktige predikeringer av hjemmeseier og borteseier går opp. Dette er nok grunnet den store mengden uavgjortresultater, hvor den første modellen rett og slett ikke har nok *features* til å predikere annet enn den klassen som forekommer oftest. Den første modellen har eksempelvis bare gjort et forsøk på å predikere en borteseier ut av 704 prediksjoner.

5.1.2 Random Forest

For å kunne kjøre Random Forest på datasettet måtte vi programmere eksperimentet i R ved bruk av CRAN-package randomForest [1]. Som tidligere nevnt sikrer Random Forest for at deler av treningssettet ikke blir inkludert sånn at disse blir brukt til å evaluere modellen. OOB-error blir automatisk kalkulert og regnes ut på samme måte som *error rate* kjent fra avsnitt 2.2.2. Som nevnt er ikke *error rate* en god nok evalueringsmetode og vi regner derfor F1-score (formel 2.5) til hver av modellene.

Under treningsfasen kan vi stille inn hvor mange klassifiseringstrær som skal genereres. Her er det viktig å bruke "riktig" mengde med klassifiseringstrær. Ved å bruke for mange klassifiseringstrær får vi høy beregningstid uten at prediksjonsmodellen tjener på det. Ved å bruke for få klassifiseringstrær får vi en prediksjonsmodell som ikke er optimal. I hvert av eksperimentene med Random Forest finner vi antall trær som bør brukes for det gitte treningssettet ved å se hvor forholdet mellom antall trær og *out-of-bag* (OOB) error flater ut.

Treningssettene vi bruker er ubalansert, det vil si at en klasse er større enn de andre (majoriteten av datapostene i vårt tilfelle har utfall = x). Random Forest vil forsøke å holde den totale feilraten lav, dette medfører at algoritmen holder feilraten lav i de største klassene og tillater høyere feilrate i de mindre klassene. Vi har derfor trent opp en prediksjonsmodell med et ubalansert treningssett og en med et balansert treningssett i hvert av eksperimentene gjort med Random Forest. Grunnen til at vi har gjort dette er for å se om et balansert treningssett kunne gi oss en bedre prediksjonsmodell.

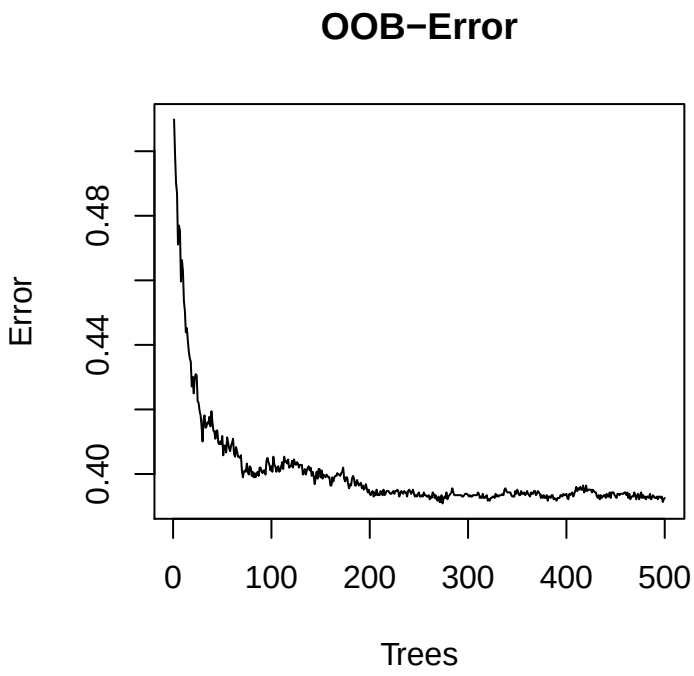
Balansering av et treningssett med Random Forest algoritmen kan gjøres ved å nedskalere majoritetsklassen eller ved å vekte minoritetsklassen høyere enn de andre (høyere straff for å feilpredikere minoritetsklassen). I tilfellet hvor majoritetsklassen blir nedskalert risikerer vi å miste viktig informasjon og ved bruk av vekting risikerer vi at den totale feilraten øker selv om feilraten for minoritetsklassen blir lavere. Vekting og balansering av treningssettet er to jevngode metoder når det kommer til å fordele feilraten likt utover klassene [5]. Nedskalering av treningssettet vil dog være beregningsmessig mer effektiv enn vekting [5] og vi valgte derfor å ta i bruk den

[1] https://cran.r-project.org/web/packages/randomForest/index.html

metoden for disse eksperimentene.

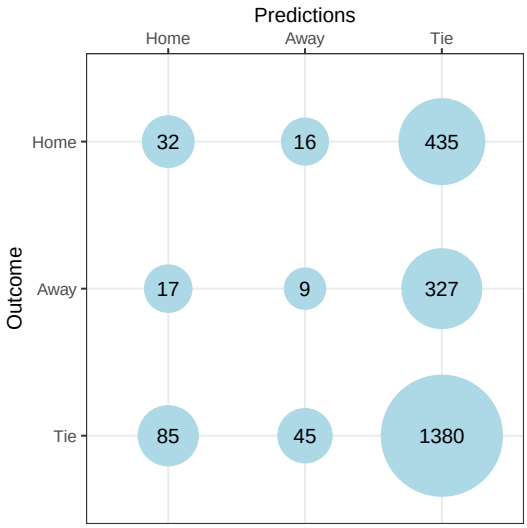
Eksperiment med få features

I eksperimentet brukte vi treningssett med få *features* (tabell 5.1) til å trene opp modellen. Figur (5.5) viser at OOB-error ikke forbedrer seg etter bruken av ca. 200 trær.



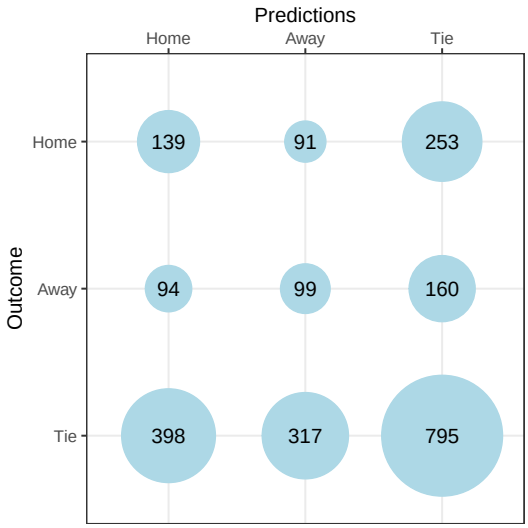
Figur 5.5: Viser plot over out of bag error og antall trær brukt i treningssett 1

Vi valgte derfor å trene opp modellen med 250 klassifiseringstrær og endte opp med en modell med OOB-error på 39,43 prosent. I figur (5.6) kan vi se resultatene fra treningen av modellen. Der kommer det tydelig frem at algoritmen har problemer med å predikere at en kamp ender med noe annet enn uavgjort (utfall = x).



Figur 5.6: Viser resultat fra prediksjon av ubalansert treningssett

[]

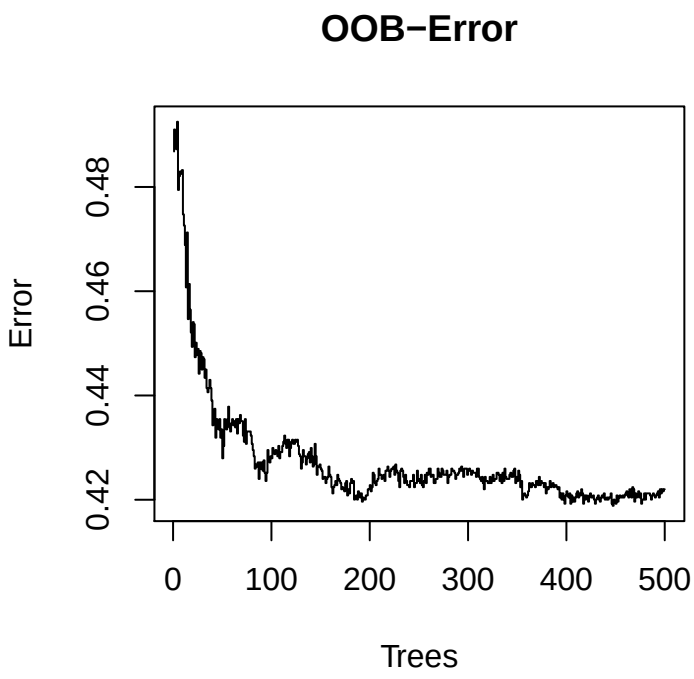


Figur 5.7: Viser resultat fra prediksjon av balansert treningssett

For å balansere treningssettet valgte vi å ta 353 instanser fra hvert av utfallene (353 hjemme seier, 353 uavgjort og 353 borteseier). Vi fikk da en modell med OOB-error lik 55,97 prosent og resultatene kan sees i figur 5.7. Balansering av treningssettet gav oss derfor ikke høyere nøyaktighet, men OOB-error ble jevnere fordelt utover utfallene når vi brukte det balanserte treningssettet. For å se hvilken av modellene som var best regnet vi ut F1-score før og etter vi balanserte treningssettet. F1-score uten et balansert treningssett var lik 0,30 og F1-score med et balansert treningssett var lik 0,35.

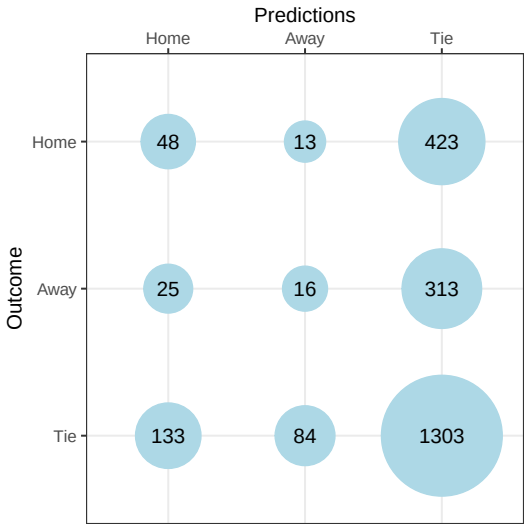
Eksperiment med medium features

I eksperimentet brukte vi treningssett med medium *features* (tabell 5.1) til å trene opp modellen. For dette treningssettet kan vi i figur 5.8 se at OOB-error flater ut etter bruk av ca. 400 klassifiseringstrær. Til å trene opp modellen brukte vi 450 klassifiseringstrær og fikk OOB-error på 42,03 prosent.



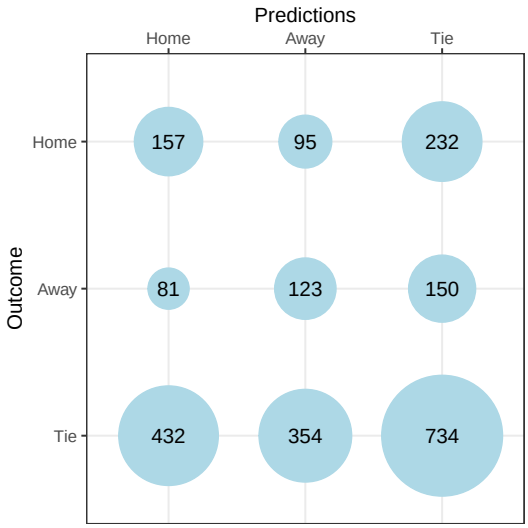
Figur 5.8: Viser plot over out of bag error og antall trær brukt i treningssett 2

Figur 5.9 viser at også denne prediksjonsmodellen har vanskeligheter med å predikere at en kamp ender med annet enn uavgjort når den trenes med et ubalansert treningssett. Vi balanserte derfor treningssettet og brukte 353 instanser fra hvert av utfallene. Som vi kan se i figur 5.10 ble feilprediksjoner fordelt bedre utover klassene.



Figur 5.9: Viser resultat fra prediksjon av ubalansert treningssett

[]

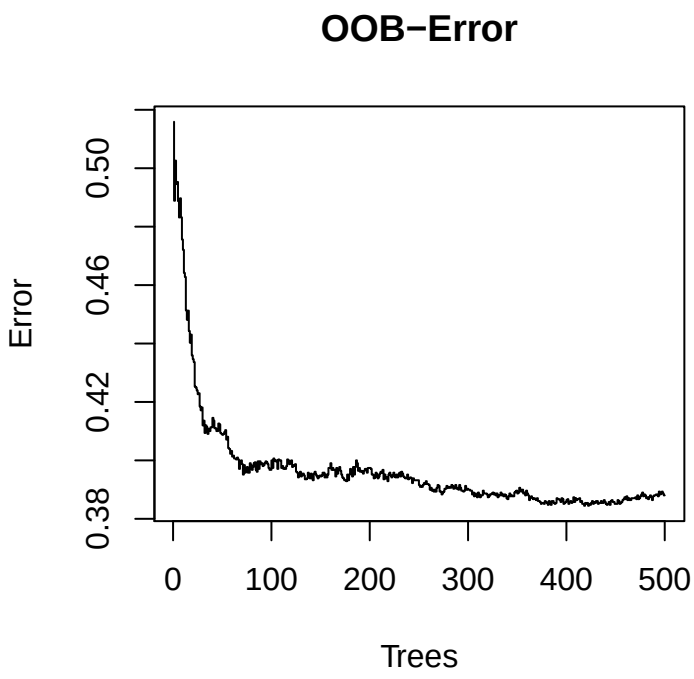


Figur 5.10: Viser resultat fra prediksjon av balansert treningssett

Til tross for at prediksjonsmodellen som ble trent opp på et balansert treningssett hadde høyere OOB-error (57,00%) fikk den en bedre F1-score. F1-score uten et balansert treningssett var lik 0.31 og F1-score med et balansert treningssett var lik 0,36.

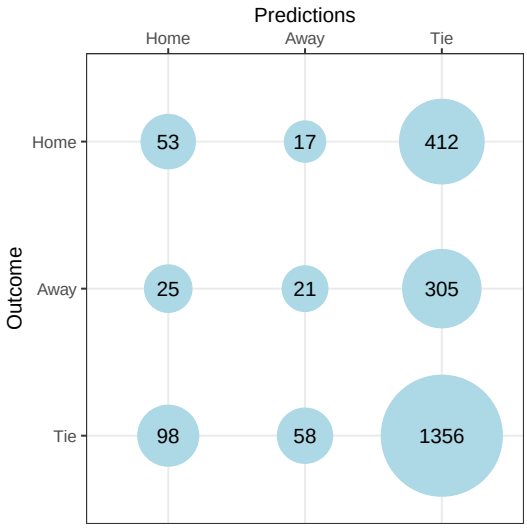
Eksperiment med mange features

I eksperimentet brukte vi treningssett med mange *features* (tabell 5.1) til å trene opp modellen. For dette treningssettet kan vi i figur 5.11 se at OOB-error flater ut etter bruk av ca. 400 klassifiseringstrær. Til å trene opp modellen brukte vi 450 klassifiseringstrær og fikk OOB-error på 39,02 prosent.



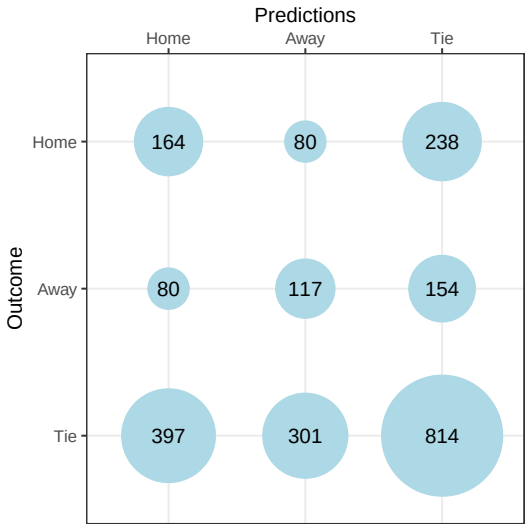
Figur 5.11: Viser plot over out of bag error og antall trær brukt i treningssett 3

Figur 5.12 viser resultater av prediksjoner gjort med en prediksjonsmodell som er trent opp med et ubalansert treningssett. Treningssettet ble balansert ved å ta 351 instanser fra hvert av utfallene og vi fikk da en OOB-error på 53,3 prosent. Figur 5.13 viser resultater av prediksjoner gjort med en prediksjonsmodell som er trent opp med et balansert treningssett.



Figur 5.12: Viser resultat fra prediksjon av ubalansert treningssett

[]



Figur 5.13: Viser resultat fra prediksjon av balansert treningssett

F1-score uten et balansert treningssett var lik 0,33 og F1-score med et balansert treningssett var lik 0,38.

Oppsummering

Resultatene forbedrer seg mellom hver iterasjon. Det er tydelig at flere *features* gir en prediksjonsmodell som generealiserer bedre. Vi fant at prediksjonsmodellene i alle eksperimentene presterer bedre når det er bruk et balansert treningssett til trening. Beste resultat ble F1-score lik 0,38. Selv om vi har brukt balansert treningssett så ser vi at prediksjonsmodellene har størst problemer med å predikere utfallene «hjemme» og «borte». I prediksjonsmodellen som har høyest F1-score så er *precision* og *recall* høyest for utfallet «uavgjort».

5.1.3 Support Vector Machine

For å kunne ta i bruk support vector macine algoritmen på datasettet måtte vi programmere eksperimentet i R ved bruk av CRAN-package e1071 [17]. Siden vår data ikke kunne separeres lineært brukte vi *kernel trick*: Gaussian *Radial Basis Function*. SVM har ikke en innebygd funk-

sjon som setter av deler av treningssettet til evaluering og vi ble nødt til å gjøre dette manuelt. Vi valgte å ta bort ca. 30 prosent av treningssettet i hvert av eksperimentene og brukte det som valideringssett. Vi endte da opp med 1643 treningseksempler og 703 eksempler til evaluering. Valideringssettet ble brukt til å regne ut *error rate* og F1-score.

SVM har problemer når det brukes et ubalansert treningssett og vi ble nødt til å håndtere dette. Siden vi allerede hadde fjernet 30 prosent av treningssettet valgte vi å balansere treningssettet ved å vekte minoritetsklassen høyere enn de andre (høyere straff for å feil-predikere minoritetsklassen). Dersom vi hadde nedskalert treningssettet til å ha like mange eksempler fra hvert utfall ville vi endt opp med få eksempler og sannsynligvis en dårligere prediksjonsmodell. Standard vektes alle klassene likt og har vekt lik 1 i CRAN-package e1071 [17]. Optimal vekting av klassene fant vi i hvert av eksperimentene ved å bruke tuning funksjonen sammen med forskjellige kombinasjoner av vekting.

I hvert av eksperimentene måtte vi stille inn sensitiviteten til algoritmen ved å sette verdien til *cost* og *gamma* variablene. Dette gjøres fordi SVM forsøker å splitte alle klasser nøyaktig fra hverandre, noe som ikke alltid er mulig og ved å stille på sensitiviteten kan vi få en mer robust modell. Ved å sette høy *cost* straffes misklassifisering mye og vi får lav bias og høy varians og motsatt tilfelle ved å sette lav *cost*. Verdien av *gamma* bestemmer hvor stor innflytelse hver support vector har for å bestemme tilhørligheten til en klasse. Lav *gamma* vil si at vektorene har høy innflytelse (påvirker andre support vektorer selv om distansen mellom dem er stor) og lav *gamma* gir at hver support vektor ikke har stor innflytelse. Vi brukte tuning funksjonen til SVM for å finne de optimale *cost* og *gamma* verdiene.

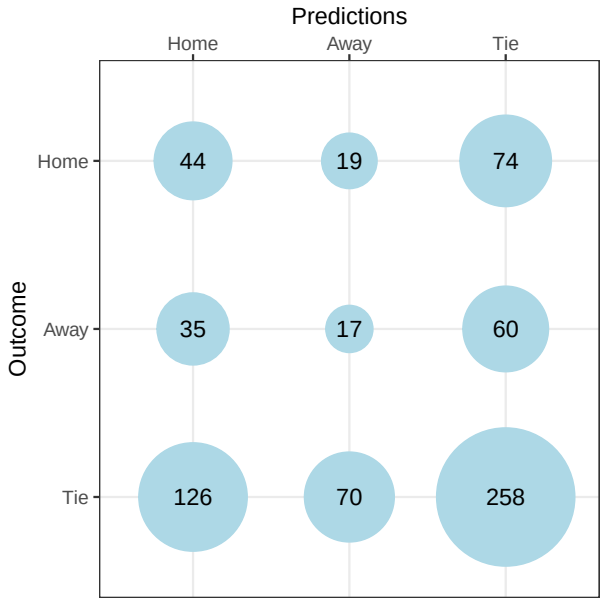
Som nevnt i avsnitt 2.2.1 er det to strategier som kan tas i bruk når SVM skal løse *multi-class classification* oppgaver. Når SVM programmers til å løse denne typen oppgaver i R følger den *One Against One* strategien [16].

Eksperiment med få features

I eksperimentet brukte vi treningssett med få *features* (tabell 5.1) til å trene opp modellen. Ved tuning av SVM modellen fant vi at beste resultater oppnås ved å sette *cost* lik 100 og *gamma* lik

0,24. Verdiene som ble brukt til å vekte utfall 1, utfall 2 og utfall x var henholdsvis 3, 4 og 1.

Etter å trent opp modellen med de optimale parameterene hadde den en *error rate* lik 54,62 prosent når den predikerte valideringssettet. F1-score var lik 0,34. Figur 5.14 viser resultatene fra prediksjonen av valideringssettet.

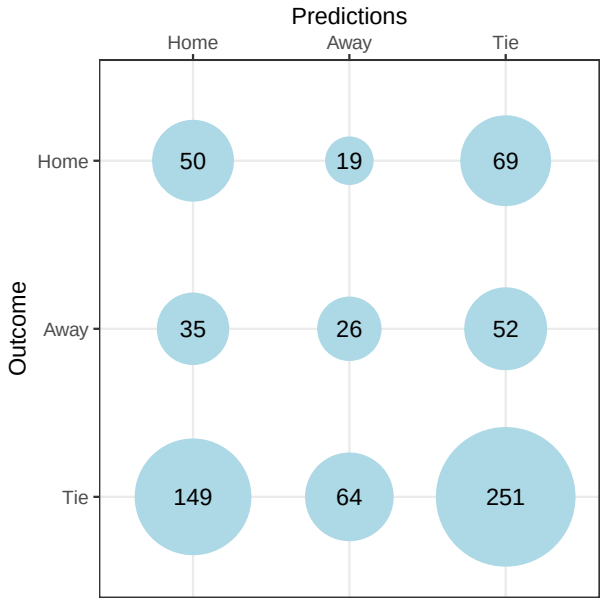


Figur 5.14: Viser resultater fra predikering av valideringssett

Eksperiment med medium features

I eksperimentet brukte vi treningssett med medium *features* (tabell 5.1) til å trene opp modellen. Ved tuning av SVM modellen fant vi at beste resultater oppnås ved å sette *cost* lik 100 og *gamma* lik 0,21. Verdiene som ble brukt til å vekte utfall 1, utfall 2 og utfall x var henholdsvis 3, 4 og 1.

Modellen med de optimale parameterene fikk *error rate* lik 54,27 prosent når den predikerte valideringssettet. F1-score var lik 0,36. Figur 5.15 viser resultatene fra prediksjonen av valideringssettet.

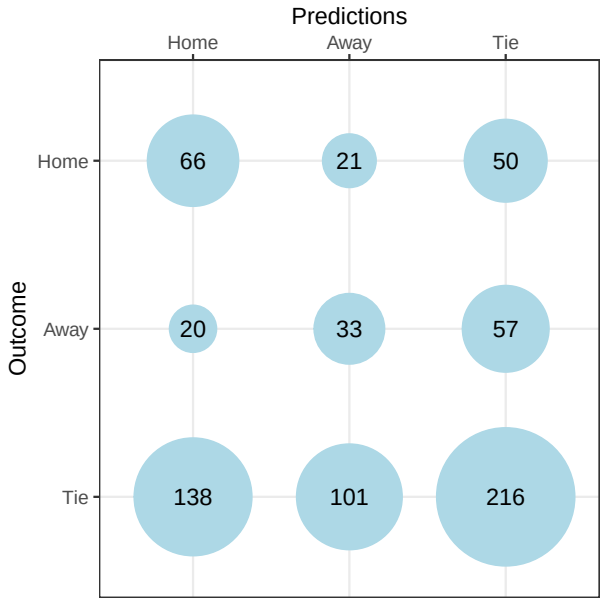


Figur 5.15: Viser resultater fra predikering av valideringssett

Eksperiment med mange features

I eksperimentet brukte vi treningssett med mange *features* (tabell 5.1) til å trene opp modellen. Ved tuning av SVM modellen fant vi at best resultat fikk vi ved å sette *cost* lik 10 og *gamma* lik 0,49. Verdiene som ble brukt til å vekte utfall 1, utfall 2 og utfall x var henholdsvis 3, 4 og 1.

Modellen med de optimale parameterene fikk *error rate* lik 55,13 prosent når den predikerte valideringssettet. F1-score var lik 0,39. Figur 5.16 viser resultatene fra prediksjonen av valideringssettet.



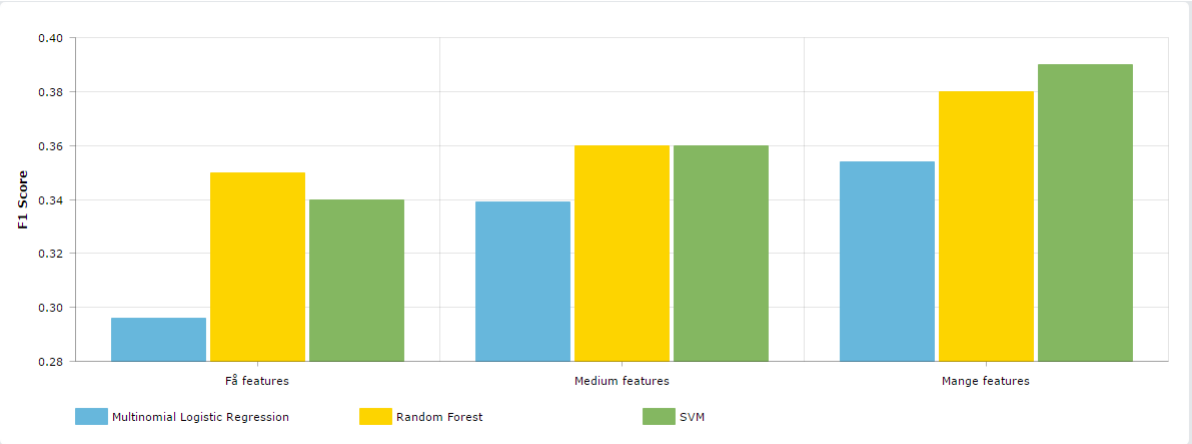
Figur 5.16: Viser resultater fra predikering av valideringssett

Oppsummering

Prediksjonsmodellen trent med mange *features* presterte best. Beste resultat ble F1-score lik 0,39. Selv om vi vekter klassene er det tydelig at prediksjonsmodellene har størst problemer med å predikere utfallene «hjemme» og «borte». I prediksjonsmodellen som har høyest F1-score så er *precision* lav i disse utfallene og *recall* scorer litt høyere en *precision*. *Precision* og *recall* for utfallet «uavgjort» er med på å øke den totale F1-score.

5.2 Resultater

Figur 5.17 viser en sammenligning av F1-Score de forskjellige eksperimentene hadde. Ved første øyekast kan det se ut som veldig lave resultater, men vi må ha i bakhodet at dette er et vanskelig tippemarked å predikere i. Det kan sammenlignes som å skulle vedde på vinneren av en fotballkamp, bare at kampen varer 15 minutter istedet for 90 minutter. «Baseline F1 Score», det vil si F1-score hvis modellen alltid predikerer den mest frekvente klassen(uavgjort), er 0.262. Alle modellene presterer bedre enn dette.

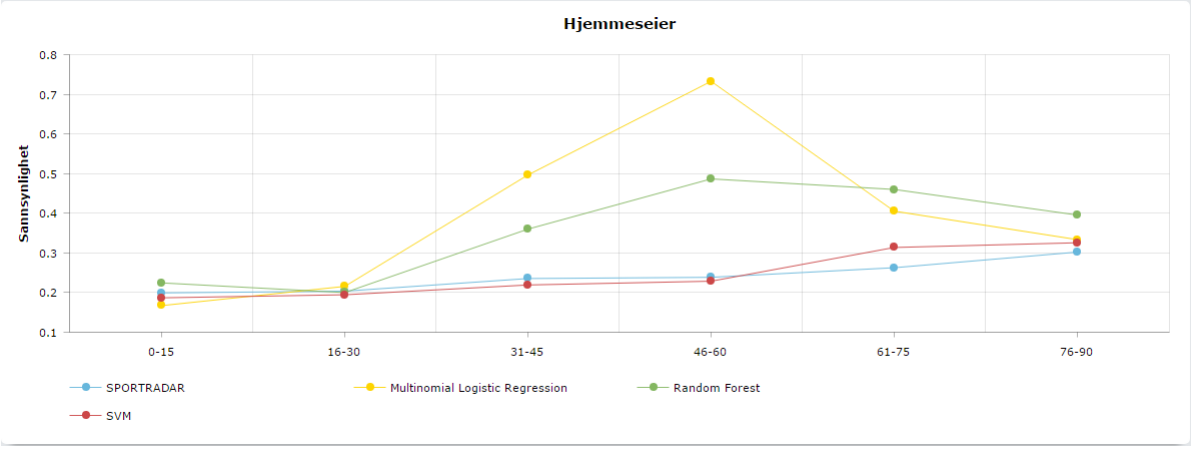


Figur 5.17: Resultater fra eksperimentene

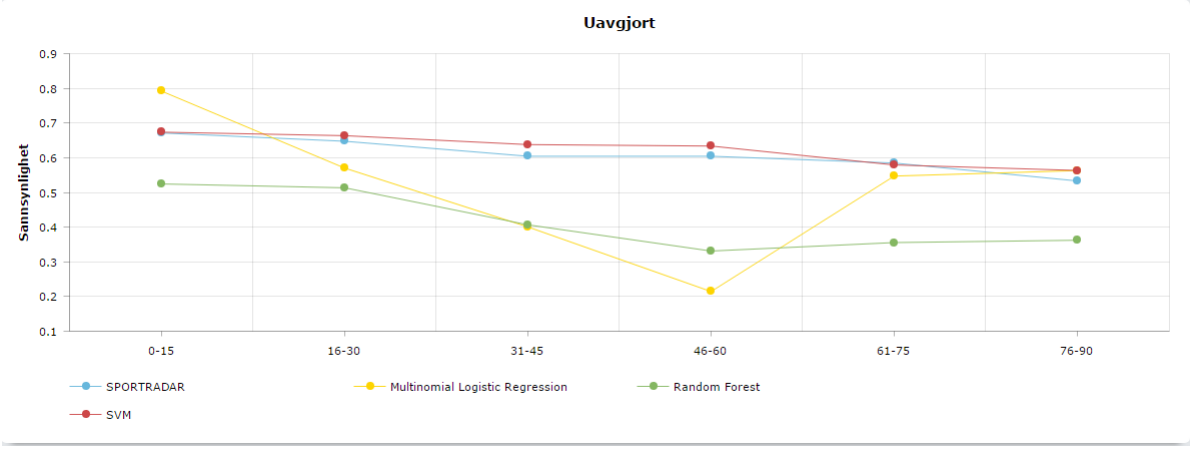
Som vi kan lese av figuren ovenfor yter modellene med mange *features* best. F1-score for disse modellene viser at SVM er den algoritmen som presterer best. For å ta en nærmere kikk på resultatene har vi tatt utgangspunkt i en eksempelkamp for å sammenligne sannsynlighetene modellene gir med de sannsynlighetene Sportradar tilbyr i de forskjellige intervallene.

Fotballkampen vi tar utgangspunkt i er kamp mellom Manchester City og Liverpool som ble spilt 19/03-2017. Stillingen i denne kampen var 1-1, hvor første målet ble scoret av Liverpool i minutt 51, og andre målet ble scoret i minutt 69. Det vil si at intervallene 46-60 endte med borteseier, og 61-75 endte med hjemmeseier, mens resten var uavgjort.

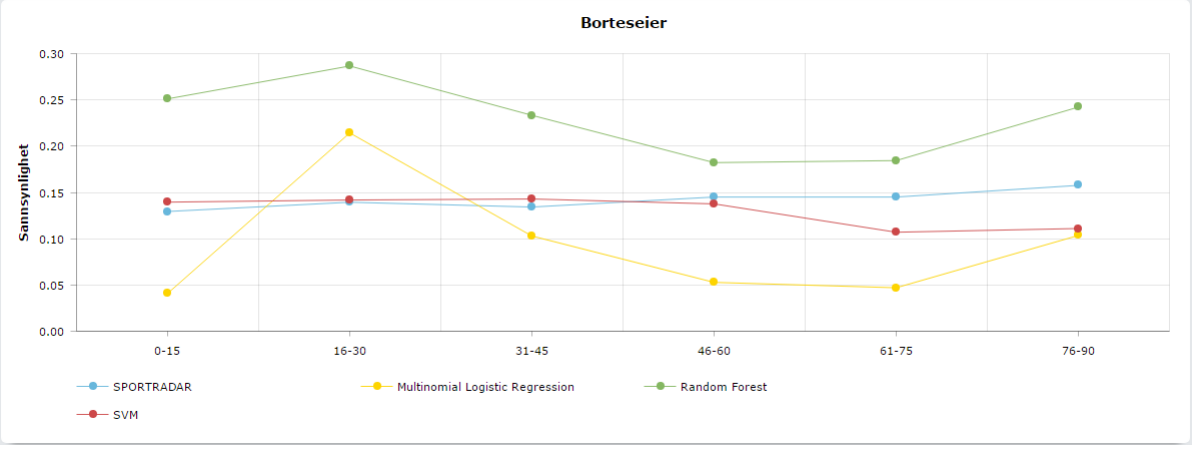
Figur 5.18 viser en sammenligning av sannsynlighetene for hjemmeseier ved starten av de forskjellige intervallene. Figur 5.19 og figur 5.20 viser henholdsvis sannsynlighetene for uavgjort og borteseier.



Figur 5.18: Sammenligning av sannsynlighet for hjemmeseier



Figur 5.19: Sammenligning av sannsynlighet for uavgjort



Figur 5.20: Sammenligning av sannsynlighet for borteseier

Som vi kan lese av figurene er modellen med SVM som algoritme den som følger sannsynlighetene Sportradar opererer med tettest. Alle algoritmene og Sportradar tilbyr relativt like sannsynligheter for uavgjort i de to første og de to siste intervallene. Multinomial Logistic Regression predikerer hjemmeseier i intervallet 46-65 med unormalt høy sannsynlighet, når det faktisk ble borteseier. Random Forest har i samme intervall også høy sannsynlighet for hjemmeseier, men til gjengjeld har den også høyest sannsynlighet for borteseier blant sannsynlighetsgiverene.

For å illustrere hva som er bra sannsynligheter i et tippeselskaps øyne, lager vi en oversikt over

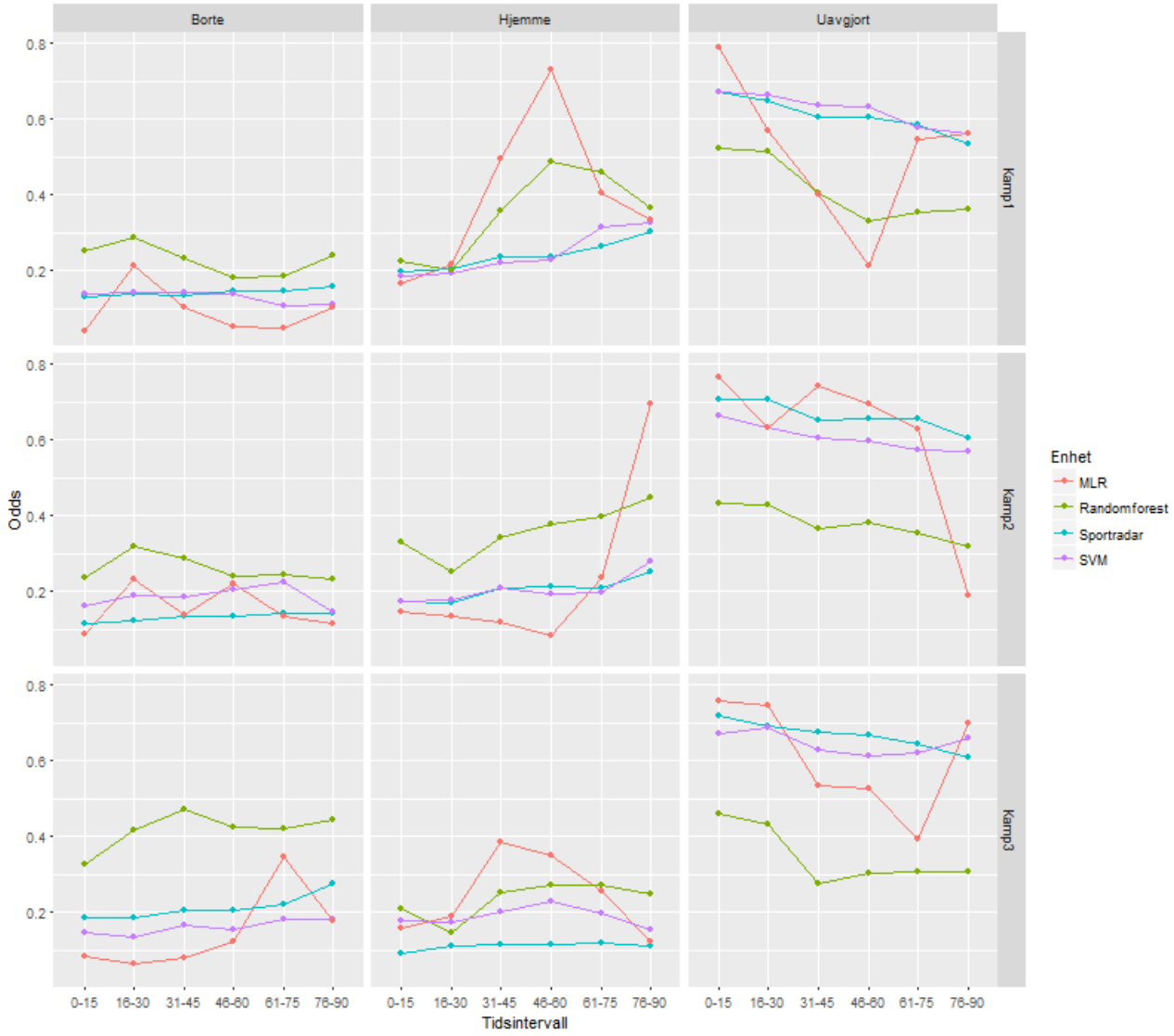
hvilken av disse modellene som hadde tjent inn mest penger i denne eksempelkampen. Vi tar utgangspunkt i en innsats på 100,- NOK på hvert utfall i de forskjellige intervallene. Det er selvfølgelig ikke tilfellet i virkeligheten at det er like mye innsats på hvert av utfallene, men det er dette tippeselskapene ønsker og justerer ofte odds slik at et lite spilt utfall skal bli mer attraktivt. Tippeselskaper operer i tillegg med en margin, som gjør at den samlede sannsynligheten for alle utfallene er større enn 1 og gevinsten øker dermed for tippeselskapet. Denne har vi naturligvis fjernet fra oddsen som Sportradar tilbyr i sammenligningene vi gjør i dette kapittelet.

	0-15	16-30	31-45	46-60	61-75	76-90	**SUM**
Sportradar	+151	+146	+135	-389	-81	+113	+75 NOK
SVM	+152	+149	+143	-427	-19	+123	+121 NOK
Random Forest	+109	+105	+54	-249	+83	+24	+126 NOK
Multinomial Logistic Regression	+174	+146	+51	-1594	+53	+122	-1048 NOK

Tabell 5.5: Gevinst for tippeselskap ved 100,- NOK innsats på hvert utfall i hvert intervall

Tabell 5.5 viser at Sportradar og modellen med SVM gir ganske like gevinster og tap fordi sannsynlighetene følger hverandre tett. Random Forest er i dette eksempelet den sannsynlighetsgiveren med størst gevinst. Multinomial Logistic Regression gir altfor lav sannsynlighet for borteseier i intervallet 46-60 og taper dermed mye penger.

Legger vi til to kamper får vi sannsynlighetsfordelingen vist i figur 5.21 og gevinstfordelingen som vist i tabell 5.6.



Figur 5.21: Sammenligning av sannsynligheter for tre kamper

	0-15	16-30	31-45	46-60	61-75	76-90	**SUM**
Sportradar	+72	+17	+434	-90	-1020	-159	-746 NOK
SVM	-79	+44	+420	-155	-369	-105	-123 NOK
Random Forest	+174	+79	+20	-239	-92	-131	-189 NOK
Multinomial Logistic Regression	-566	-133	+330	-1327	-489	-290	-2475 NOK

Tabell 5.6: Gevinst for tippeselskap ved 100,- NOK innsats på hvert utfall i hvert intervall i tre kamper

Som tabellen ovenfor viser går ingen av modellene i pluss ved innsats i disse tre kampene. Det skyldes nok at det tilfeldigvis ble trukket ut tre kamper med mye mål i. I en av kampene undervurderer overraskende Sportradar et av lagene veldig i forhold til de andre modellene, noe som

gjør at de går betraktelig mer i minus enn modellene med SVM og Random Forest. Modellen med Multinomial Logistic Regression(MLR) ser ut til å overtilpasse seg til treningsdata og svinger veldig i sannsynlighetene til tider. I dette eksempelet er modellen med SVM som har "størst" gevinst.

Tabell 5.7 viser de faktiske utfallene i hvert intervall i de forskjellige kampene. H, U, B står henholdsvis for hjemmeseier, uavgjort, og borteseier.

	0-15	16-30	31-45	46-60	61-75	76-90
Kamp 1	U	U	U	B	H	U
Kamp 2	U	H	U	U	B	B
Kamp 3	B	U	U	U	H	U

Tabell 5.7: Faktiske utfall i hvert intervall innenfor de tre kampene

Kapittel 6

Oppsummering og anbefalinger til videre arbeid

6.1 Oppsummering og konklusjoner

I dette prosjektet har vi utviklet maskinlæringsmodeller som tar i bruk forskjellige algoritmer med den hensikt å forbedre sannsynlighetene i et spesifikt tippemarked i fotball. Resultatene fra forrige kapittel viser at selv om maskinlæringsmodeller har omtrentlig samme evalueringsresultater, så kan det variere mye i sannsynlighetene de gir for hvert utfall. Vi har brukt F1-score som evalueringsmetode for å sammenligne våre eksperimenter, før vi plukket ut de beste maskinlæringsmodellene for å sammenligne sannsynlighetene de gir med de som blir tilbydd av Sportradar den dag idag.

Vi har gjennom vårt iterative arbeid vist hvordan en best setter sammen *features* for å få en best mulig prediksjonsmodell. Forrige kapittel viste at en maskinlæringsmodell med SVM som algoritme gir nesten helt like sannsynligheter som Sportradar utarbeider i det spesifikke tippemarkedet. Med tanke på at det ikke er mer enn halvannen sesong med detaljert informasjon om fotballkamper i Premier League, vil maskinlæringsmodellene muligens kunne yte enda bedre jo større mengde data og jo mer detaljerte statistikker som blir tilgjengelig i fremtiden. Eksperimentene gir stor grunn til å tro at maskinlæring kan ta over for mye manuelt arbeid i fremtiden.

6.2 Svar på forskningsspørsmål

- *Hvordan kan en best mulig sette sammen hendelser fra fotballkamper for å lage et feature set?*

  Ved iterativt arbeid har vi testet flere kombinasjoner med *features* for å finne det beste subsettet. Kombinasjonene av *features* testet vi med å utvikle flere prediksjonsmodeller, for deretter å sammenligne evalueringsresultatene. Vi startet med få *features*, og økte antallet for hver iterasjon. I kapittel 6 viste vi hvordan prediksjonsmodeller trent opp med tre forskjellige *feature set* presterte. Vi kom frem til at prediksjonsmodellene presterte best når de ble trent opp av *feature set* som inneholdt mye og detaljert informasjon om fotballkampen.

- *Hvordan presterer maskinlæringsalgoritmer når det kommer til predikering av utfall i et spesifikt tippemarked innenfor fotball?*

  Ved å ta utgangspunkt i tre maskinlæringsalgoritmer har vi utviklet prediksjonsmodeller og evaluert disse ved bruk av F1-score. Vi kom frem til at det var vanskelig å predikere utfallet. Dette kommer av at det er et vanskelig tippemarked og at fotball er uforutsigbart. Prediksjonsmodellene hadde problemer med å finne tydelige mønstre som skilte de tre utfallene. Det er få tilfeller hvor prediksjonsmodellene finner mønstre som tilsier at intervallet skal ende med en av de underrepresenterte klassene «hjemmeseier» eller «borteseier». Det er dog slik at alle prediksjonsmodellene vi har utviklet har en høyere F1-score enn hva en prediksjonsmodell som kun predikerer den mest representerte klassen ville hatt.

- *Er det mulig å forbedre live odds i et spesifikt tippemarked kun ved hjelp av maskinlæring?*

  Vi har tatt utgangspunkt i tre eksempelkamper og den beste prediksjonsmodellen innenfor hver algoritme. Vi sammenligner sannsynlighetene disse prediksjonsmodellene gir for hvert utfall i hvert intervall med hverandre og de sannsynlighetene som er utarbeidet av Sportradar. Resultatene viser at to av tre prediksjonsmodeller gir bedre gevinster enn Sportradar hvor innsatsen er likt fordelt over hvert av utfallene. Basert på eksperimentene vi har gjort har vi kommet frem til at maskinlæring kan brukes til å forbedre *live odds* i et

spesifikt tippemarked.

6.3 Diskusjon og anbefalinger til videre arbeid

Selv om resultatene kan tyde på at maskinlæringsmodeller kan bli brukt til å forbedre odds i et spesifikt tippemarked, bør nok sammenligningen av sannsynligheter gjøres over en stor mengde kamper. Sammenligningen i forrige kapittel er gjort manuelt og er ekstremt tidkrevende arbeid. Derfor vil første steg i videre arbeid være å finne en egnet evalueringsmetode av sannsynlighetene, som f.eks vist i tabell 5.5, for så å automatisere denne prosessen og evaluere et stort nok testsett. Som erfart med Multinomial Logistic Regression kan maskinlæringsmodeller overtilpasse seg treningsdata og i enkelte tilfeller gi altfor høy/lav sannsynlighet på et utfall. På bakgrunn av dette vil det nok være lurt med en form for øvre og nedre grense på hva en tillater en eventuell applikasjon å sette som odds, avhengig av hvilken maskinlæringsmodell en ender opp med. Det kan også være interessant å bruke dette prosjektet som et fundament for å predikere i andre tippemarkeder, ved å endre på *features* som er markedsspesifikke.

6.4 Refleksjoner

Arbeidet med å finne frem til de beste *feature sets* har vært tidkrevende arbeid med den tilnærmingen vi har brukt. Det har vært en stor fordel at vi har hatt gode kjennskaper til domenet(fotball), noe som gjorde at vi raskt hadde en forståelse for hvilke *features* som muligens kunne være hensiktsmessig å ta med i de forskjellige iterasjonene.

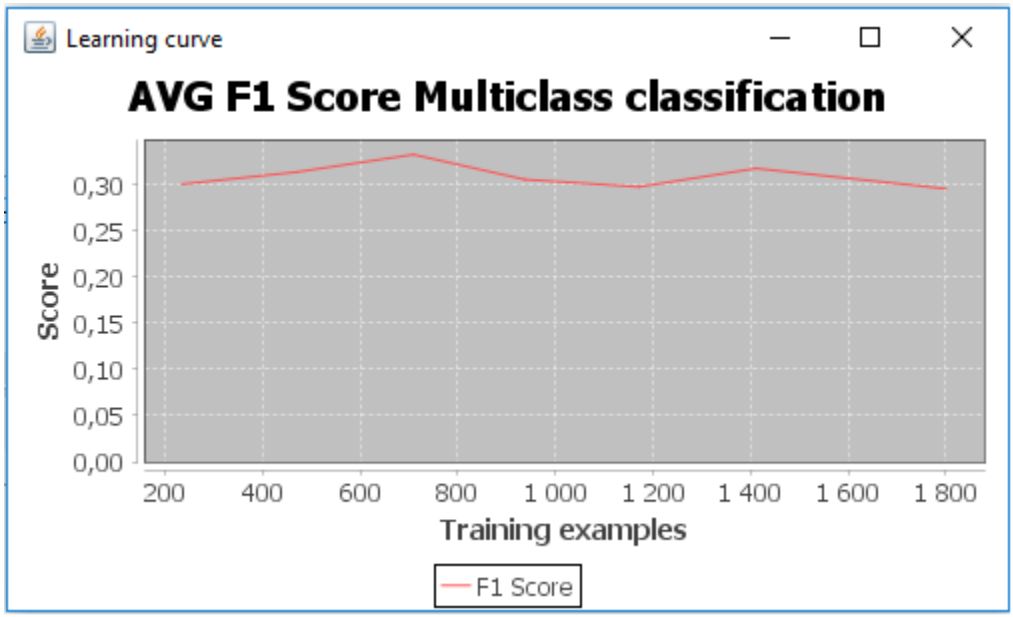
Den teoretiske delen av dette prosjektet har vært viktig for å finne frem til hvilke maskinlæringsalgoritmer som kan være aktuelle å bruke i vår utvikling. Det er viktig å påpeke at teorien innenfor maskinlæring kun gir en pekepinn på hvilke bruksområder de forskjellige algoritmene har og at det er få fasitsvar. Dette gjør at det kreves eksperimentering innenfor det spesifikke domenet for å finne ut hvorvidt de egner seg.

Tillegg A

Tilleggsinformasjon - Mengde treningsdata

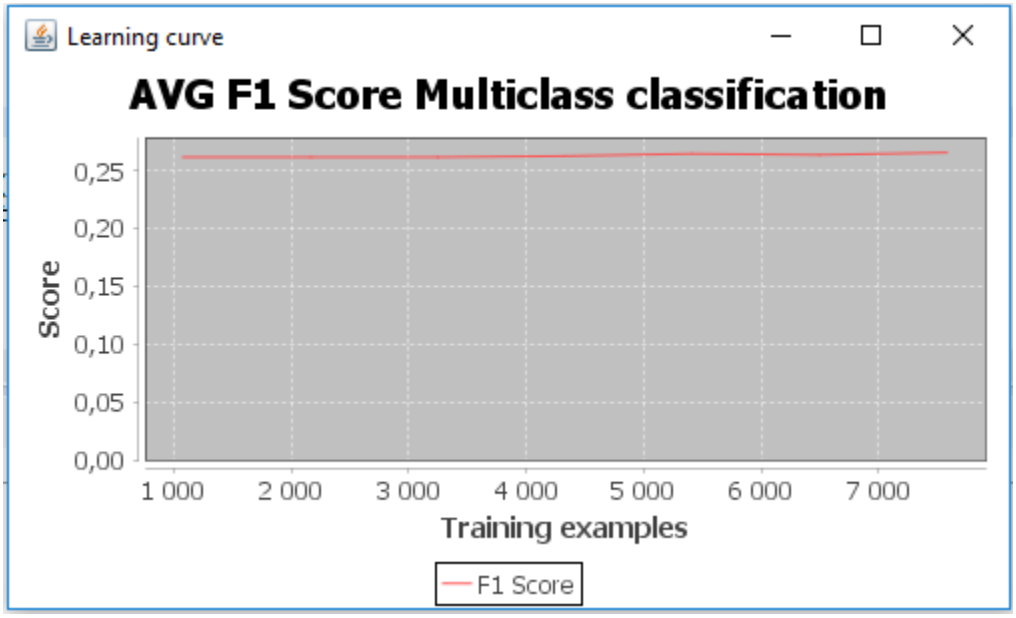
A.1 Utvikling med økt mengde treningsdata

Første treningssett som kun bestod av Premier League kamper hadde rundt 1680 treningseksempler. Det er en mulighet for at dette er et relativt lite sett, og for å få oversikt over hva som er en tilfredsstillende størrelse på treningssettet velger vi å dele settet opp i syv forskjellige størrelser og evaluerer det mot sett på 30 prosent av det totale treningssettet (rundt 860 prediksjoner). Når grafen begynner å flate ut eller falle vil det være tilstrekkelig med data. Å legge til data utover dette punktet vil kunne føre til overtilpasning. For å øke nøyaktigheten etter dette punktet vil en måtte ta inn andre variabler, og i noen tilfeller regulere hvordan variablene vektes.



Figur A.1: Læringskurve for multiclass classification, kun Premier League

Som vi kan lese av figur A.1 er det vanskelig å se noe klart mønster. Sportradar mangler detaljert informasjon om kamper før januar 2016, så vi vil videre prøve samme fremgangsmåte men ved å ta inn data fra tre andre ligaer; Serie A, La Liga og Bundesliga. Denne dataen vil sannsynligvis ikke gjøre modellen bedre til å se mønstre spesifikke lag har, men kan forbedre modellen mer generelt.



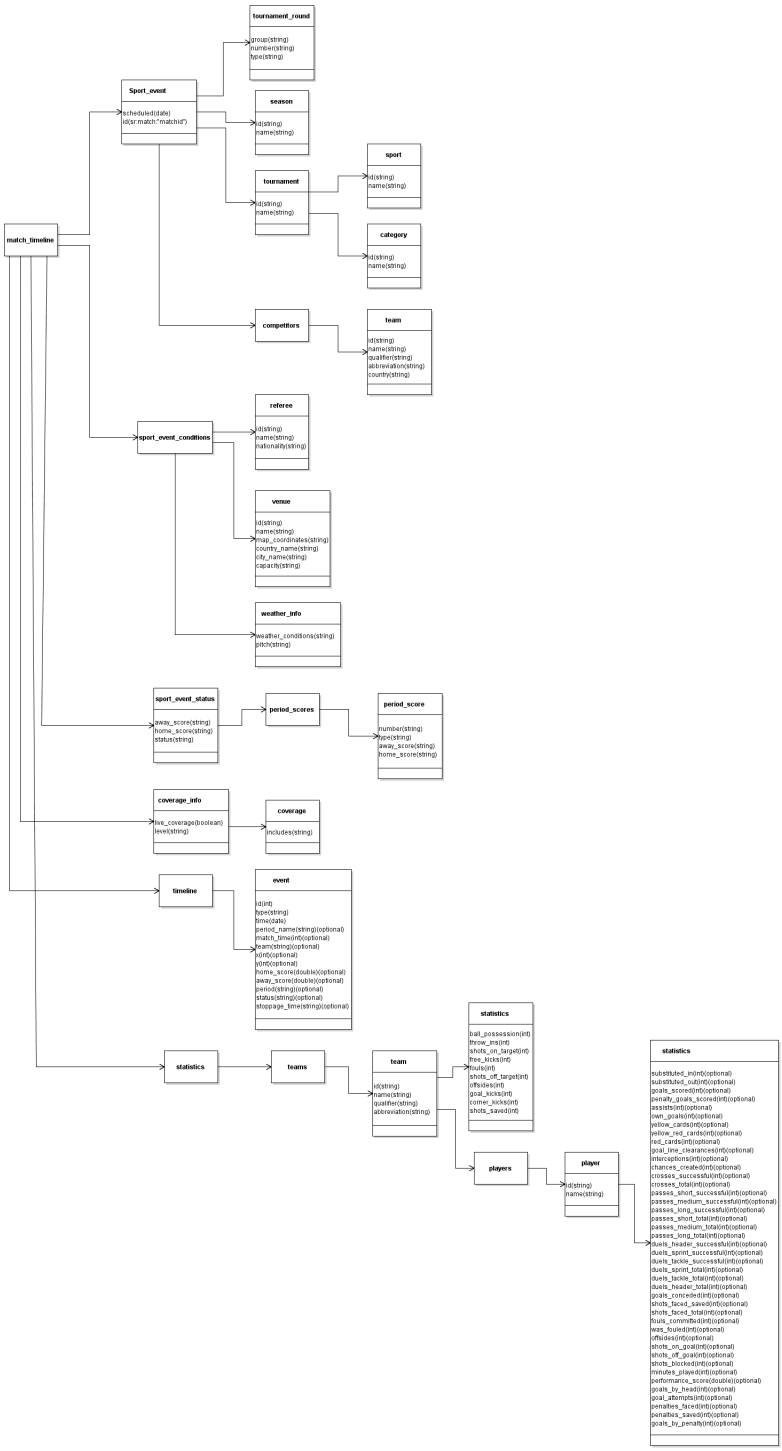
Figur A.2: Læringskurve for multiclass classification, 4 ligaer

Vi kan se av figur A.2 at resultatene ikke forandrer seg ved å legge til data fra andre ligaer, og at vi må øke antallet features fra kamper i Premier League for å forbedre modellene videre.

Tillegg B

Tilleggsinformasjon - ER Diagram

B.1 Fotball - timeline



Tillegg C

Tilleggsinformasjon - Statistikker

Her er det en oversikt over deskriptive statistikker for input data i de forskjellige utkastene av treningssettene. Dette er omtrentlige statistikker.

C.1 Statistikker

Attributes	Correlations to target	Unique values	Most frequent categories	Least frequent
away_team	0.01296	23	sr:competitor:7	sr:competitor:263
home_team	0.01028	23	sr:competitor:35	sr:competitor:39
match_time	0.00746	6	0	5
outcome	Not available	3	x	2

Figur C.1: Kategoriske attributter i første utkast

Attributes	Correlations to target *	Range	Mean	Median
away_score	0.00241	0 - 6	0.43691389599317987	0
home_score	0.00324	0 - 5	0.5946291560102301	0

Figur C.2: Numeriske attributter i første utkast

Attributes	Correlations to target	Unique values	Most frequent categories	Least frequent
away_team	0.01286	23	sr:competitor:48	sr:competitor:263
home_team	0.01013	23	sr:competitor:35	sr:competitor:39
match_time	0.00727	6	0	5
outcome	Not available	3	x	2

Figur C.3: Kategoriske attributter i andre utkast

Attributes	Correlations to target *	Range	Mean	Median
away_probability	0.05043	0.034 - 0.822	0.3207455470737913	0.291
away_score	0.00232	0 - 6	0.43808312128922816	0
home_probability	0.04775	0.075 - 0.917	0.459002544529262	0.449
home_score	0.00338	0 - 5	0.5932994062765055	0

Figur C.4: Numeriske attributter i andre utkast

Attributes	Correlations to target	Unique values	Most frequent categories	Least frequent
away_team	0.01298	23	sr:competitor:48	sr:competitor:263
home_team	0.01015	23	sr:competitor:35	sr:competitor:39
match_time	0.00751	6	1	0
outcome	Not available	3	x	2

Figur C.5: Kategoriske attributter i tredje utkast

Attributes	Correlations to target *	Range	Mean	Median
away_corners	0.00629	0 - 11	1.7961620469083155	1
away_freekicks	0.00687	0 - 18	4.1560767590618335	4
away_probability	0.0508	0.034 - 0.822	0.3199918976545842	0.29
away_red_cards	0.00378	0 - 1	0.006396588486140725	0
away_score	0.00252	0 - 6	0.4332622601279318	0
away_shots_off_target	0.00519	0 - 10	1.5560767590618336	1
away_shots_on_target	0.00232	0 - 10	1.3987206823027718	1
away_yellow_cards	0.00702	0 - 6	0.5377398720682303	0
home_corners	0.00467	0 - 16	2.295948827292111	2
home_freekicks	0.0059	0 - 19	4.484434968017058	4
home_probability	0.04825	0.075 - 0.917	0.45998678038379515	0.451
home_red_cards	0.00143	0 - 1	0.011940298507462687	0
home_score	0.00351	0 - 5	0.5940298507462687	0
home_shots_off_target	0.00341	0 - 12	1.6925373134328358	1
home_shots_on_target	0.00495	0 - 12	1.861407249466951	1
home_yellow_cards	0.00262	0 - 5	0.4933901918976546	0

« ‹

Figur C.6: Numeriske attributter i tredje utkast

Attributes	Correlations to target	Unique values	Most frequent categories	Least frequent
away_team	0.00842	89	sr:competitor:38	sr:competitor:2541
home_team	0.00779	89	sr:competitor:29	sr:competitor:2677
match_time	0.00411	6	0	5
outcome	Not available	3	x	2

Figur C.7: Kategoriske attributter i fjerde utkast

Attributes	Correlations to target *	Range	Mean	Median
away_corners	0.00403	0 - 13	1.7120288248337028	1
away_freekicks	0.0026	0 - 25	5.060883222468588	4
away_probability	0.01929	0.018 - 0.915	0.3158586474501108	0.28
away_red_cards	0.00247	0 - 2	0.013673318551367332	0
away_score	0.00359	0 - 6	0.43144863266814487	0
away_shots_off_target	0.00427	0 - 13	1.522450110864745	1
away_shots_on_target	0.00408	0 - 12	1.4475240206947524	1
away_yellow_cards	0.00339	0 - 7	0.6856984478935698	0
home_corners	0.00344	0 - 18	2.1867147080561713	2
home_freekicks	0.00217	0 - 24	5.215816703621582	5
home_probability	0.01885	0.038 - 0.97	0.468141352549889	0.465
home_red_cards	0.00264	0 - 2	0.011456023651145602	0
home_score	0.00183	0 - 7	0.6045824094604583	0
home_shots_off_target	0.0037	0 - 13	1.8895048041389504	1
home_shots_on_target	0.00361	0 - 17	1.900129342202513	1
home_yellow_cards	0.00197	0 - 6	0.58859940872136	0

Figur C.8: Numeriske attributter i fjerde utkast

Bibliografi

[1] Amazon. Multiclass model insights. "http://docs.aws.amazon.com/machine-learning/latest/dg/multiclass-model-insights.html".

[2] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classiers. 1992.

[3] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] Leo Breiman and Adele Cutler. Random forests. "https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm".

[5] Chao Chen, Andy Liaw, and Leo Breiman. Using random forest to learn imbalanced data.

[6] Sanku Dey and Enayetur Raheem. Multilevel multinomial logistic regression model for identifying factors associated with anemia in children 6–59 months in northeastern states of india. *Cogent Mathematics*, 2016.

[7] Nick Dingwall and Chris Potts. Are categorical variables getting lost in your random forests? "https://roamanalytics.com/2016/10/28/are-categorical-variables-getting-lost-in-your-random-forests/".

[8] Kai-Bo Duan, Jagath C. Rajapakse, and Minh N. Nguyen. One-versus-one and one-versus-all multiclass svm-rfe for gene selection in cancer classification. In *Evolutionary Computation,Machine Learning and Data Mining in Bioinformatics*. Springer, Berlin, Heidelberg, 2007.

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York, Stanford, California, 2nd edition, 2009.

[10] Tony Jebara. *Machine Learning Discriminative and Generative*. Springer US, 1st edition, 2004.

[11] Bing-Chen Jhong, Jhih-Huang Wang, and Gwo-Fong Lin. An integrated two-stage support vector machine approach to forecast inundation maps during typhoons. *Journal of Hydrology*, 2017.

[12] Matthew G. S. Kerr. Applying machine learning to event data in soccer. Master's thesis, Massachusetts Institute of Technology, 2015.

[13] Gunjan Kumar. Machine learning for soccer analytics. Master's thesis, KU Leuven, 2013.

[14] Bing Liu. *Web Data Mining Exploring Hyperlinks, Contents, and Usage Data*. Springer, 2nd edition, 2011.

[15] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Springer, 1st edition, 1998.

[16] David Meyer. Support vector machines the interface to libsvm in package e1071. 2017.

[17] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, Friedrich Leisch, Chih-Chung Chang, and Chih-Chen Lin. Misc functions of the department of statistics, probability theory group (formerly: E1071), tu wien. 2017.

[18] Kevin P. Murphy. *Machine Learning, A Probabilistic Perspective*. Massachusetts Institute of Technology, 2012.

[19] Y. Andrew Ng and I. Michael Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. 2002.

[20] Jean Francois Puget. The most popular language for machine learning is ... "https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Language_Is_Best_For_Machine_Learning_And_Data_Science?lang=en".

[21] Karthik Ramasubramanian and Abhishek Singh. *Machine Learning Using R*. Apress, 1st edition, 2017.

[22] Stuart J. Russel and Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, 3rd edition, 2010.

[23] J.R. Wilson and K.A Lorenz. *Modeling binary correlated responses using SAS, SPSS, and R*, chapter 2. Springer, 2015.

[24] Albina Yezus. Predicting outcome of soccer matches using machine learning. http://www.math.spbu.ru/SD_AIS/documents/2014-12-341/2014-12-tw-15.pdf, 2014.

[25] Huan Zhang, Pengbao Wu, Aijing Yin, Xiaohui Yang, Ming Zhang, and Chao Gao. Prediction of soil organic carbon in an intensively managed reclamation zone of eastern china: A comparison of multiple linear regressions and the random forest model. *Science of the Total Environment*, 2017.