INF161 - prosjekt

Sheldon Dyrdal

October 2024

1 Prosjektbeskrivelse

Målet med prosjektet er å lage en maskinlæringsmodell som predikerer lengden på sykehusopphold til en gitt pasient. Datasettet som brukes for å trene modellen inneholder info om 8261 kritisk syke pasienter. Dataene er fra periodene 1989-1991 og 1992-1994, og samlet inn fra 5 medisinske sentre i USA.

2 Metode

2.1 Dataforberedelse

Datasettene for prosjektet er delt inn i *raw_data*, som brukes for å trene opp modeller, og *sample_data*, datasettet som predikeres på. Hvert datasett er videre delt inn i fire datasett: informasjon om sykdomsalvorlighet, sykehusdata, fysiologiske data og demografiske data. Henholdsvis leses disse inn som en $pandas.DataFrame$. For å rense dataene og gjøre dem mer strukturerte, lages flere hjelpefunksjoner: *clean_demographic_data*, *clean_severity_data*, *clean_physiological_data* og *clean_hospital_data*. Felles for disse hjelpefunksjonene er at de håndterer både duplikater og "umulige verdier".

2.1.1 Håndtering av "umulige verdier"

Flere av datasettene inneholder numeriske variabler hvor det ikke er mulig å ha negative verdier, f.eks. alder. Hjelpefunksjonene for rensing av datasettene håndterer dette ved å erstatte verdiene med NaN-verdier (Not a Number). Altså gjøres umulige verdier om til manglende verdier.

2.1.2 Fjerning av data

I demografisk data viste det seg at det var to duplikater av samme pasient, hvor disse ble fjernet. I tillegg blir duplikater kontrollert og fjernet i alle øvrige datasett. I datasettet om sykdomsalvorlighet fjernes *sykdomskategori_id* og *sykdom_kategori* siden de gir samme informasjon som *sykdom_underkategori*, som beholdes.

Målet er å predikere på nye pasienter ut fra data som er tilgjengelig fra dag 1. Det gjør at data som ikke er tilgjengelig fra dag 1, ikke er hensiktsmessig å ta med for å trene opp en modell. Derfor fjernes variablene "*adl_pasient*"og "*bilirubin*", da disse er tilgjengelig fra dag 7. I tillegg ble "$dødsfall$"og "*sykehusdød*fjernet, da det ikke gir mening å ha den informasjonen fra dag 1 når man skal predikere fremtidig oppholdslengde.

2.1.3 Info om sykdomsalvorlighet

Datasettet om sykdomsalvorlighet, $severity.json$, er indeksert etter *sykdomskategori_id*. Dette gjør at datasettet opprinnelig har 4 rader og er formet som et nestet"oppslagsverk. Vi er interessert i å se på data for hver "*pasient_id*". For å indeksere datasettet etter *pasient_id*, må man

bruke *explode*-metoden. Et problem som oppstår da, er at man må manuelt gjøre om numeriske variabler til typen $float64$ eller $int64$. Dette løses ved å bruke *pandas.to_numeric*.

2.1.4 Sammenslåing av datasett

Datasettene settes sammen ved hjelp av *pandas.merge*-metoden, hvor de slås sammen etter "*pasient_id*". Det implementeres også en *assert*-test for å sjekke at det ikke fins flere rader med samme "*pasient_id*". Etter man har slått sammen til en felles *pandas.DataFrame*, kan *pasient_id* fjernes, da den ikke vil ha noe direkte innvirkning på oppholdslengde og man ikke vil at modellen skal trene på id-er. I tillegg fjernes alle rader hvor "*oppholdslengde*"mangler, da dette er målvariabelen, men dette gjøres ikke på *sample_data*, da den ikke inneholder *oppholdslengde* fordi det er det man skal predikere til slutt. Videre nå skal det kun tas utgangspunkt i *raw_data* hvor man kommer tilbake til *sample_data* i kap. 2.6.

2.1.5 train_test_split

Ved datamodellering er det viktig å unngå datalekkasje. En typisk feil man kan gjøre da er å trene på data som man i framtiden skal predikere på. Dette gjør at modellen er 'biased' og man ikke får en ren objektiv vurdering av hvor god modellen er [7]. Derfor er den anbefalte metoden for modellutvalg og evaluering å dele datasettet i tre deler: trening-, validerings- og testdata. For utforskende dataanalyse og trening (fit) av modeller, brukes treningsdata. Valideringsdata brukes for å velge ut beste modell, hyperparametre og generelt hvilke metoder for dataforberedelse og variabelutvinning som er best ut fra et gitt ytelsesmål. Til slutt brukes testdata for å gi en objektiv evaluering av den beste modellen. Med denne metoden unngår man nettopp å predikere på data som modellen har sett fra før. For å gjøre dette i praksis brukes *sklearn.model_selection.train_test_split*.

Aller først splitter man opp datasettet for uavhengige variabler, X, og målvariabelen, y. Hvor målvariabelen y er oppholdslengde og X er alle andre variabler som skal brukes for å predikere y. Det er X og y som brukes som input i *train_test_split*. For reproduserbarhet, brukes $random\_state = 42$ som argument. Hvis $random\_state = None$, som er standard, vil den ikke splittes likt hver gang man kjører koden [6]. For prosjektet valgte jeg en fordeling av trenings-, validerings- og testdata til å være henholdsvis 70%, 15% og 15% av hele datasettet.

2.2 Utforskende dataanalyse

Som tidligere nevnt, gjøres all utforskende dataanalyse på treningsdata. Her prøver man å undersøke og forstå eventuelle sammenhenger i datasettet.

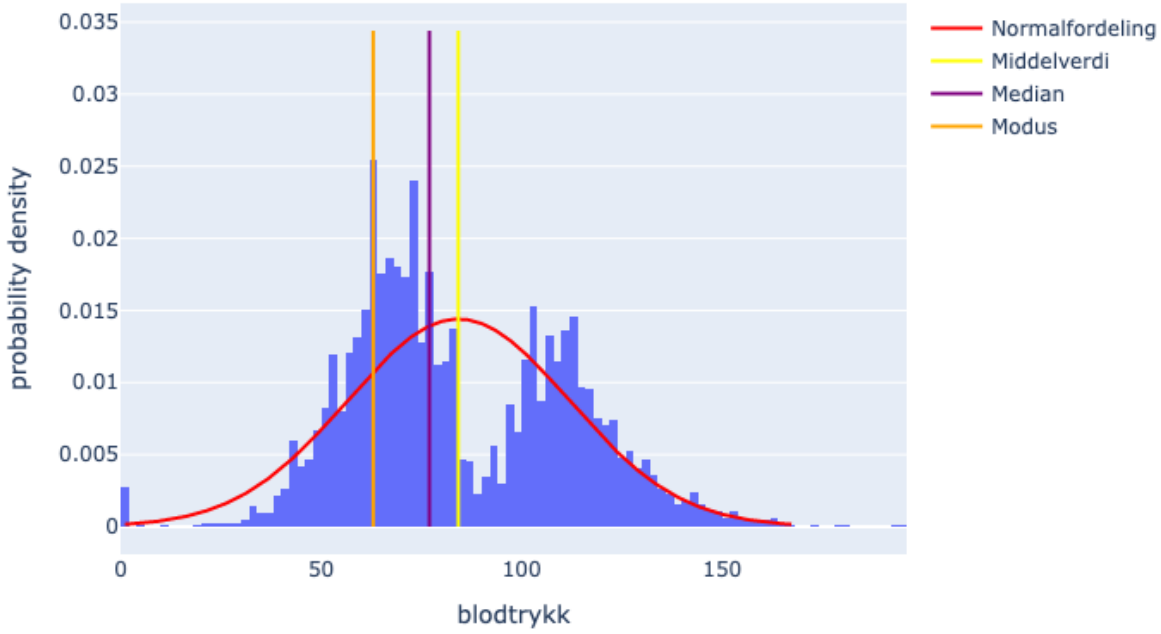
2.2.1 Tetthetsfunksjoner

Til å begynne med benyttes *pandas.DataFrame.describe*-metoden for å få en oversikt over de ulike variablene i datasettet, inkludert gjennomsnitt, minimums- og maksimumsverdier, standardavvik samt 25%, 50% og 75% kvantiler, som vist i figur 1. Her indikerer 25% kvantilen at 25 % av dataene er lavere enn denne verdien, 50% kvantilen viser at halvparten av dataene ligger mellom 25% og 75% kvantilen, mens 75% kvantilen viser at 25% av dataene ligger over 75% kvantilen.

	alder	utdanning	blodtrykk	hvite_blodlegemer	hjertefrekvens	respirasjonsfrekvens	kroppstemperatur	lungefunksjon	serumalbumin
count	5408.000000	4435.000000	5413.000000	5281.000000	5413.000000	5413.000000	5413.000000	4040.000000	3421.000000
mean	62.725937	11.802480	84.210881	12.411832	97.694624	23.482912	37.145275	237.822606	2.941227
std	15.637994	3.421883	27.700134	8.989788	31.487143	9.639604	1.264891	109.478257	0.924940
min	18.118990	0.000000	0.000000	0.000000	0.000000	0.000000	31.699220	12.000000	0.399963
25%	52.863470	10.000000	63.000000	7.000000	72.000000	18.000000	36.195310	154.468750	2.399902
50%	65.015990	12.000000	77.000000	10.699219	100.000000	24.000000	36.796880	222.500000	2.899902
75%	74.116718	14.000000	107.000000	15.398438	120.000000	28.000000	38.195310	304.000000	3.599609
max	100.849000	31.000000	195.000000	113.593750	232.000000	90.000000	41.695310	869.375000	29.000000

Figur 1: Utdrag av describe-metoden på treningsdata

Ved å ta utgangspunkt i verdier fra *describe*, så visualiseres noen variabler som kan være interessante. Variablene som er valgt å se på er *alder*, *blodtrykk*, $hjertefrekvens$ og $fysiologisk\_score$. I tillegg ser man på målvariabelen, oppholdslengde. Her ser man på tetthetsfunksjonen til hver variabel og viser i tillegg middelverdi, median og modus. Et eksempel ser man på figur 2.



Figur 2: Tetthetsfunksjonen til $fysiologisk\_score$

Hvor gitt data X_i fra 0 til n-datapunkter, så er middelverdi $\frac{1}{n}\sum_{i=0}^{n} X_i$ (gjennomsnittet), median er den midterste verdien etter alle verdier er sortert etter stigende rekkefølge, og modusen er verdien som hyppigst forekommer i X_i.

2.2.2 Manglende data

Det kan være flere årsaker til hvorfor det mangler data for ulike variabler. Likevel så er det like problematisk at det mangler. For å få innsikt i manglende data, kan man bruke $pandas.DataFrame.info$-metoden. Den sier noe om hvilke type en variabel har, men også hvor mange ikke-null verdier den inneholder. Deretter regnes det ut hvor mange NaN-verdier og prosentandel av manglende

data for hver variabel. Resultatet settes i en *pandas.DataFrame* som vist i figur 6. Håndtering av manglende data skal diskuteres senere under kap. 2.3.

2.2.3 Korrelasjon

Korrelasjon beskriver den lineære sammenhengen mellom to variabler. Derfor er det viktig å sikre at alle variabler er numeriske, siden analysen fokuserer på lineære forhold. Metoden for å gjøre kategoriske variabler om til numeriske forklares under 2.3.5.

Positiv korrelasjon oppstår når to variabler øker eller reduseres sammen. Negativ korrelasjon betyr at når en variabel øker, reduseres den andre. Verdier for korrelasjon kan dessuten variere mellom -1 og 1. En variabel som er sterkt korrelert til målvariabelen vil være nyttig å finne. For å finne korrelasjonen med oppholdslengde brukes *pandas.DataFrame.corrwith*-metoden mot målvariabelen, y (y_train). Dette visualiseres som vist på figur 8.

Sterkt korrelerte variabler utenom målvariabelen, er også nyttig å finne. Hvis to variabler er perfekt korrelert, enten -1 eller 1, vil det da være nok å se på kun den ene. Dette gjør at man kan fjerne variabler hvis dette er tilfelle, noe som vil være nyttig for å forenkle datasettet. For å finne korrelasjonen mellom ulike variabler bruker man *pandas.DataFrame.corr*.

2.3 Feature extraction (variabelutvinning) og håndtering av manglende data

På lik linje som utforskende datanalyse, må valg som tas for variabelutvinning og håndtering av manglende data gjøres ut fra det man vet om treningsdata. Likevel skal metodene implementeres på valideringsdata og testdata. I tillegg vil evaluering av metodene gjøres ut fra resultatet man får på valideringsdata. Dette er essensielt for å unngå datalekkasje og få en objektiv vurdering av metodene.

2.3.1 Fjerne variabler med mye manglende data

Hvis det mangler mye data, kan det være lurt å ignorere dem. Det vil si at man fjerner de variablene det gjelder. Ulempen er at man kan risikere å miste mye data, men da man er avhengig av at det ikke mangler data, kan det derimot være problematisk å imputere dem. Terskelen settes til at alle variablene som mangler mer enn 30% i treningsdata fjernes fra hele datasettet (trenings-, valideringsdata og testdata). Dette gjøres ved hjelp av funksjonen *remove_cols*.

2.3.2 Imputering

I tillegg til at man kan ignorere manglende data, kan man også erstatte dem med nye verdier. Denne prosessen kalles å imputere manglende data, eller imputasjon. Da kan man for eksempel bruke *sklearn.impute.SimpleImputer* eller *sklearn.impute.KNNImputer*. Dette er strategier som henholdsvis gjør at man imputerer med median/middelverdi/modus eller ved bruk av en k-nærmeste nabo-regresjonsmodell [6]. Ved datamodellering er man ikke interessert i hvilken imputeringsstrategi som gjør det best alene, men heller hvilken kombinasjon av imputeringsstrategi og regresjonsmodell som gir best resultat på valideringsdata. Likevel kan det være verdt å merke seg forskjellene for de ulike strategiene, noe som vises på figur 7.

2.3.3 National Early Warning Score

National Early Warning Score 2 (NEWS2) er et skåringssystem som brukes til å overvåke vitale funksjoner hos syke pasienter. Poengene tildeles basert på målinger fra en rekke vitale parametere og er vist i figur 3. En høy NEWS-score indikerer en alvorlig sykdomstilstand, noe som kan kreve umiddelbar klinisk intervensjon [3]. Systemet vurderer funksjoner som respirasjonsfrekvens, blodtrykk, hjertefrekvens og kroppstemperatur. Disse variablene er tilgjengelige i datasettet, og ved å anvende NEWS2 kan vi beregne en score for hver pasient. Dette gjøres ved hjelp av funksjonen *calculate_news_score* og *apply_news_score*, hvor sistnevnte funksjon lager en ny kolonne med utregnet NEWS-score fra førstnevnte funksjon.

Chart 1: The NEWS scoring system

Physiological parameter	Score						
	3	2	1	0	1	2	3
Respiration rate (per minute)	≤8		9–11	12–20		21–24	≥25
SpO_2 Scale 1 (%)	≤91	92–93	94–95	≥96			
SpO_2 Scale 2 (%)	≤83	84–85	86–87	88–92 ≥93 on air	93–94 on oxygen	95–96 on oxygen	≥97 on oxygen
Air or oxygen?		Oxygen		Air			
Systolic blood pressure (mmHg)	≤90	91–100	101–110	111–219			≥220
Pulse (per minute)	≤40		41–50	51–90	91–110	111–130	≥131
Consciousness				Alert			CVPU
Temperature (°C)	≤35.0		35.1–36.0	36.1–38.0	38.1–39.0	≥39.1	

Figur 3: NEWS2 skåringsskjema

2.3.4 Fra numerisk til kategorisk: dele inn i intervaller

For å forenkle analysen av noen variabler, deles de inn i intervall for å skille mellom 'unormale' og 'normale' verdier. Dette gjøres på *blodtrykk*, $respirasjonsfrekvens$ og $lungefunksjon$. Intervallene bestemmes ut fra domenekunnskap om hva som er 'normalt' og'ikke 'normalt' [1][2][3]. Dette gjøres ved hjelp av funksjonen *categorize_values*. Da lages det nye variabler *blodtrykk_range*, $respirasjonsfrekvens\_range$ og $lungefunksjon\_range$, og *blodtrykk*, $respirasjonsfrekvens$ og $lungefunksjon$ fjernes. Funksjonen $transform\_X$ er definert slik at den kaller på *calculate_news_score*, *apply_news_score* og *categorize_values* for å forenkle prosessen.

2.3.5 Fra kategorisk til numerisk: OneHotEncoder

Ved regresjon ser man etter sammenheng mellom numeriske variabler. Derfor må vi først gjøre kategoriske variabler om til numeriske, såkalte $dummy - variabler$. Dette gjøres ved hjelp av $sklearn.preprocessing.OneHotEncoder$. Man kunne også brukt $pandas.get\_dummies$, men $OneHotEncoder$ er mer fleksibel da den er i stand til å gjøre ting likt hver gang den transformerer et datasett, selv når det dukker opp nye, usette variabler. I det tilfellet hvor det dukker opp usette variabler, settes $handle\_unknown =' ignore'$ hvor de resulterende nye kolonnene vil alle få verdiene 0 [6]. I tillegg brukes $sklearn.compose.ColumnTransformer$ slik at OneHotEncoder kun kjøres på kategoriske kolonner. Dessuten legges det til prefiksene 'cat' og 'num' for henholdsvis kategoriske og numeriske variabler. Senere ved modellutvalg implementeres også imputasjonsstrategier i $ColumnTransformer$.

2.3.6 Variabelutvinning med korrelasjon i betraktning

En ulempe med å lage $dummy-variabler$ er at antallet kolonner kan øke betydelig. Dette krever en nøye analyse for å vurdere om noen av variablene kan utelates. Ved å undersøke korrelasjonen mellom variablene, kan de med høy korrelasjon bli identifisert for fjerning. I tilfelle av perfekt korrelasjon vil én av de to kolonnene bli eliminert, mens den andre beholdes. Et eksempel på dette er $kjønn$ som resulterer i nye kolonner $cat\_\_kjønn\_female$ og $cat\_\_kjønn\_male$, som er perfekt korrelert. I tillegg vurderes variabler med høy korrelasjon, og produktet av disse blir utregnet. Dette er et eksempel på 'feature interaction'. De opprinnelige variablene fjernes, men deres 'feature interaction' beholdes. For å implementere dette i praksis benyttes funksjonen $drop\_high\_corr$.

2.4 Datamodellering

De ulike modellene som testes ut, er hentet fra $Scikit-learn$. For å sette sammen metoder for dataforberedelse og regresjonsmodell, brukes $sklearn.pipeline.Pipeline$. Dette er en klasse som gjør det mulig å utføre dataforberedelse og deretter implementere en regresjonsmodell i ett sammenhengende steg. Man kan bruke fit, $transform$, og $predict$ slik at alle stegene skjer samtidig for et gitt datasett. Ytelsen til modellene evalueres med å sammenligne predikert verdier med faktisk verdier. Dette gjøres ved hjelp av å regne ut rot-middel-kvadrat-feil (RMSE) gitt ved likning 1:

$$\sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}} \tag{1}$$

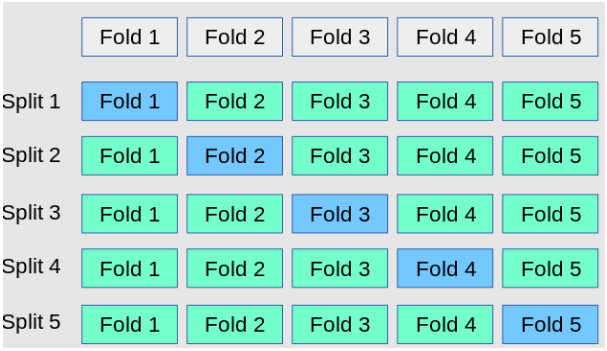
Hvor N er antall prediksjoner, $\hat{y}_i$ er den i-te predikerte verdien og y_i er tilsvarende faktiske verdi.

2.4.1 Grunnlinjemodell

For å sammenligne ulike modeller er det lurt å lage en grunnlinjemodell. I dette tilfellet brukes $sklearn.dummy.DummyRegressor$ med $strategy = "mean"$, som beregner middelverdien av treningsdataen, y_train, og returnerer denne verdien for hver prediksjon. Dette er ikke en god modell, men den gir en pekepinn på hvor dårlig en modell kan være. Siden det ikke er et reelt forsøk på å lage en modell, velges det heller ikke å sette den inn i en pipeline for å vurdere effekten av dataforberedelse.

2.4.2 Modellutvalg

For å teste ut ulike kombinasjoner av dataforberedelsesmetoder og regresjonsmodeller brukes $sklearn.model\_selection.RandomizedSearchCV$. Dette er en klasse som utfører n iterasjoner av forskjellige kombinasjoner av hyperparametre for en gitt $pipeline/modell$. Den har innebygde metoder for fit og $score$. I praksis kjører den sin egen $train\_test\_split$ for å unngå datalekasje, som vist i figur 4. Den benytter kryssvalidering (CV) som splitter dataene den trener på i k-deler (k-folds). Modellen trener på $k-1$ deler og predikerer på den gjenværende delen [5][8]. Dette er en stor fordel, fordi den predikerer på data som modellen ikke har trent på. Nettopp på grunn av at RandomizedSearchCV benytter kryssvalidering, kan man bruke et datasett som inkluderer både trenings- og valideringsdata uten noe datalekasje. Dette gir mening å gjøre fordi valideringsdata brukes til å velge den beste modellen og de beste hyperparametrene. I tillegg er det en fordel å ha mer data tilgjengelig for trening.



Figur 4: K-fold kryssvalidering. For hver *split*, så indikerer grønt område datapunktene brukt til å trene modellen, mens det blå området viser datapunktene som brukes til prediksjon.

For hver kombinasjon av hyperparametre beregnes middelverdien (*mean_test_score*) av alle *splits*, og kombinasjonene rangeres deretter. Disse resultatene kan hentes med *cv_results_*-metoden. Skåringsmetoden er satt til å være rot-middel-kvadrat-feil (RMSE). Dessuten er n iterasjoner satt til å være 10, og $n\_jobs = -1$ slik at alle tilgjengelige kjerner i datamaskinen brukes når den kjøres for å minimere kjøretid. For å hente ut den beste modellen fra $RandomizedSearchCV$ etter at fit-metoden er kjørt, bruker man *best_estimator_*-metoden.

De ulike regresjonsmodellene som testes er $LinearRegression$ (lineær regresjon), $ElasticNet$, $RandomForestRegressor$ og $GradientBoostingRegressor$. Det testes også etter ulike imputasjonsstrategier på numeriske variabler som er nevnt under 2.3.2, mens for kategoriske variabler så imputeres det etter verdien som oppstår flest ganger (modus). For numeriske variabler skaleres de også ved hjelp av $sklearn.preprocessing.StandardScaler$. Dette er en klasse som skalerer slik at for hver variabel så vil middelverdi konverge mot 0 og standardavvik mot 1. I tillegg testes det for både med og uten funksjonene $transform\_X$ og *drop_high_corr* for å sjekke om disse gir bedre resultater. Disse funksjonene implementeres i pipelinen ved hjelp av $sklearn.preprocessing.FunctionTransformer$[4].

Pipelinen med lavest RMSE, velges som beste modell (*best_model*). Til slutt sjekker man generaliseringsevnen til den beste modellen ved å predikere på testdata, *X_train*. Etter at man har valgt beste modell, trenger man ikke lenger inndelingen fra *train_test_split* og modellen trenes på nytt igjen på hele datasettet fra *raw_data*. Det vil likevel ikke skje noe datalekasje siden målet nå er å predikere på nye, usette data fra enten nettsiden eller *sample_data*.

2.5 Implementering av modellen til en nettside

Modulen *pickle* brukes for å lagre modellen i en fil, *model.pkl*. I tillegg benyttes *HTML Forms*, i filen *index.html*, for å lage en enkel nettside med et skjema. Deretter bruker man filen *app.py* og $flask$-modulen for å oppdatere nettsiden basert på interaksjon fra brukeren.

2.5.1 index.html

For å sende dataen fra nettsiden til *app.py* brukes *post*-metoden. Dessuten består nettsiden av flere 'inputs', som tar inn tekst og brukes for numeriske variabler, og flere rullegardinmenyer som brukes for kategoriske variabler.

2.5.2 app.py

Skjemaet som sendes fra nettsiden til *app.py* kan hentes ut som i ett *oppslagsverk*. Videre sjekkes oppslagsverket for ugyldig data. Dette gjøres ved hjelp av funksjonen *to_numeric*. Kategoriske

står uendret fordi *OneHotEncoder* håndterer selv disse ugyldige dataene. For numeriske gjøres de om til typene *float* eller *int*. Dette gjøres i en *try* og *except* blokk, slik at hvis ikke det kan typekonverteres, så returnerer den feilen til nettsiden slik at brukeren selv kan rette oppi feilen. Dette kan skje ved at brukeren skriver inn negative verdier eller skriver tekst hvor nettsiden forventer positive tall. Hvis brukeren svarer med bare gyldige data, vil nettsiden bli oppdatert med en prediksjon på sykehusopphold. Dette gjøres da ved hjelp av modellen fra *model.pkl*.

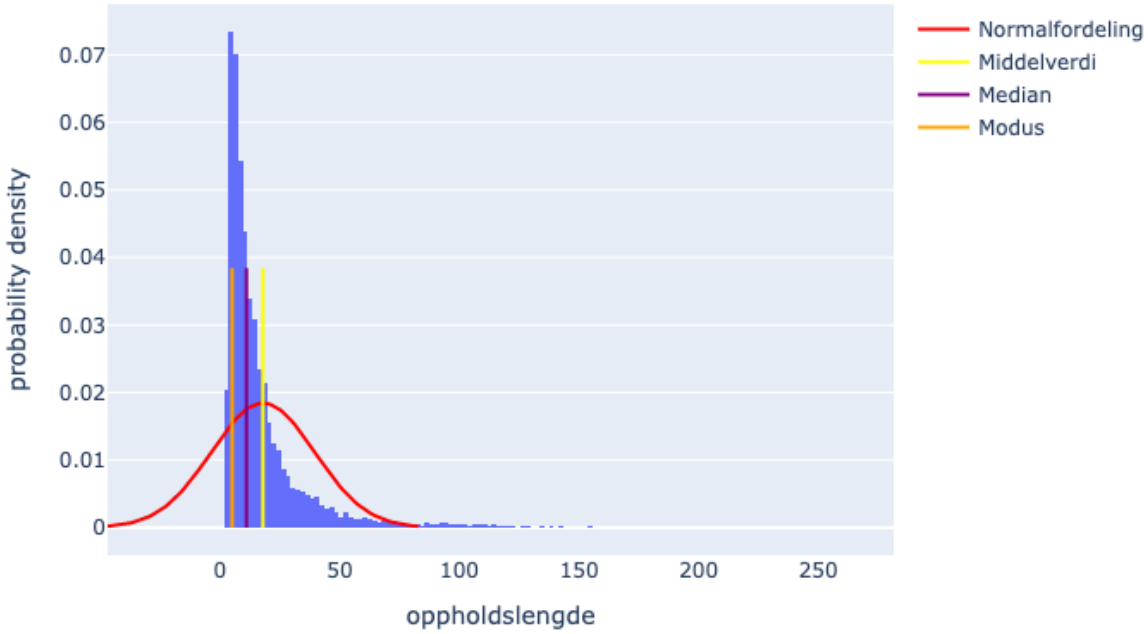
2.6 Predikere på nytt datasett

Til slutt skal det predikeres på de nye datapunktene fra *sample_data*. Som nevnt tidligere er den delt inn på lik måte som *raw_data*. De renses på lik måte som forklart under kap. 2.1 bortsett fra *train_test_split*. Det som skiller seg mellom datasettene er at *sample_data* er et betydelig mindre datasett og inneholder heller ikke *oppholdslengde*. Det betyr at man kan predikere oppholdslengde, men man kan ikke evaluere resultatet da man ikke har noen faktiske verdier å sammenligne med. Likevel sjekkes det for at det ikke predikeres noen negative verdier. Resultatet eksporteres til *predictions.csv* som består av kolonnene *pasient_id* og *predikert_sykehusopphold*.

3 Resultat

3.1 Tetthetsfunksjonener

Av tetthetsfunksjonene som ble forklart i kap. 2.2.1, ble også tetthetsfunksjonen til oppholdslengde undersøkt. Resultatet vises på figur 5.



Figur 5: Tetthetsfunksjonen til målvariabelen, oppholdslengde

Ut fra figuren ser man at oppholdslengde er betydelig '*skewed*'. Det vil si at fordelingen er skjev hvor modus, median og middelverdi er forskjellige. For de andre variablene som ble testet, er faktisk alle til en grad '*skewed*'.

3.2 Manglende data fra treningsdata

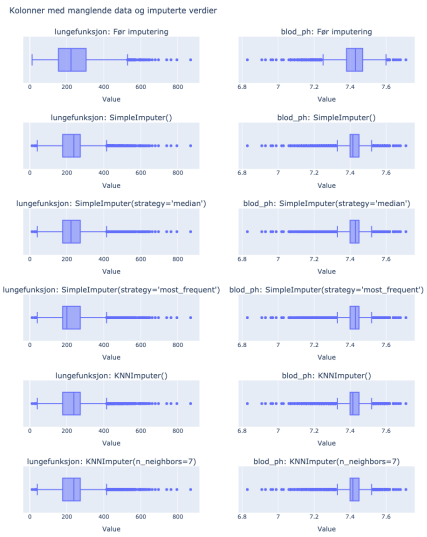
Resultatet av manglende data fra treningsdata, X_train, er vist på figur 6. Terskelen for manglende data ble satt til 30% som resulterer i at kolonnene som ble fjernet fra hele datasettet er *dnr_dag*, *dnr_status*, *urinmengde*, *glukose*, *blodurea_nitrogen*, *serumalbumin*, *inntekt* og *adl_stedfortreder*.

	Antall manglende verdier	Prosent manglende verdier
dnr_dag	4657	86.033623
dnr_status	4657	86.033623
urinmengde	2864	52.909662
glukose	2660	49.140957
blodurea_nitrogen	2568	47.441345
serumalbumin	1992	36.800296
inntekt	1793	33.123961
adl_stedfortreder	1714	31.664511
lungefunksjon	1373	25.364862
blod_ph	1354	25.013856
lege_overlevelsesestimat_2mnd	984	18.178459
utdanning	978	18.067615
lege_overlevelsesestimat_6mnd	972	17.956771
hvite_blodlegemer	132	2.438574
kreatinin	42	0.775910
etnisitet	31	0.572695
alder	5	0.092370

Figur 6: Resultat av utregning av manglende verdier

3.3 Effekten av ulike imputeringsstrategi

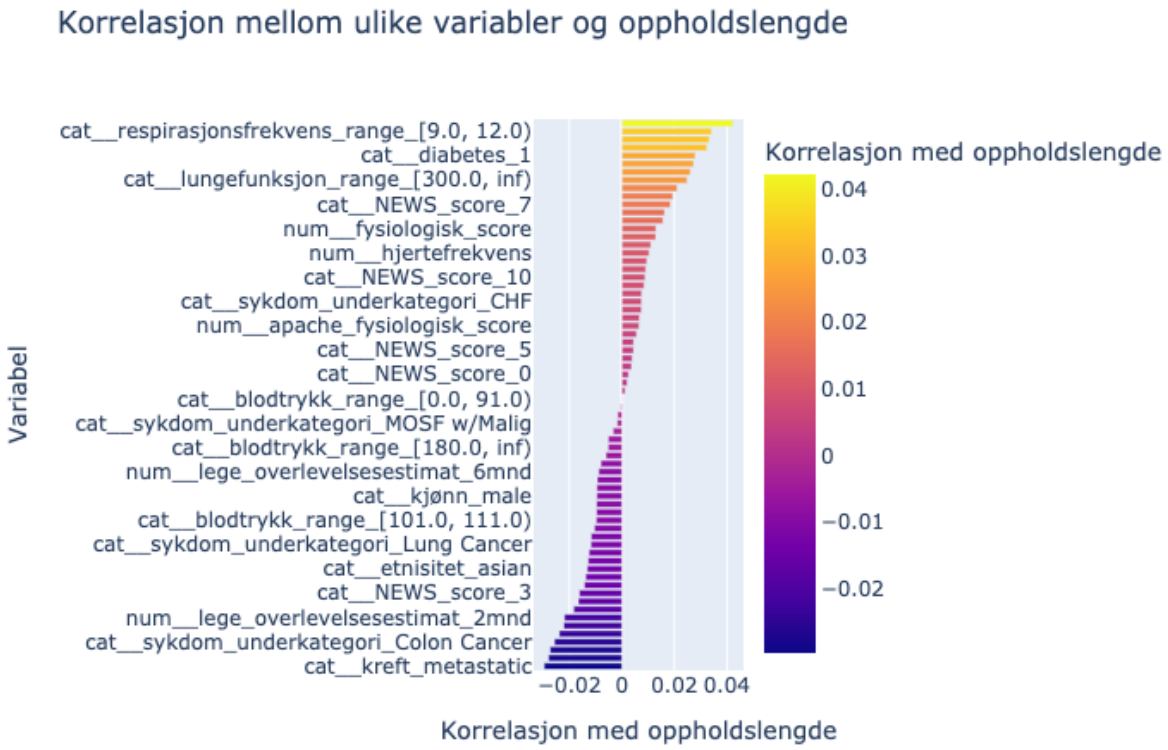
Etter man har fjernet data, som forklart i kap. 3.2, sjekkes effekten av ulike imputeringsstrategier. Variablene som undersøkes er de to med mest manglende data, *lungefunksjon* og *blod_ph*, som vist på figur 7.



Figur 7: Sammenligning av ulike imputeringsstrategier

3.4 Resultat av korrelasjon

Ved korrelasjon mellom ulike variabler og oppholdslengden var den høyeste korrelasjonen på 0,04, og den laveste var på -0,03. Dette vises på figur 8.



Figur 8: Korrelasjon mellom ulike variabler og oppholdslengde

Det ble også regnet ut korrelasjon mellom ulike variabler utenom målvariabelen. De mest interessante var dem med absolutt korrelasjon høyere enn 0,7. Resultatet av variablene det gjelder vises på figur 9. Under kap. 2.3.6, ble det nevnt å regne ut produktet mellom noen variabler med absolutt korrelasjon høyere enn 0,7. Variablene dette gjelder er *num__apache_fysiologisk_score* med *num__fysiologisk_score*, *num__lege_overlevelsesestimat_2mnd* med *num__lege_overlevelsesestimat_6mnd*, og *num__overlevelsesestimat_2mnd* med *num__overlevelsesestimat_6mnd*.

	Kolonne1	Kolonne2	Korrelasjon
771	cat__kjønn_female	cat__kjønn_male	-1.000000
1539	cat__demens_1	cat__demens_0	-1.000000
1399	cat__diabetes_1	cat__diabetes_0	-1.000000
983	cat__etnisitet_black	cat__etnisitet_white	-0.823050
4548	num__overlevelsesestimat_2mnd	num__fysiologisk_score	-0.753200
561	cat__kreft_metastatic	cat__kreft_no	-0.703161
4411	num__fysiologisk_score	num__apache_fysiologisk_score	0.798670
4691	num__lege_overlevelsesestimat_2mnd	num__lege_overlevelsesestimat_6mnd	0.897192
4551	num__overlevelsesestimat_2mnd	num__overlevelsesestimat_6mnd	0.960550

Figur 9: Variabler med absolutt korrelasjon høyere enn 0.7

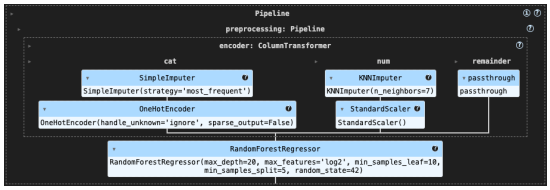
3.5 Beste modellen

Ut fra *RandomizedSearchCV* fra hver modell ble den beste modellen og dens tilsvarende RMSE hentet ut og lagret i en *DataFrame* som vist på figur 10.

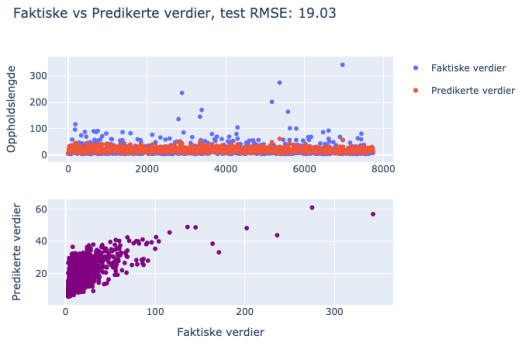
	Navn	Modell	rmse
0	RandomForestRegressor	((ColumnTransformer(remainder='passthrough',\n...	19.584668
1	GradientBoostingRegressor	((ColumnTransformer(remainder='passthrough',\n...	19.584668
2	LinearRegression	((ColumnTransformer(remainder='passthrough',\n...	19.856554
3	ElasticNet	((ColumnTransformer(remainder='passthrough',\n...	19.905218
4	DummyRegressor	DummyRegressor()	20.603791

Figur 10: Oversikt over de beste modellene fra *RandomizedSearchCV*

Resultatet ble at *RandomForestRegressor* gjorde det best med en avrundet RMSE på 19,6. Kombinasjonen av hyperparametre, imputeringsstrategier og andre dataforberedelsesmetoder vises på figur 11(a). Faktisk var den beste modellen uten funksjonene $transform\_X$ og $drop\_high\_corr$. For å vurdere generaliseringsevnen, predikerte man på testdata, X_train, og regnet ut RMSE på 19,0. Prediksjonene visualiseres på figur 11(b).



(a) Pipeline for beste modell

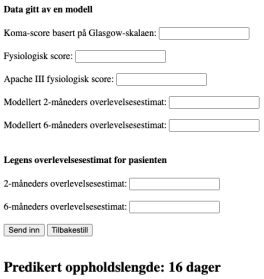


(b) Visualisering av generaliseringsevnen til den beste modellen

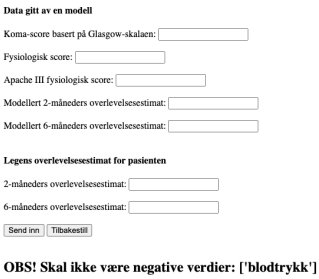
Figur 11: Beste modell: RandomForestRegressor

3.5.1 Nettside

Ved metodene forklart under kap. 2.5 resulterte det i en nettside som kan predikere på data fra brukeren, og den er i stand til å håndtere ugyldige data som henholdsvis er vist på figur 12(a) og 12(b).



(a) Gyldig data

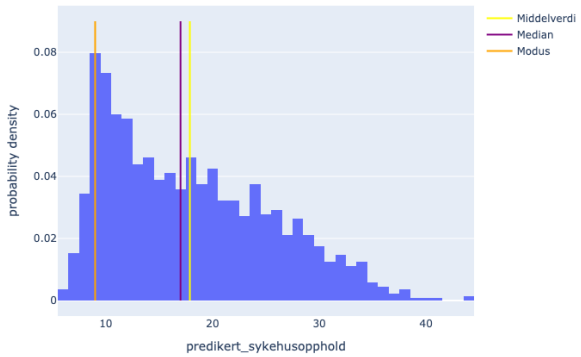


(b) Ugyldig data

Figur 12: Skjermbilder av nettsiden.

3.5.2 Prediksjon på sample_data

Resultatene for predikerte oppholdslengde på *sample_data* finnes i filen *predictions.csv*. Dens tilsvarende tetthetsfunksjon vises på figur 13.



Figur 13: Tetthetsfunksjonen til predikert oppholdslengde på *sample_data*

4 Diskusjon

Hypotesen min før datamodellering var at variabelutvinningfunksjonene $transform\_X$ og $drop\_high\_corr$ skulle gi et bedre resultat. Derimot skjedde det motsatte, hvor funksjonene ga dårligere resultat, og den beste modellen var den uten disse funksjonene i pipelinen. Årsaken til dette kan være at under nærmere undersøkelse på korrelasjon, så ga faktisk de nye kolonnene relativt dårlig korrelasjon med oppholdslengden. I tillegg kan det være at funksjonene gjorde datasettet for komplekst, da disse funksjonene opprettet betydelig flere kolonner etter at man hadde brukt $OneHotEncoder$.

Dessuten er RMSE på valideringss- og testdata relativt nær hverandre med henholdsvis verdier på 19,6 og 19,0, altså en forskjell på 0,3. Dette tyder på at modellen greier å generalisere godt, da den undertilpasser i liten grad. Dette kan indikere at predikerte verdier vil i snitt avvike med omtrent ± 19 fra de faktiske verdiene. I den virkelige verden ville dette vært et for stort avvik for å implementere på sykehuset. Faktisk kan man risikere å predikere feil på nesten tre uker. I tillegg ser man ut fra figur 11(b) og figur 13 at modellen har en tendens til å predikere 'lave' verdier. Dette er i utgangspunktet ikke et problem i de tilfellene hvor de faktiske verdien er 'lave', men ut fra figur 5 ser man på 'halen' av grafen at det ligger flere 'uteliggere' som har høy verdi. På figur 11(b) viser den at modellen har en tendens til å predikere dårlig på høye oppholdslengder. Det å predikere 'uteliggere' i data-science er et kjent problem og som er vanskelig å fikse. Noe som kan forklare årsaken bak hvorfor modellen sliter med å predikere høye sykehusopphold, da dette er sjeldne tilfeller.

Hadde jeg hatt bedre tid på prosjektet, hadde jeg brukt mer tid på å utforske dataen, da spesielt fysiologisk data for å se om noe kan hentes ut der. Domenekunnskap innen fysiologi og patologi kunne bygd en bred forståelse på hvem som har en tendens til å oppholde seg lenge på sykehus. I tillegg kunne jeg vurdert å prøvd PCA-transformasjon for å redusere kompleksiteten på datasettet. Dessuten var dette prosjektet en læringsprosess hvor mye av arbeidet ga mer og mer mening desto lengre tid jeg brukte på det.

Til slutt vil jeg nevne at reproduserbarhet er et viktig element i data science, da man avhenger av å få samme resultat hver gang for å diskutere resultatene. Derfor er alle tilfeldige prosesser seedet med en fast seed ved hjelp av *random_state* argumentet i disse funksjonene det gjelder.

5 Konklusjon

Konkludert sagt, kan man si at kunnskapen man har fått fra INF161, gir et solid grunnlag for utviklingen av en maskinlæringsmodell i stand til å predikere sykehusopphold basert på pasientdata tilgjengelig fra dag 1. Dessuten kan dette videre implementeres til en nettside hvor brukeren kan legge inn egne data. Resultatene viser at modellen har en RMSE på ca. 19 dager, noe som tyder på at den ikke er bra nok for å implementeres for klinisk bruk helt enda. For å forbedre modellen kunne man brukt mer tid på analyse og anvendt mer avanserte metoder innen maskinlæring, f.eks. metoder fra INF264.

6 Referanser

Nettsider

[1] Helsebiblioteket/BMJ. *Blodtrykksmåling ved høgt blodtrykk*. `https://www.helsenorge.no/undersokelse-og-behandling/blodtrykksmaling/`. [Nettside; besøkt 2024-10-31]. 2023.

[2] Chris Nickson. *PaO2/FiO2 Ratio (P/F Ratio)*. `https://litfl.com/pao2-fio2-ratio/`. [Nettside; besøkt 2024-10-28]. 2024.

[3] Trond Nordseth. *NEWS (National Early Warning Score)*. `https://sml.snl.no/NEWS_-_National_Early_Warning_Score`. [Nettside; besøkt 2024-10-28]. 2024.

[4] Nick Potter. *Pipelines for Preprocessing: A tutorial*. `https://www.kaggle.com/code/nnjjpp/pipelines-for-preprocessing-a-tutorial`. [Nettside; besøkt 2024-10-28]. 2024.

[5] scikit-learn. *3.1. Cross-validation: evaluating estimator performance*. `https://scikit-learn.org/1.5/modules/generated/sklearn.model_selection.train_test_split.html`. [Nettside; besøkt 2024-10-30].

[6] scikit-learn. *scikit-learn - Machine Learning in Python*. `https://scikit-learn.org/stable/`. [Nettside; besøkt 2024-10-31].

[7] Mirko Stojiljković. *Split Your Dataset With scikit-learn's train_test_split()*. `https://realpython.com/train-test-split-python-data/`. [Nettside; besøkt 2024-10-29]. 2024.

Videoer

[8] Vincent D. Warmerdam. *Scikit-learn Crash Course - Machine Learning Library for Python*. `https://www.youtube.com/watch?v=0B5eIE_1vpU&ab_channel=freeCodeCamp.org`. [Nettside; besøkt 2024-10-28]. 2021.