ML Eksamensopgave Efteråret 2020

PBA Softwareudvikling

Lars Søndergaard Petersen

Introduktion

Denne opgave er lavet som afsluttende eksamen i faget machine learning.
Rapporten indeholder et afsnit per spørgsmål stillet i opgaveformuleringen. Samt et kort extra afsnit, F4, om optimering med Optuna.

Efter afleveringsfristen kan tilhørende kode findes her:
https://github.com/trezum/ML-Exam

I.Opgave 1

A.Forstå problemet og data

1.Hvilken slags ML problem taler vi om med dette datasæt?

Det er et binært klassifikations problem. Fordi vi har to klasser, survived og died. Det betyder også at vi kender den label der hører til de enkelte samples og kan derfor bruge supervised learning til løsningen.

2.Hvor mange features indeholder dette datasæt?

Datasættet indeholder følgende features: PassengerId, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked.

Survived er vores label, som vi vil prøve at forudsige, så den kan ikke bruges som feature. Man kan argumentere for at nogle af features er ubrugelige, men mere om det senere.

Altså 11 features og 1 label

3.Hvor mange samples indeholder dette datasæt?

En sample er en række i vores datasæt, titanic_800.csv har 801 rækker, men den øverste er en beskrivelse af hvad hver kolonne indeholder, så i alt 800 samples.

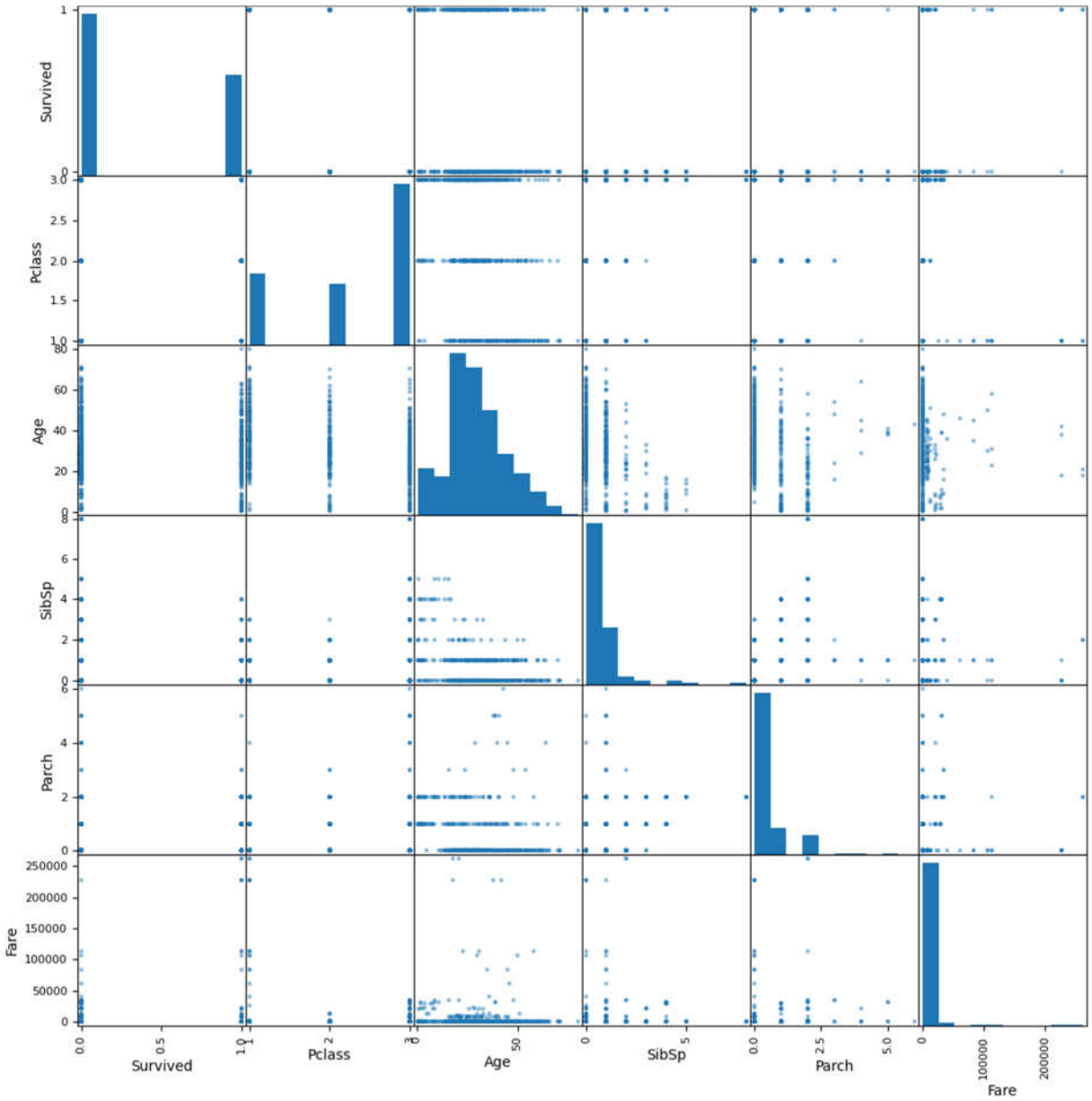
B.Visualisering af data og statistik

Inden data er renset, kan de features der indeholder tekst ikke tegnes i et diagram eller arbejdes med matematisk, derfor er der nogle af dem som ikke visualiseres eller laves korrelationer for her.
Se titanic_visualize.py for kode tilhørende denne del.

1.Visualiser de ting du mener er relevante for at forstå data bedre

Ved første øjekast på scatter_matrix ser det ud som om der ikke rigtig er nogle korrelationer.

Måske en let sammenhæng mellem age og nogle af de andre features:



Det giver fin mening at jo yngre man er jo flere søskende har man med om bor eller jo ældre du er jo flere penge har du til at købe en højere klasse billet.

2.Er der nogle stærke (lineære) korrelationer mellem noget af data?

Når man køre df.corr() udregner pandas korrelationerne, det giver følgende resultat:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1,000000	-0,001568	-0,051697	-0,051697	-0,081148	-0,021540	0,041213
Survived	-0,001568	1,000000	**-0,325399**	-0,087131	-0,026043	0,076603	0,059578
Pclass	-0,051697	**-0,325399**	1,000000	**-0,372172**	0,077940	0,012698	-0,144630
Age	0,067472	-0,087131	**-0,372172**	1,000000	**-0,306498**	-0,182402	-0,023523
SibSp	-0,081148	-0,026043	0,077940	**-0,306498**	1,000000	**0,410972**	0,079996
Parch	-0,021540	0,076603	0,012698	-0,182402	**0,410972**	1,000000	0,090814
Fare	0,041213	0,059578	-0,144630	-0,023523	0,079996	0,090814	1,000000

Som man kan se i tabellen, er der kun 4 kombinationer der har en absolut korrelation på over 0.3. Så ingen korrelationer stærke end ca. 0.41.

En interessant observation er at korrelationen mellem Pclass og survived var meget svær at se i grafen fordi der kun er 6 kombinationsmuligheder i alt, men pandas kan finde den alligevel. Derudover er det også interessant er at survived ikke korrelere ret meget med andet end Pclass.

C.Rensning og klargøring af data

Koden tilhørende dette afsnit kan findes starten i alle de py filer der starter med titanic_

1.Features der smides væk

Følgende features, markeret med fed, smides væk:

PassengerId er et tildelt tal og indeholder derfor ingen information om en person på nogen måde.

Name burde ikke indeholde ret meget information. Jeg har dog nogle tanker om at det måske kunne bruges, men vælger at lade være i første omgang. Der er en titel fx Dr, Master, Duchess, Mr, Mrs, Miss. som kunne vælges ud og sige noget om en persons tilstand eller rigdom. Det ville nok give mere mening hvis der var flere med de sjældne titler, fx er der kun en duchess.

Udover det kan et efternavn måske også give noget information, men jeg tror at meget af dette vil være dækket af Fare featuren.

Ticket er et billet nummer som ikke indeholder meget info om personen.

Cabin har så meget manglende data at det næsten i sig selv er en grund til at smide den væk, ud over det har vi en separat feature der beskriver hvilken klasse passageren rejser på. Så den information man måske kunne få omkring klasse findes allerede.

2.Håndtering af manglende data

Som tidligere vist ser age ud til at være en af de features der korrelerer meget med de andre og bør derfor bevares selvom der er en del manglende data. Mine erfaringer fra undervisningen siger mig at et godt startsted for at udfylde manglende data er at fylde gennemsnittet for det andet data ind.

Her skal man huske at bruge gennemsnittet for træningsdata til at udfylde manglende træningsdata, så man ikke bruger information fra testsættet. Det samme gælder for test sættet, de to sæt af data skal holdes isoleret fra hinanden.

Udover age er der en anden af de features som jeg har valgt at beholde der mangler data, det drejer sig om Embarked, hvor en sample er tom.

Jeg vælger at droppe denne række fordi det kun drejer sig om en. Det gøres lige efter indlæsning fordi det giver øget kompleksitet hvis det gøres efter opsplitning imellem test og train.

3.Brug af skalering/normalisering

Jeg vælger at skalere data fordi det giver bedre performance ved ML modeller som afhænger af den euklidiske afstand mellem data, fx knn, mlp, kmeans og pca. Mig bekendt er der ikke nogen af de algoritmer vi har arbejdet med der bliver decideret dårligere af at data bliver skaleret. Nogle af dem kan dog arbejde på ikke skaleret data uden problemer fx Random forrest.

4.Overblik over resterende data / tilstand.

Har er et eksempel på hvordan en sample ser ud efter data er blevet klargjort:

[-3.37843215e-01 7.12092320e-01 4.36690046e-01 -4.91687309e-01 -4.63258177e-01 -1.92108559e-01 5.99831852e-01]

Jeg har altså 7 features tilbage. Det der kun er blevet droppet 1 sample, den pga manglende embarked, er der 799 tilbage.

5.Hvor mange samples vil du putte i træningssættet og hvor mange i testsættet og hvorfor?

For at kunne validere at modellen virker korrekt er det vigtigt at have en tilstrækkelig mængde test data. Da jeg kun arbejder med 799 samples, vælger jeg at dele data op så træning sættet bliver 80% af data og testsættet bliver 20%. Hvis man havde flere samples, kunne testsættes relative andel gøres mindre fx 10% eller 5%.

D.Vælg en ML model og træn på data.

Jeg ved at problemet er et binært klassifikationsproblem, der for skal den model der vælges kunne arbejde med det. Det kan fx være LogisticRegression, KNeighborsClassifier, LinearSVC, DecisionTreeClassifier, RandomForestClassifier, MLPClassifier.

Tidligere har vi konstateret at der ikke er nogen særlig lineær sammenhæng i data, hellere ikke mellem survived og features. Derfor virker LinearSVC ikke særlig lovende, men lad os tage den med i udvælgelsen og køre en test med alle de nævnte classifiers med default parametre, dog en højere max_iter på de modeller der har brug for det.

Her ses outputtet en test kørsel, koden kan findes i titanic_classifier_selection_singlerun.py:

	Precision	Recall	F1
LogisticRegression	0,7656250	0,7205882	0,7424242
DecisionTreeClassifier	0,6774194	0,6176471	0,6461538
LinearSVC	0,7903226	0,7205882	0,7538462
RandomForestClassifier	0,7600000	0,5588235	0,6440678
MLPClassifier	0,8333333	0,6617647	0,7377049
KNeighborsClassifier	0,8103448	0,6911765	0,7460317

Overraskende nok er LinearSVC den bedste ud fra denne ene kørsel med default parametre. Man kan sikkert få de andre til at performe beder ved at prøve andre parametre. Man bør teste hver classifier flere gange og tage gennemsnittet for at få et mere præcist billede.

Det er gjort her, hver classifier er kørt 100 gange, koden kan findes i titanic_classifier_selection_multipleruns.py:

	Precision	Recall	F1
LogisticRegression	0,7656250	0,7205882	0,7424242
DecisionTreeClassifier	0,6809820	0,6039710	0,6399170
LinearSVC	0,7903226	0,7205882	0,7538460
RandomForestClassifier	0,7417350	0,5805880	0,6508530
MLPClassifier	0,8216420	0,6747060	0,7406050
KNeighborsClassifier	0,8103448	0,6911765	0,7460320

Med et gennemsnit over 100 kørsler ser det ud til at LinearSVC, MLPClassifier, LogisticRegression og KNeighborsClassifier performer nogenlunde lige godt LinearSVC er stadig foran, men jeg tror de andre kan komme med efter parameter optimering. Derfor går jeg videre med dem og vælger min endelige model i afsnit F, om eksperimenter.

E.Evaluating performance on the test set.

1.Min model

Som tidligere vist har de udvalgte modeller nogenlunde lignende F1 score:

	Precision	Recall	F1
LogisticRegression	0,7656250	0,7205882	0,7424242
LinearSVC	0,7903226	0,7205882	0,7538460
MLPClassifier	0,8216420	0,6747060	0,7406050
KNeighborsClassifier	0,8103448	0,6911765	0,7460320

2.Confusion Matrix

Confusion matrix er en matrice som er n*n stor hvor n er antallet af forskellige labels. I vores binære klassifikationsproblem med Titanic data vil den altså være 2*2. Hvorimod fx MNIST datasættet vil være en 10*10 matrice, datasættet har en label per tal mellem 0 og 9

Når der er snak om en 2*2 matrice opbygning være som nedenstående dog med tal i stedet

TN	FP
FN	TP

Eksempler for hvordan begreberne fra confusion matrix bruges i Titanic Datasættet:

	Betyder	Titanic
TN	True Negative	Det antal modellen gætter som døde, som faktisk var døde i træningsdata.
FP	False Positive	Det antal modellen gætter på overlevede men som faktisk var døde.
FN	False Negative	Det antal modellen gætter på var døde men faktisk overlevede.
TP	True Positive	Det antal modellen gætter på overlevede som faktisk overlevede.

3.Forklare også hvad er forskellen på precision og recall

a)Precision

Precision scoren viser hvor god modellen er til at klassificere med ”positive” labels.

I Titanic datasættet viser precision forholdet mellem det samlede antal overlevende og det antal modellen klassificerer til at have overlevet, altså hvor god modellen er til at klassificere om folk overlevede.

Formlen er: Precision = TP / (TP + FP)

b)Recall

Recall scoren er forholdet mellem det fundne antal positive og det samlede antal som burde have været positive.

I Titanic datasættet viser recall forholdet mellem det antal modellen klassificerer som overlevede og det faktiske antal overlevede.

Formlen er: Recall = TP / (TP + FN)

4.Hvad betyder F1 scoren?

F1 Scoren er et gennemsnit af precision og recall, dog ikke et normalt gennemsnit. F1 er et vægtet gennemsnit der gør at lave værdier, tæller mere.

Når man bruger F1 til at evaluere sin classifier vægter man at precision og recall skal være ens. Til nogle applikationer kan det være vigtigere at have en høj recall eller precision, de er ens.

5.Hvad din model er god til og ikke god til?

Her ses confusion matrix, fra en enkelt kørsel, for de udvalgte modeller, koden kan findes i titanic_confusion_matrix.py:

LogisticRegression	
77	15
19	49

LinearSVC	
79	13
19	49

MLPClassifier	
82	10
23	45

KNeighborsClassifier	
81	11
21	47

Som også vist tidligere er alle modellerne stort set lige dårlige. Det vises af tabellen over precision og recall i afsnit E, men det er lidt spøjst at MLPClassifier er den model der får flest False Negatives.

På samme tid den der får færrest False Positives. Derfor har den har også flest True Negatives og er altså den model man bedst kan stole på når den siger en person er død. Samtidig er det også MLPClassifier man mindst kan stole på når den siger at en person overlevede.

Alt i alt virker performance ringe, det bliver spændende at se hvor stor forbedring der kan opnås.

F.Experimenter - forbedringer af performance

1.Eksperimenter og resultater

Jeg har kørt grid serach eksperiment for hver af de udvalgte modeller, her er en tabel hvor de opnåede F1 Score og de originale kan ses:

	F1 Grid	F1
LogisticRegression	0,7343749	0,7424242
LinearSVC	**0,7538461**	0,7538460
MLPClassifier	0,7286821	0,7406050
KNeighborsClassifier	0,7394957	0,7460320

Koden brugt til at finde F1 Grid kan findes i de filer der starter med titanic_experiment_*.py, hvor * er navnet på classifieren, nederst i denne fil findes også de opnåede bedste parametre.

2.F1 Forbedring

Som man kan se i tabellen ovenfor, er alle F1 scorerne undtagen for LinearSVC faktisk blevet dårligere end i det originale eksperiment.

Dermed bliver den endelige udvalgte model LinearSVC, godt den kom med videre i forløbet.

3.Overraskende eller forventet

Ved første øjekast blev jeg overrasket over at F1 Scoren generelt blev dårligere, men det giver egentlig god mening når man går koden igennem. Det sker af to grunde.

GridSearchCV laver, som navnet antyder, Cross Validation. Det betyder at den opdeler det givne datasæt på nogle forskellige måder og tester med forskellige træning og test set. Hvorimod den originale kode kun tester med samme datasplit. Altså bør Grid F1 være en mere præcis score for hvor godt modellen generalisere.

I koden uden GridSearchCV bruges 80% af datasættet til træning og 20% til test. I koden med GridSearchCV bruges de 80% som input, dermed trænes der på en andel af de 80% og altså en mindre datamængde. Denne måde at gøre det på følger scikit learn, og de generelle, anbefalinger om at dele sine data op i et development sæt og et evaluation sæt.

Se evt: https://scikit-learn.org/stable/modules/grid_search.html#model-selection-development-and-evaluation

4.Optuna optimering uden crossvalidation

Koden tilhørene dette afsnit kan findes i de filer der slutter med _optuna.py

Efter at have løst resten af opgaverne fik jeg lyst til at gå tilbage og se hvor høj F1 score jeg kunne opnå med Titanic data. I undervisningen havde vi en konkurrence hvor jeg fandt ud af at optimeringsframeworket Optuna kan være ret nyttigt.

For mere info se: https://optuna.org/ og https://optuna.readthedocs.io

En af fordelene ved Optuna er at det ikke er alle muligheder der prøves af, så søgningen er noget hurtigere.

Udover skiftet til optune er der lavet nogle rettelser i forhold til de kørsler der blev lavet med GridSearchCV.

Den manglende embarked værdi er sat til S, så der er en observation mere i det samlede datasæt. S er valgt fordi det er den værdi der optræder oftest.

Der søges gennem forskellige måder at håndtere manglende data, drop, mean og median.

Der søges gennem forskellige måder at skalere data på, MinMaxScaler, MaxAbsScaler, RobustScaler, StandardScaler, QuantileTransformer, PowerTransformer, PolynomialFeatures.

Der laves ikke cross validation, det vil sige at resultatet kommer til at passe meget til de data vi arbejder med og generalisere noget dårligere.

Her ses F1 scorerne fundet med Optuna:

	F1
LogisticRegression	0,837606838
LinearSVC	0,806722689
MLPClassifier	**0,862068966**
KNeighborsClassifier	0,823529412

Det udvider søgerum lille rettelse til embarked har givet noget af en forbedring i forhold til tidligere, hvor den bedste F1 var omkring 0.75. Som vist tidligere har manglende crossvalidation også en del at sige. Når man sammenligner med de kørsler der er lavet tidligere uden cross validation er det stadig en stor forbedring.

Se nederst i de tilhørende filer for de endelige parametre til hver model.

II.Opgave 2

A.Forstå data

1.Hvor features er det i dette datasæt?

Datasættet indeholder seks features (island, culmen_length_mm, culmen_depth_mm, flipper_length_mm, body_mass_g og sex) og en label species.

2.Hvor mange samples er der i dette datasæt?

Der er 344 samples i datasættet.

B.Visualisering og klargøring af data

Koden til dette afsnit findes i filen penguins_visualize.py

1. Håndtering af manglende data og begrundelse.

Der er to samples hvor alle features undtagen island er NA, de samples indeholder så lidt reel data at de bare kan droppes uden den store påvirkning.

Der er ni samples hvor sex featuren ikke er MALE eller FEMALE. Til trods for at de ni samples indeholder en del data vælger jeg i først omgang at droppe dem. En anden løsning kunne være at tildele et tilfældigt køn, eller tildele det køn der ligner bedst ud fra det sæt, test eller train, som de samples ender op i. Hvis det senere viser sig at sex featuren har en meget lav betydning kan man evt. droppe den i stedet.

2. Forskellen på PCA og SelectKbest.

Hvis formålet her er at udvælge de to bedste features uden at transformere data kan PCA ikke bruges. SelectKbest passer perfekt til det formål. Data vil blive transformeret hvis man bruger PCA så den visualisering vi skal have lavet ville ikke vise reel data men det rum som PCA transformere til.

Så hvis formålet er at finde de to bedste features og visualisere dem vil PCA ikke være brugbar. Man kan dog transformere alt data til to dimensioner med PCA og visualisér dem, så hvis man er ligeglad med om visualiseringen viser reel data kan PCA godt bruges.

3.Hvilke to features (angive navnene på de to features) er de bedste ifølge SelectKbest?

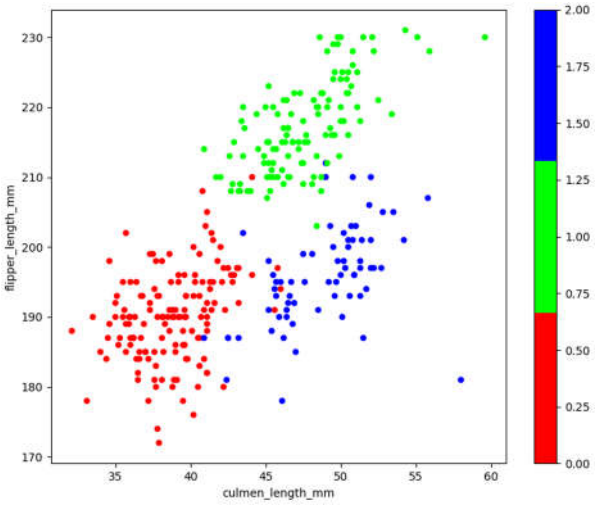
Her ses scorene fra SelectKbest, de to udvalgte features er markeret med fed:

island	**culmen_length_mm**	culmen_depth_mm	**flipper_length_mm**	body_mass_g	sex
142,70968	**397,29944**	344,82508	**567,40699**	341,89489	0,02409

Som man kan se på scoren, har sex featuren, ifølge SelectKbest, meget lav betydning for hvilken race en pingvin er. Hvilket logisk set giver rigtig god mening. Så man kan argumentere for at fjerne sex featuren og inkludere de samples der blev droppede på grund af manglende værdier i stedet.

4.Visualisering af resultat

Her ses resultatet af visualisering, det ser ud til at der er et par enkelte individer som vores cluster algoritme får svært ved at placere:



C.Brug af clustering kmeans

1.Udfra din viden om data, hvor mange clusters skal vi så forvente at finde, hvis en clustering skal være god?

Der er tre mulige labels (pingvin racer), derfor skal der være tre clusters.

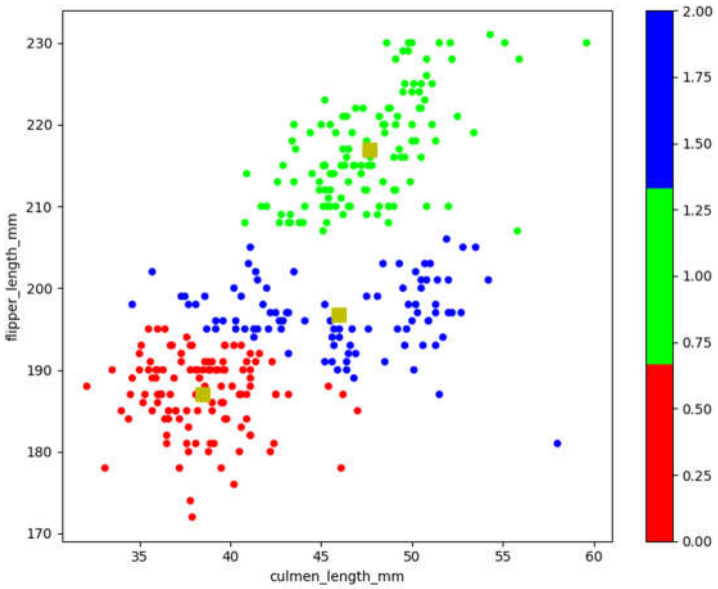
2.Paser dette tal nogenlunde med visualiseringen (scatterplot) fra delopgave b) ?

Når man ser på scatterplottet fra tidligere ser de røde og grøn racer nogenlunde adskilte fra de andre og samtidig samlede i et cluster. Den blå race ser også ud til at være adskilt fra de andre, men ser ud til at være delt i to pga. manglende individer med omkring 47mm culmen length.

Det kan give udfordringer for clustering algoritmer.

3.Sammenlign visuelt de to plots - hvor god synes du kmeans er?

Her ses plottet over de clusters kmeans har fundet:



Det ser meget ringe ud, en stor del af det ene cluster er kategoriseret forkert.

4. Hvis vi nu gerne ville have et tal for hvor god vores clustering er, har du så nogle ideer til hvad man kunne gøre?

Da vi kender det den rigtige klassifikation kan vi regne ud hvor stor en andel af individerne der er klassificerede forkert ved at tælle dem og finde andelen. Det er et okay tal for hvor god vores clustering er, se metoden correct_observation_calculator.

Resultatet herover ligger på 83,48% rigtige.

D.Brug af DBScan

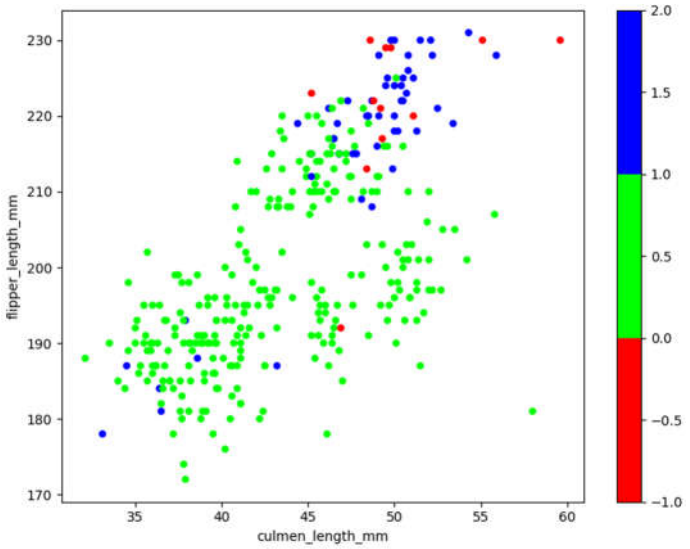
Koden tilhørene dette afsnit kan findes i penguins_dbscan.py

1.Tror du at DBScan vil kunne klare sig bedre på de to bedste features end kmeans?

Da clusters i data for de to bedste features ser rimelig adskilt ud afstandsmessigt tror jeg DBScan klare sig bedre end kmeans. Som tidligere nævnt vil der nok være lidt udfordringer med det ene cluster.

Grunden til at jeg tror DBScan kommer til at klare sig bedre er at den kigger på afstanden mellem de enkelte samples. Hvorimod kmeans vælger det antal punkter man giver den, i dette tilfælde tre, som flyttes rundt indtil den har fundet de tre punker hvor der er mindst afstand fra punktet og til de omkring liggende samples.

Her ses resultatet fra en kørsel af DBScan:



Til min store forundring er det, selv når man kender svaret, ret svært at få DBScan til at performe godt på dette datasæt. Det højeste jeg har kunnet komme op på er 55% rigtige som ses på billedet herover. Værdierne til den kørsel blev fundet med optimeringsframeworket Optuna.

Min konklusion er at der er for meget overlap mellem clusters til at algoritmen kan finde noget brugbart med tre clusters og at DBScan i modsætning til mine forventninger faktisk er dårligere end kmeans på dette data.