MAL O4 Journal

ITMAL-01 Forår 2022

Journal for O4 - Projekt

Afleveret af: Gruppe 21

Retning	Navn		Studie ID	AU-nummer
E	Lukas Kezic	LKE	201906917	AU637735
E	Rasmus Holm Lund	RHL	201900058	AU630898
E	Jakob Peter Aarestrup	JPA	201907898	AU632998
E	Frederik Thomsen	FT	201906146	AU637451

Kontaktperson:

Retning	Navn	Studie ID	Email
E	Rasmus Holm Lund	201900058	201900058@post.au.dk

Aarhus Universitet
16. maj 2022

Indholdsfortegnelse

1 Indledning

Denne journal giver et indblik i det endelige projekt udarbejdet af gruppe 21. Journalen er baseret på et selvvalgt datasæt, der bliver beskrevet i et senere kapitel, og en problemstilling, der vil blive forsøgt på at blive besvaret. Denne besvarelse vil være ud fra et end-to-end perspektiv - Dvs. en fuldstændig proces, der starter med i en problemstilling og slutter i et færdigt produkt.

Arbejdsfordeling

Arbejdsfordelingen i projektet og journalen er som udgangspunkt en fælles indsats der er nogenlunde ligeligt fordelt. Derfor vil vi som standpunkt bestemme at alle medlemmer har deltaget i alle dele.

2 Problemstilling

Med de mange forskellige vine på markedet. Kan det være svært som forbruger at skelne mellem, hvad der er en god vin og hvad der er en dårlig vin. Dette problem vil vi prøve at løse i denne opgave ved at kigge på de kemiske egenskaber, som findes i forskellige vine og herved undersøge om der er sammenhæng mellem den kemiske opbyggelse af vin og forbrugernes smag? Der er derfor valgt et datasæt der netop består af vine rangereret og analyseret. Vores algoritme skal, ud fra dette sæt, kunne differentiere imellem gode og dårlige vine. Algoritmen skal kunne rationalisere en sammenhæng mellem de kemiske parametre og kvalitet af vin ud fra en score sat i datasættet. Endeligt skal algoritmen kunne give en vin med bestemte parametre og derved kunne rangere vinen af sig selv.

3 Datasæt

3.1 Beskrivelse af datasæt

Til O4 projektet har vi valgt at arbejde med vine og deres tilhørende kvalitet. Formålet er at kunne differencere mellem gode og dårlige vine ved at kigge på de kemiske egenskaber. Dette er gældende for både røde og hvide vine, hvor hver vin har en score fra 1-10. Sammenhængen mellem scoren og de kemiske egenskaber skal skabe grundlaget for hvordan algoritmen bestemmer hvilke vine, der er gode og hvilke der er dårlige.

Til projektet er der valgt et datasæt baseret på en undersøgelse fra Portugal [1], der beskriver typen "Vinho Verde"i både rød og hvid variant. Vinene i sættet har en række parametre, der består af deres kemiske kvaliteter som; tæthed, pH-værdi, sulfitter, alkohol og kvalitet, der rangerer vinene mellem 1 og 10. Kvaliteten af vinen er rangeret ud fra 3 vineksperters vurdering. Datasættet er fundet igennem UCI Machine Learning Repository, der er en database med datesæt, som specifikt er gode til machine learning. Selve sættet kan findes på følgende hjemmeside:

https://archive.ics.uci.edu/ml/datasets/Wine+Quality

Datasættet, der består af 4898 samples indeholder 12 parametre, hvor de første 11 er de kemiske egenskaber og den sidste er scoren. Parametrene kan ses på tabel 1:

Som udgangspunkt er der ikke en særlig stor samplesize i dette sæt, hvilket typisk godt kan skabe problemer, hvis man vil lave en succesfuld machine learning algoritme. Det kunne f.eks. være i nogle billedgenkendelses opgaver, hvor en stor samplesize kan være ret vigtig. I tilfældet med dette datasæt burde det dog ikke

1	Fixed acidity	7	Total sulfur dioxide
2	Volatile acidity	8	Density
3	Citric acid	9	pH
4	Residual sugar	10	Sulphates
5	Chlorides	11	Alcohol
6	Free sulfur dioxide	12	Quality (1-10)

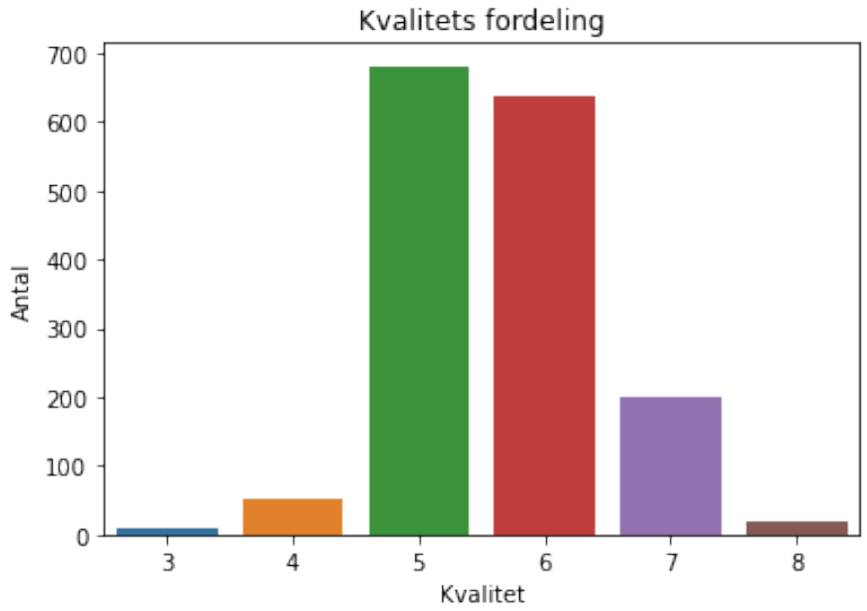
Tabel 1: Parametre i datasæt

skabe store problemer. Derudover gælder sættet kun for en enkelt type vin, hvilket gør det svært at bruge resultaterne ift. andre typer. Samtidigt er det ikke sikkert at alle parametre er relevante, men dette giver også mulighed for at teste en feature selection method.

Datasættet kan både bruges til at lave en regression eller klassifikation, men muligheder som neural network giver ikke meget mening med dette datasæt. Som udgangspunkt er størstedelen af vinene rangeret enten som 5 eller 6 i score. Derfor vil sættet ikke være særlig effektivt til at rangere middelmådige vine ift. hinanden. Derimod burde sættet fungere en del bedre til at bestemme outliers. Dvs. sættet burde være godt til at finde de meget dårlige vine og dem som er helt exceptionelle. Det er derfor vores fokus i dette projekt.

3.2 Dataanalyse af datasættet

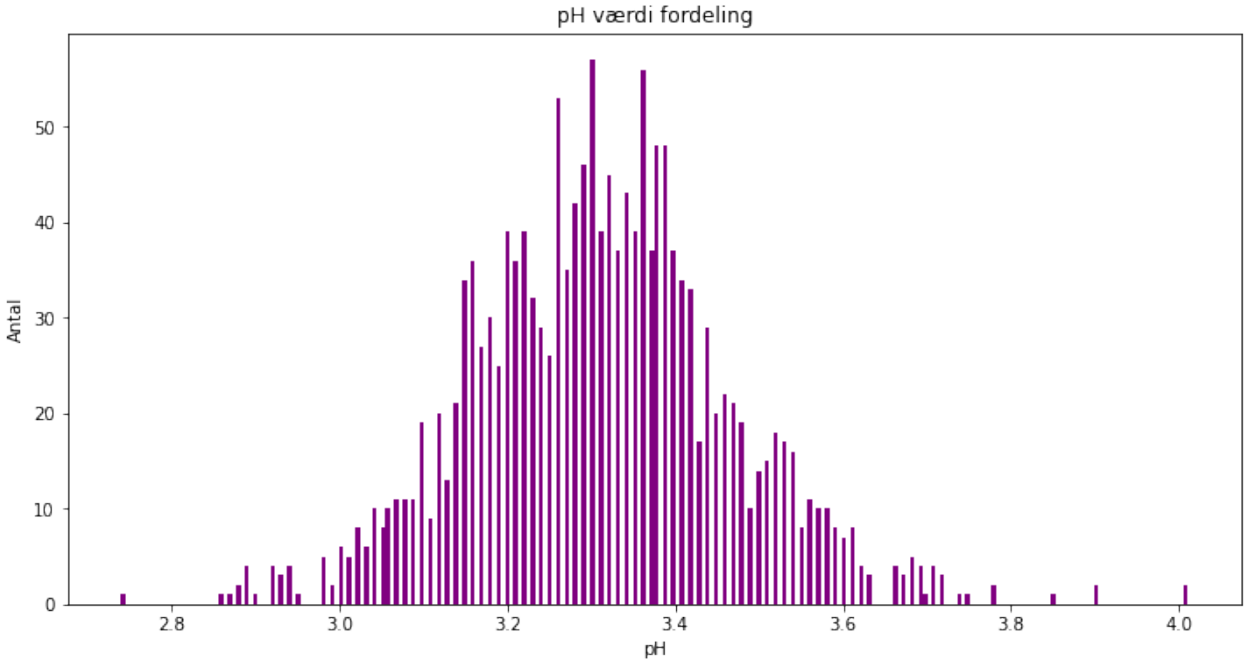
På Figur 1 kan fordelingen af kvaliteten ses. Heraf at de fleste vine i vores datasæt har fået en score på 5 og 6. Derudover er der en del som har fået 7 også. Outlierne i denne fordeling er helt klart vine med scoren 3 og 8. Hvor der er flere med en score på 8 end 3.



Figur 1: Kvalitets histogram

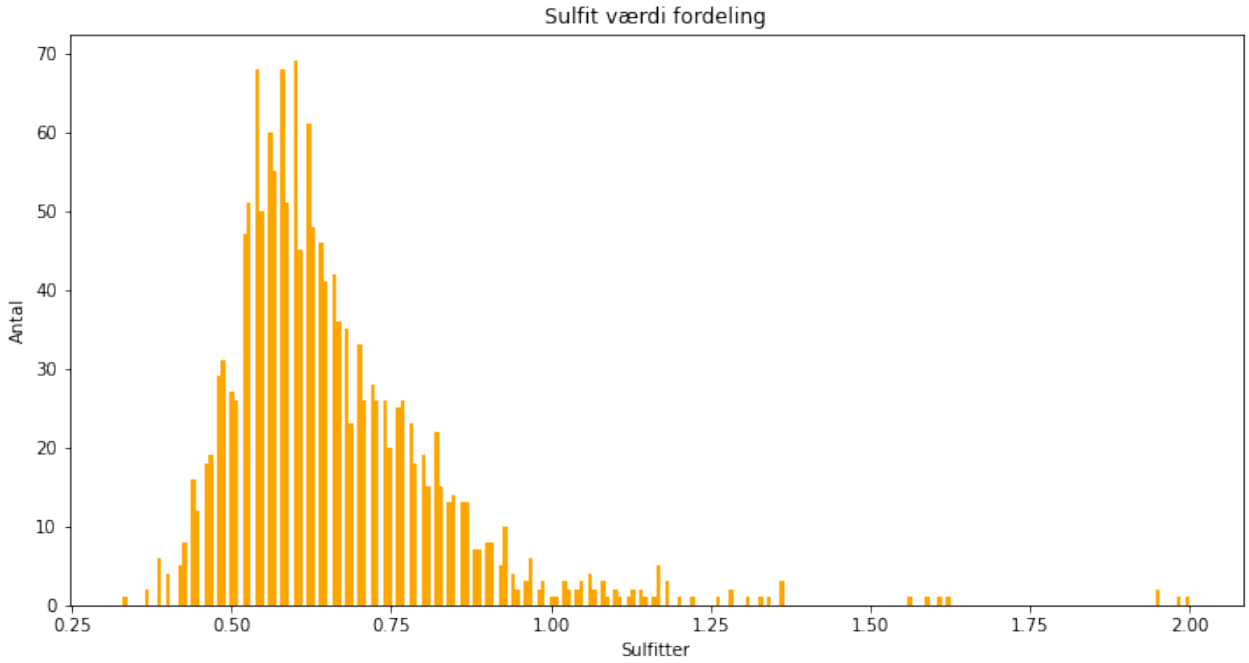
På Figur 2 er pH-værdi fordelingen vist. I denne fordeling kan ses at størstedelen af vinene har en pH-værdi

mellem 3.2-3.4. Hvor at der er nogle enkelte over 4 og enkelte under 2.7. Dvs. hvis man kigger på pH-skalaen er det meste af vinen omkring syrlig i pH-værdi fordelingen. Dog med outliers på 4 som er mindre syrlige som bevæger sig mod basis og 2.7 der bevæger sig mod mere ætsende.



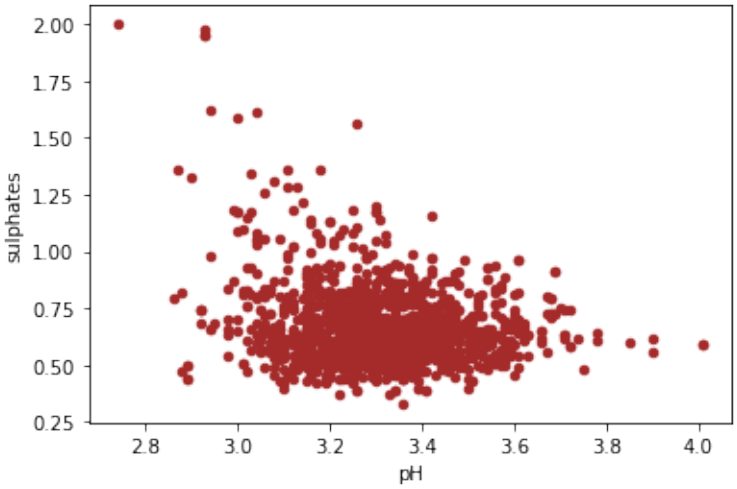
Figur 2: pH histogram

På Figur 3 kan fordelingen af sulfitter i vinene ses. Her ses den samme normalfordeling som ved pH-værdien ikke helt. Størstedelen af vinene har et lavere indhold af sulfitter, men der ses også outliers som næsten har firedobbelt så mange som medianen. Typisk for syrlige vine behøves der ikke samme mængde af sulfitter for at vinen er holdbar. Det hænger godt sammen med at størstedelen af vinene i dette sæt er i den syrlige ende.



Figur 3: Sulfit histogram

Kigger vi på Figur 4 er sammenhængen mellem pH-værdi og sulfitter også tydelig. Her har størstedelen af vinene en pH-værdi mellem 3,0 og 3,6, men samtidigt har de fleste også en sulfit værdi mellem 0,5 og 0,8.



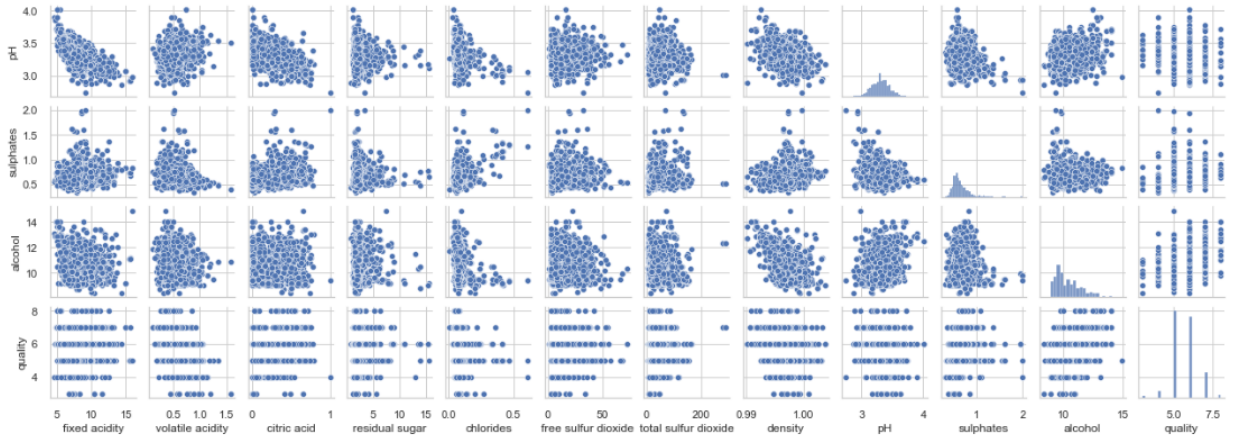
Figur 4: pH over sulfit scatter plot

Ligesom at dataen kan plottes kan der også findes median, middelværdi og standard afvigelse for de respektive parametre. På Kode snippet 1 kan udregningen af dette ses samt outputtet. Her er det tydeligt at middelværdien er tættest på medianen for pH-værdierne og længst fra for kvalitet. Samtidigt er standard afvigelsen betydeligt lavere for pH og sulfitter.

```
from statistics import mean, median, stdev

quality = wine['quality']
sulphates = wine['sulphates']
pH = wine['pH']

print("Quality:")
print("Mean:", round(mean(quality),3))
print("Median:",round(median(quality),3 ))
print("Std deviation:",round(stdev(quality),3))
print("Sulphates:")
print("Mean:",round(mean(sulphates),3))
print("Median:",round(median(sulphates),3))
print("Std deviation:",round(stdev(sulphates),4))
print("pH:")
print("Mean:",round(mean(pH),3))
print("Median:",round(median(pH),3))
print("Std deviation:",round(stdev(pH),3))

> Quality:
  Mean: 5.636
  Median: 6
  Std deviation: 0.808
  Sulphates:
  Mean: 0.658
  Median: 0.62
  Std deviation: 0.1695
  pH:
  Mean: 3.311
  Median: 3.31
  Std deviation: 0.154
```

Listing 1: Median, middelværdi og standard afvigelse

Generelt set kunne der ikke ses en bestemt sammenhæng mellem de forskellige parametre selv efter mange plottet grafer. Det kan også ses på Figur 5 hvor det ikke kan ses om der er en sammenhæng mellem kvalitet og de mange parametre. Derfor er der valgt at opdele dataen og lave en PCA for at finde de mest betydende parametre for kvalitet.



Figur 5: Pairplot for vin data

4 Valg af ML algoritme

Der er i projektet gjort forskellige overvejelser om hvilken algoritme, der ville være bedst at bruge. Derfor er projektet også testet med forskellige algoritmer for at bestemme den mest effektive. Til fælles har algoritmerne at de er gode til at finde outliers i et datasæt. Da dette datasæt klart ligger op til en outlier detection bliver algoritmen valgt ud fra dette. Heriblandt blev Support Vector Machine (SVM), men decision tree, isolation forest, k-NN og local outlier factor (LOF) var også relevante kandidater. Som hovedfaktor for valg af algoritmen var en høj præcision, der ikke var resultat af en overfitting. Derudover så vi en bedre præcision, der samtidigt ikke gav en helt skæv score. Det skal dog nævnes at andre algoritmer sagtens kan fungere til datasættet. Det handler primært om at sætte data'en ordenligt op og passe på at outliers ikke forskruer præcisionen. Samtidigt var det vigtigt at kunne regulere på algoritmen, hvor dette kunne gøres ved at ændre på C-værdien for SVM algoritmen. Fordelen ved at vælge SVM er at den producere præcise resultater med lavt brug af computer kraft. Derudover fungerer SVM godt når der er en klar seperation mellem klasser. Til gengæld har SVM svært ved at håndtere større datasæt og når der er meget støj i dataen.

5 ML data processering

For at kunne lave på vores data kræver det, at vi kan importere vores data samt dele det op. Det datasæt vi bruger er gemt i en CVS-fil og importeres til Python vha. read_csv() funktionen fra biblioteket pandas. I koden har vi delt datasættet op i en x og y-værdi. x-værdien indeholder alle de 12 kemiske parametre, hvor y-værdien indeholder en parameter kaldet 'Reviews', som svarer til en vins kvalitet fordelt over værdierne '1 - Bad', '2 - Average' og '3 - Excellent'. Vinene er blevet tildelt denne score ud fra parameteren quality. Nedenfor på Listing 6 kan koden ses.

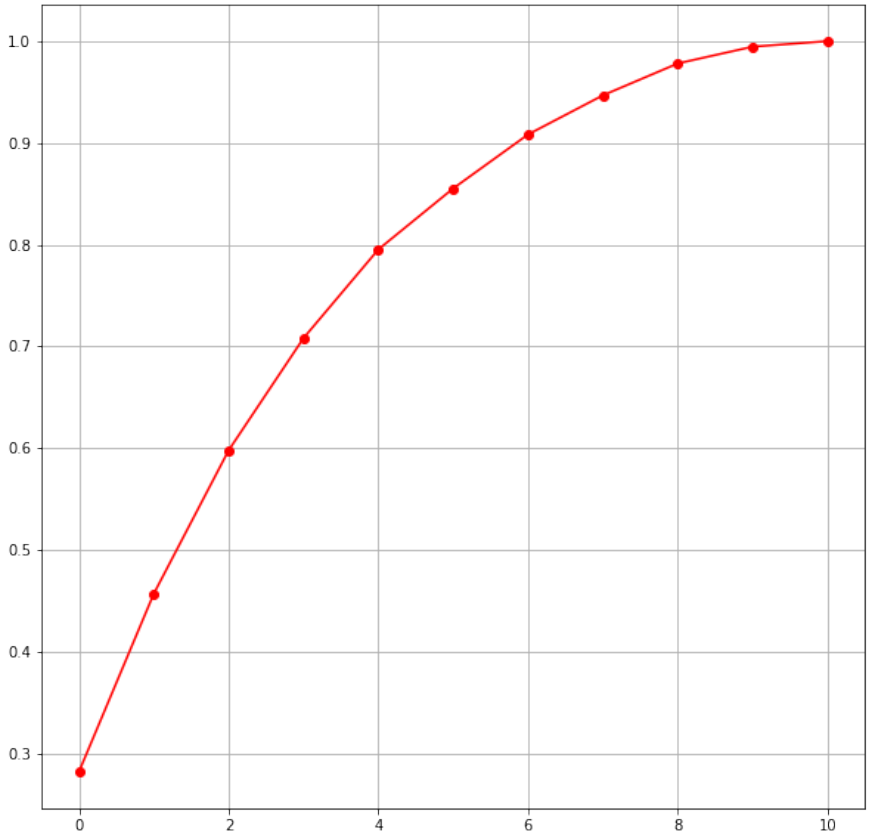
```
import pandas as pd

#data
data = pd.read_csv("input/winequality-red.csv")

#x
x = data.iloc[:,:11]

#y
reviews = []
for i in data['quality']:
    if i >= 1 and i <= 3:
        reviews.append('1')
    elif i >= 4 and i <= 7:
        reviews.append('2')
    elif i >= 8 and i <= 10:
        reviews.append('3')
data['Reviews'] = reviews

y = data['Reviews']
```

Listing 2: Opdeling af data

For at der kan udføres en PCA-test på datasættet, så skal datasættet skaleres ned. Dette gøres med StandardScaler() funktionen fra preprocessing biblioteket. Herefter kan der udføres en PCA-test. PCA-testen viser, at de 6 første parametre i datasættet, dvs. fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide og total sulfur dioxid har 90% betydning for vinens kvalitet. Nedenfor på Listing 3 kan koden ses og på Figur 6 kan PCA-testen ses.

```
#Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(x)

#PCA
from sklearn.decomposition import PCA
pca = PCA()
x_pca = pca.fit_transform(x)
plt.figure(figsize=(10,10))
plt.plot(np.cumsum(pca.explained_variance_ratio_), 'ro-')
plt.grid()
```

Listing 3: PCA-test dataprocessing



Figur 6: PCA-test

Herefter deler vi hhv. x- og y-værdierne ind i et test og train sæt. Train-sættet er det sæt vi vil træne med vores ML-algoritme, hvor test-sættet er det vil teste vores ML-algortime med for at se hvor godt den performer. Opdelingen udføres med funktionen train_test_split() fra biblioteket model_selection. Her er der valgt en test-size på 0.5, dvs. at fordelingen mellem dataen for train- og testsættet bliver 50%. Nedenfor på Listing 4 kan koden ses.

```
1 from sklearn.model_selection import train_test_split
2 x_train, x_test, y_train, y_test = train_test_split(x_new, y, test_size =
      0.5)
```

Listing 4: Train- & testset

6 Performance metrics

Der findes mange måde at checke performance for ens algoritme. Heriblandt accuracy, precision, recall og F1-score. På Scikits egen dokumentation for SVM algoritmen bruges funktionen score til at tjekke performance. Dette kan ses på scikits hjemmeside:

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

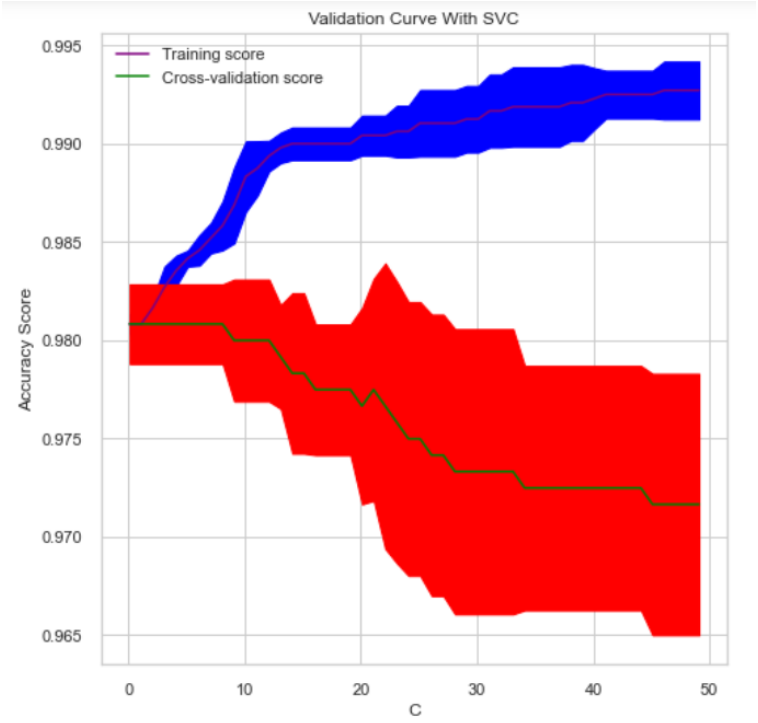
Scoren består af en middelværdi af accuracy for et givent datasæt. Dette skulle gerne give os en nogenlunde idé om præcisionen i systemet, men kan også være en relativt hård score, der kan afvige fra virkeligeheden i systemer, der har komplekse datasæt. Accuracy bruges typisk når man både vil tjekke for falske negativer og falske positiver. Derfor passer den også bedst til vores tilfælde, da vi gerne vil finde de vine, som er dårlige og dem som er rigtig gode.

7 Under- og overfitting

For at sørge for at vores data ikke bliver under- eller overfit, så benytter vi reguleringsmetoden kaldet Early Stopping til SGDClassifier modellen, og til SVM (Support Vector Machine) modellen ændrer vi på en reguleringsparameter. Early stopping metoden fungerer ved, at vi tjekker ved hvilken kapacitet (antallet af træninger) vi får den mindste mulige RMSE (Root Mean Squared Error) fejl. Denne kapacitet kaldes den optimale kapacitet, da vores system ikke under- eller overfitter vores data her.

For SGD-classifier modellen er Early Stopping blevet implementeret ved at skrive 'early_stopping=True'. Dette er den automatiske early stopping regulerings metode, og dermed stopper den selv, når modellen har nået sin optimale kapacitet. Early stopping kunne også have blevet implementeret manuelt med et for-loop og en if-sætning. Her ville vi i for-loopet træne samt udføre fit og predict på vores datasæt, hvorefter vi ville udregne vores Mean Sqauared Error. Herefter ville der være indsat en if-sætning der tjekker om vores validation error er mindre end vores minimum error. Når if-sætningen er true, svarer det til den kapacitet hvor vores validation error er mindst, og dermed er det vores optimale kapacitet.

For SVM har vi ændret på reguleringsparameteren 'C' for at sikre, at modellen ikke bliver under- eller overfit. Parameteren beskriver hvor meget regulering der skal foretages på modellen. For at se parameterens betydning er der blevet lavet en validation curve på den. Hvor det tydeligt kan ses på Figur 7 at desto højere en C der bruges desto mindre præcision fås der. På Listing 5 kan implementeringen af reguleringen ses.



Figur 7: Validation curve for SVM

```
1 #SGDClassifier
2 sgd_reg = SGDClassifier(max_iter=1,penalty=None,eta0=0.0005,warm_start=True
      ,early_stopping=True,learning_rate="constant",tol=-float("inf"),
      random_state=42)
3
4 #SVM
5 svm = SVM(C=0.01)
```

Listing 5: Implementering af regulering for SGDClassifier og SVM

8 Optimeringer og forbedringer

Der findes mange forskellige hyperparameter, i forskellige algoritmer. Disse hyperparameter kan man manuelt ændre efter hver test, eller man kan bruge GridSearchCV. GridSeachCV er en funktion hvor man kan indsætte forskellige hyperparameter, som gør at ens algoritme prøver de forskellige hyperparameter. Dette betyder, at man nemt kan finde de bedste hyperparameter kombinationer. Ulempen ved at bruge denne proces er dog at det tager lang tid, derfor køres denne proces separat for hoved koden, for at finde de bedste hyperparameter.

Eftersom vi valgte at SVC var den bedste algoritme, prøvede vi at gøre den endnu bedre med noget regulering af dens hyperparameter. Dette blev gjort ved at implementere vores kode i en ny kernel, og erstatte algoritme implementeringen med GridSearchCV. I GridSearchCV, opstillede vi 3 hyperparameter "C", "kernel"og "max_iter". Der blev fundet frem til at en "C"på 0.1, "kernel"skal være linear og en "max_iter"på 100, giver den bedste score. Der blev dog ikke testet meget, eftersom scoreren allerede var høj i forvejen og yderligere ændringer ikke ville gøre den store forskel. Vi fik en ændring fra 98,2% til 98,3%, hvilket ikke er en

stor ændring, men stadig en forbedring. Vi kunne have tilføjet endnu flere hyperparameter, men som nævnt tidligere ville dette tage længere tid og ikke nødvendigvis give os et bedre resultat.

```
model = SVC()

tuning_parameters = {
                    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
                    'kernel' : ('linear', 'poly', 'rbf', 'sigmoid'),
                    'max_iter' : [1, 10, 100, 1000]
                    }

CV = 5
VERBOSE = 0

grid_tuned = GridSearchCV(model,
                              tuning_parameters,
                              cv=CV,
                              scoring='accuracy',
                              verbose=VERBOSE,
                              n_jobs=-1
                             )
sv = grid_tuned.fit(x_train, y_train)
```

Listing 6: GridseachCV

Scoreren her er dog en estimering af den rigtige score, derfor er den ikke helt præcis, dog fik vi en score på 98% efter hyperparameterne blev indsat, hvilket var rimelig præcis. Dog gør hyperparameterne ikke en stor forskel for vores specifikke algoritme, eftersom der ikke er vildt meget data og fordi algoritmen allerede har en høj score. Dog kan vi få algoritmen til at blive dårligere ved at ændre "C"og "max_iter". Hvis vi ændre "C"til at være høj, vil vi få en lavere score.

9 Konklusion

Der er i dette projekt gjort brug af et datasæt bestående af vine, med deres kemiske parametre og en rangerering. Dette datasæt bruges til at bestemme, om der er en sammenhæng mellem de kemiske egenskaber i en vin og eksperters smagsløg. Som første punkt er datasættet blevet analyseret for om hvilke parametre, der havde størst indflydelse på rangeringen. I analysen blev det også gjort klart at datasættets bedste use case var en outlier algoritme. Derfor blev der udviklet en algoritme, som kan differentiere mellem gode og dårlige vine ud fra disse parametre. Vi kan konkludere at egenskaberne Volatile acidity og Fixed acidity havde den største effekt på rangeringen. Derfor var det også hvad algoritmen prioriterede højest når den blev præsenteret for en ny vin. Med vores eksempel datasæt kan vi også konkludere at algoritmen fungerede nogenlunde til at bestemme kvaliteten af vinen ud fra parametrene. Ved brug af SVM kunne algoritmen effektivt finde outliers. Dette blev gjort ud fra tre kategorier god, dårlig og middelmådig. Nøjagtigheden for algoritmen var på over 98%, hvilket er meget højt. Derfor kunne man godt mistænke at systemet er overfittet.

Litteratur

[1] Paulo Cortez m.fl. "Modeling wine preferences by data mining from physicochemical properties". I: *Decision support systems* 47.4 (2009), s. 547–553.