ENEL 682

**ML Course Project Report**

Author: Ahmad Masri (30115844)

horizontal line

# 

**Code and Explanation**

* The three main code files for each ML model are shown below, as for the full notebook output per code block executed, it will be in the .ipynb files uploaded.
* Please note that the preprocessing methods and others steps might be a bit different than the proposal’s structure, because while coding I might find new enhancing methods to add or ones to remove at that matter.
* Comments in yellow explain each code block in terms of preprocessing and methods used to train the ML model.
* Github URL as well: <https://github.com/ahmadelmasri95/ENEL-682-ML-Project>
* Dataset: winequality-red.csv

1. **Using Random Forest Classifier:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

*#reading data downloaded from Kaggle dataset*

wine\_df = pd.read\_csv('winequality-red.csv')

*#examining the dataframe for features and label*

wine\_df.head(5)

*#checking for missing values*

wine\_df.isnull().sum()

*#statistics*

wine\_df.describe()

*#examining the data through a plot*

sns.catplot(*x*='quality', *data*=wine\_df, *kind*='count')

*#splitting our data into X and y*

*#Doing a binary classification such that if quality is >= 7, then we have high quality i.e. 1, else 0*

X = wine\_df.drop('quality',*axis*=1)

y = wine\_df['quality'].apply(*lambda* *yval*:1 if yval>=7 else 0)

*#quick check of the spread of quality output in our dataset*

y.value\_counts()

*# train, test, split*

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, *test\_size*=0.2, *random\_state*=3)

print(y.shape, Y\_train.shape, Y\_test.shape)

*#train the Random Forest Classifier model*

model = RandomForestClassifier()

model.fit(X\_train, Y\_train)

*# accuracy on test data*

X\_test\_prediction = model.predict(X\_test)

print(accuracy\_score(X\_test\_prediction, Y\_test))

*#Trying new unseen dummy data in our prediction system*

input\_data = (6.8,0.220,0.34,3.4,0.031,14.0,16.0,0.92484,3.39,0.92,10.6)

np\_input\_data = np.asarray(input\_data)

reshaped\_data = np\_input\_data.reshape(1,-1)

predict = model.predict(reshaped\_data)

if predict[0] == 1:

print("Good Quality Wine")

else:

print("Bad Quality Wine")

1. **Using SVM Classifier:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.graph\_objects as go

from sklearn.preprocessing import StandardScaler

scaler=StandardScaler() *# creating instance of StandardScaler*

from sklearn.metrics import confusion\_matrix,accuracy\_score,precision\_score,recall\_score

data\_import = pd.read\_csv('winequality-red.csv')

*# Printing whole dataset*

data\_import

*# Printing the Total Row and Column using Shape*

data\_import.shape

*# Printing First Five Rows of dataset using head()*

data\_import.head()

*# Printing Last Five Row of Dataset using tail()*

data\_import.tail()

*# info(): This method prints information about a DataFrame including the index dtype and column dtypes, non-null values*

data\_import.info()

*# describe() : is used to view some basic statistical details like percentile,mean, std, etc*

data\_import.describe()

*#checking for missing values*

data\_import.isnull().sum()

*#good visualization of data*

*# PLotting Histogram for whole dataset using hist()*

data\_import.hist(*figsize*=(17,12),*color*='Pink')

plt.show()

*# Quality is out target variable so we are see its distribution*

data\_import['quality'].hist(*color*='pink')

plt.title('Quality of wine')

plt.show()

*# So changing quality value to True or False Based on if a wine has quality value over 5*

*#(if quality > 5 its is a good wine or else its False)*

y=data\_import['quality']>5

y

*# Removing Target Variable i.e Quality*

X=data\_import.iloc[:,:-1]

X

*#split*

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,*random\_state*=42,*test\_size*=0.2)

*#scale the features*

scaler=StandardScaler()

scaler\_X\_train=scaler.fit\_transform(X\_train)

scaler\_X\_train

*#transform*

scaler\_X\_test=scaler.transform(X\_test)

scaler\_X\_test

*#training the model*

from sklearn.svm import SVC

svc\_clf = SVC(*C*=1.0,

*kernel*='rbf',

*degree*=3,

*gamma*='auto',

*coef0*=0.0, *shrinking*=True,

*probability*=False,

*tol*=0.001, *cache\_size*=200,

*class\_weight*=None,

*verbose*=False, *max\_iter*=-1,

*decision\_function\_shape*='ovr',

*break\_ties*=False,*random\_state*=None)

svc\_clf.fit(scaler\_X\_train,y\_train)

svc\_clf\_predictions=svc\_clf.predict(scaler\_X\_test)

svc\_clf\_predictions

c=confusion\_matrix(y\_test,svc\_clf\_predictions)

a=accuracy\_score(y\_test,svc\_clf\_predictions)

p=precision\_score(y\_test,svc\_clf\_predictions)

r=recall\_score(y\_test,svc\_clf\_predictions)

print('Confusion Matrix:\n',c)

print('Accuracy:',a\*100)

print('Precision:',p\*100)

print('Recall:',r\*100)

*### Overall SVR perform quite well with:*

*### Accuracy: 77.1875*

*### Precision: 81.17647058823529*

*### Recall: 77.09497206703911*

1. **Using Deep feed-forward Neural Network:**

*#THIS File was run in google colab*

import pandas as pd

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

import seaborn as sb

from sklearn.preprocessing import StandardScaler, LabelEncoder

*# Load dataset*

import io

from google.colab import files

data = files.upload()

wine\_data = pd.read\_csv('winequality-red.csv')

wine\_data

display(wine\_data.describe())

wine\_data['quality'].value\_counts()

*#checking for missing data*

wine\_data.isnull().sum()

*#using Using random forrest to analyse the feature importance --- for feature reduction*

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(*random\_state*=1, *max\_depth*=12)

x = wine\_data.drop(['quality'] , *axis* = 1)

wd = pd.get\_dummies(wine\_data)

model.fit(x, wine\_data.quality)

display(model.feature\_importances\_)

features = wd.columns

importances = model.feature\_importances\_

indices = np.argsort(importances)[:]

plt.title('Feature Importances')

plt.barh(range(len(indices)), importances[indices], *color*='b', *align*='center')

plt.yticks(range(len(indices)), [features[i] for i in indices])

plt.xlabel('Relative Importance')

plt.show()

*#Removing least important features*

*#del x['fixed acidity']*

*#del x['free sulfur dioxide']*

*#del x['citric acid']*

x

*#Encoding the quality label*

le = LabelEncoder()

y = le.fit\_transform(wine\_data.iloc[: , -1])

y = pd.DataFrame(y.reshape(len(y),1))

*#Data Over sampling using SMOTE*

from imblearn.over\_sampling import SMOTE

strategy = {0:1700, 1:1700, 2:1700, 3:1700, 4:1700, 5:1700}

oversample = SMOTE(*sampling\_strategy*=strategy)

x, y = oversample.fit\_resample(x, y)

x.shape

*#splitting data*

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, *test\_size* = 0.2, *random\_state* = 0)

*#transforming quality to categorical*

y\_train\_cat = tf.keras.utils.to\_categorical(y\_train, 6)

y\_test\_cat = tf.keras.utils.to\_categorical(y\_test, 6)

*#scaling features*

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.fit\_transform(x\_test)

*#training ML*

ann = tf.keras.models.Sequential(*layers* = None , *name* = None)

ann.add(tf.keras.layers.Input(*shape* = 8,))

ann.add(tf.keras.layers.Dense(*units* = 16 , *activation* = "relu" ))

ann.add(tf.keras.layers.Dense(*units* = 8 , *activation* = "relu" ))

ann.add(tf.keras.layers.Dense(*units* = 6 , *activation* = "sigmoid"))

ann.summary()

ann.compile(*optimizer* = 'adam' , *loss* = 'categorical\_crossentropy' ,*metrics*= ['accuracy'])

history = ann.fit(x\_train, y\_train\_cat, *batch\_size*= 32, *epochs* = 150 , *validation\_data* = (x\_test,y\_test\_cat))

plt.plot(history.history['loss'], *label*='MAE training data')

plt.plot(history.history['val\_loss'], *label*='MAE validation data')

plt.legend()

plt.title('MAE for model')

plt.ylabel('MAE')

plt.xlabel('epoch')

plt.show()

plt.plot(history.history['accuracy'], *label*='Accuracy training data')

plt.plot(history.history['val\_accuracy'], *label*='Accuracy validation data')

plt.legend()

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('epoch')

plt.show()

**Results**

Data Results: These will be shown in the .ipynb files uploaded, after each code block execution. I will summarize the important results for model comparison and performance.

* Random Forest Classifier: We got an accuracy score of 92.8% on the test set as new unseen data. After that, while trying new input data, we predict using this model, and are able to find the quality of wine as good or bad as such:

Text

Description automatically generated

* SVM Classifier: After printing and showing the confusion matrix, we were able to see the spread of TP, TN, FP, FN. Furthermore, an accuracy score of 77.18%, precision of 81.17%, and recall of 77.10% were calculated as in the end of the code shown in “Code and Explanation” for SVM.
* Deep feedforward Neural Network: The model accuracy and the MAE for the model were outputted in the results at the end of the neural network code. The accuracy on the validation set was around 75% as such:

Graphical user interface, chart

Description automatically generated

Model Comparison: Comparing these 3 models based on accuracy, precision, and recall is the appropriate way to select the best model, although all of them can perform relatively well with the wine dataset. It turns out that Random forest classifier for the wine dataset is the best because it has the highest accuracy score at 92.8%, as well as the highest precision and recall values.

**Interpretation**

The topic that was being investigated in the proposal was answered in such a way that with new unseen data or inputs, we can predict whether the wine quality is good or bad, using multiple machine learning algorithms, each having a different performance.