ENEL 682

ML Course Project Report

Author: Ahmad Masri (30115844)

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, and the important ones would be shown in the results and interpretation.
- 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. Comments will be quite informative in terms of explaining the procedure in each .ipynb file.
- How to run the code: Each .ipynb file can be run separately, and the dataset is already in the same path in the folder structure.
- The neural network code was run in google colab, not VS code as the other 2 files.
- Note: I used/referenced multiple parts of my code from samples from the course "examples/assignments" and online research as well which are close to what I want to achieve.
- 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)
# validation metrics
from sklearn.metrics import classification_report
X_test_prediction = model.predict(X_test)
print(classification_report(X_test_prediction, Y_test))

# Hyperparameter tuning using GridSearchCV
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [25, 50, 100, 150],
```

```
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [3, 6, 9],
    'max_leaf_nodes': [3, 6, 9],
}
grid_search = GridSearchCV(RandomForestClassifier(),
                           param_grid=param_grid)
grid_search.fit(X_train, Y_train)
print(grid_search.best_estimator_)

#using the preferable parameters as shown above
model_grid = RandomForestClassifier(max_depth=6,
                                    max_features=None,
                                    max_leaf_nodes=9,
                                    n_estimators=50)
model_grid.fit(X_train, Y_train)
y_pred_grid = model.predict(X_test)
print(classification_report(y_pred_grid, 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")
```

2. 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

#Can perform better
#Trying hyperparameter tuning with GridSearchCV

#Hyperparamter tuning using Gridsearch
from sklearn.model_selection import GridSearchCV
svm = SVC()
param_grid = {'C': [0.1, 1, 10, 100, 1000],
            'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
            'kernel': ['rbf','linear']}
```

```
grid = GridSearchCV(svm,param_grid)
grid.fit(scaler_X_train,y_train)

#Calculating the accuracy of tuned model
from sklearn.metrics import classification_report

grid_svc = grid.predict(scaled_X_test)
accuracy_score(y_test,grid_svc)
#Classification report for the tuned model
print(classification_report(y_test,grid_svc))
#Finally: After tuning, the accuracy, precision, and recall are almost the same as without.
```

3. 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 forest 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: After doing hyperparameter tuning (and even without, the results are really close), and printing the classification report, we got:

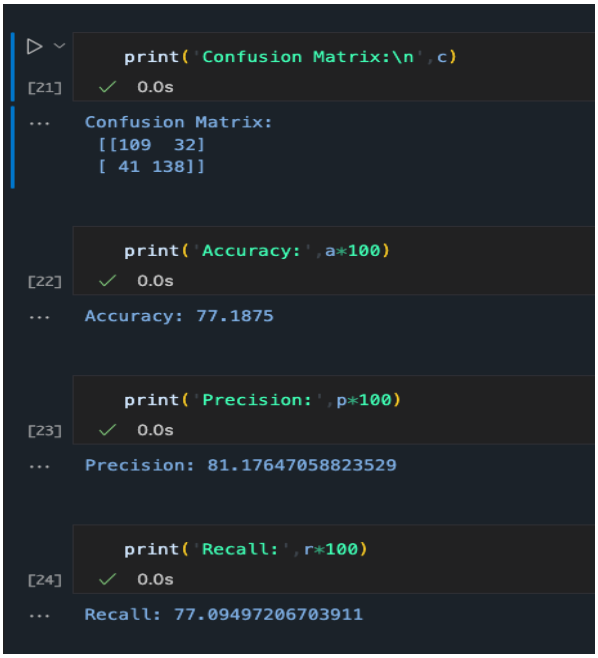
	precision	recall	f1-score	support
0	0.98	0.94	0.96	294
1	0.54	0.77	0.63	26
accuracy			0.93	320
macro avg	0.76	0.86	0.80	320
weighted avg	0.94	0.93	0.93	320

This shows that the RF classifier performs really well because it has high precision, recall, f1-score, and accuracy score (all above 90%).
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:

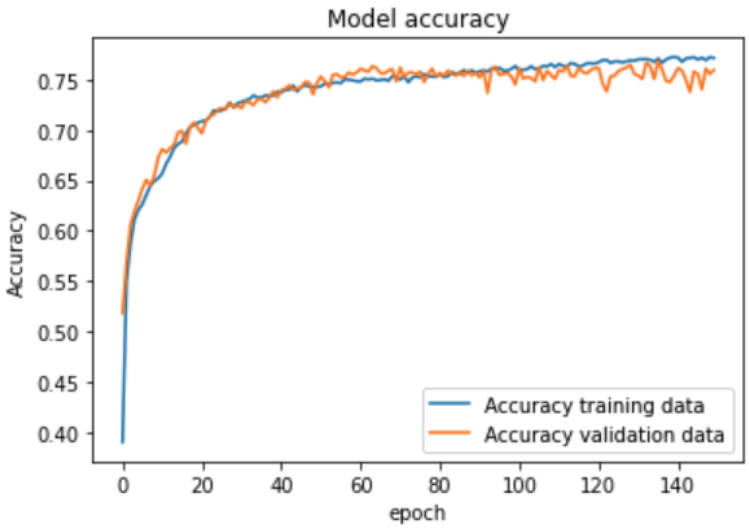
```
#Trying new unseen dummy data in our prediction system
input_data = (6.8,0.220,0.34,3.4,0.031,14.0,16.0,0.92484,3.39,0.92,10.6)
np_input_data = np.asarray(input_data)
reshaped_data = np_input_data.reshape(1,-1)
predict = model.predict(reshaped_data)

if predict[0] == 1:
    print("Good Quality Wine")
else:
    print("Bad Quality Wine")
```

```
Bad Quality Wine
```

- 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. (hyperparameter tuning was not ideal in my code, so I considered the results without the tuning for comparison)



- 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:



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, recall, and f1-score 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.