project2

April 15, 2020

1 Machine Learning in Python - Project 2

Due Wednesday, April 15th by 5 pm.

1.1 1. Setup

1.1.1 1.1 Libraries

[1]:
```
import pkg_resources
if pkg_resources.get_distribution("scikit-learn").version == '0.21.3':
    !pip install --upgrade scikit-learn scipy pandas seaborn
```

[2]:
```
import warnings
warnings.filterwarnings("ignore")
```

[3]:
```
# Display plots inline
%matplotlib inline

# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting defaults
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80

# ipython interactive widgets
from ipywidgets import interact

# sklearn modules
import sklearn
from sklearn.metrics import mean_squared_error
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV, KFold
import sklearn.ensemble
import sklearn.preprocessing
from sklearn.pipeline import make_pipeline
```

1.1.2 1.2 Data

[4]:
```
wine_train = pd.read_csv("wine_qual_train.csv")
wine_test  = pd.read_csv("wine_qual_test.csv")
```

1.2 2. Exploratory Data Analysis and Preprocessing

Figure 1 in the supplementary materials file revealed that there is a big difference between red wine and white wine for all other features. There is a visible separation between the blue and orange clusters in each plot. However, since we only want to create one model, we should split up the wine column into one column where its either red wine or not or vice versa. The type of wine that we choose is arbitrary. Since the wine can only be one of two options, we can convert the wine column into binary such that if it is white wine then it is 1 and if it is red wine, then it is 0.

[5]:
```
wine_train['wine_white'] = wine_train['wine'].isin(['white']).astype(int)
wine_test['wine_white'] = wine_test['wine'].isin(['white']).astype(int)
```

[6]:
```
wine_train = wine_train.drop(['wine'], axis=1)
wine_test = wine_test.drop(['wine'], axis=1)
```

Looking at the correlation plot labeled figure 2 in the supplementary materials file, since our target is predicting the quality of the wine, we want to have high correlations between the features and quality. At the moment, only alcohol has a decent correllation with quality. Hence we have to transform some of the data features to improve the machine learning model. I looked at the distribution plots of each column to see if the data was skewed. I found that 7 data features had positively skewed data. I tested different ways to resolve the skewed data, illustrated in figure 3 in the supp_mat notebook.

Based on the distribution plots in figure 3 of the supp_mat notebook, I decided it was best to log transform 5 of the data features, including chlorides, residual sugar, volatile acidity, sulphates and fixed acidity. I decided to cube root transform the free sulfur dioxide and citric acid. As for density, I decided to leave it alone, because none of the possible transformations for positively skewed data fixed the skew.

[7]:
```
def df_transform(df):
    for col in ['fixed_acidity', 'volatile_acidity', 'residual_sugar',␣
 ↪'chlorides', 'sulphates']:
        df[col] = np.log(df[col])
    for col2 in ['free_sulfur_dioxide', 'citric_acid']:
```

```
        df[col2] = (df[col2])**(1/3)
```

[8]:
```
df_transform(wine_train)
df_transform(wine_test)
```

[9]:
```
def quality_to_cat(df, cat):
    for i in range(len(df)):
        if df[cat][i] >= 7:
            df[cat][i] = 'Excellent'
        elif df[cat][i] == 6:
            df[cat][i] = 'Good'
        elif df[cat][i] == 5:
            df[cat][i] = 'Average'
        elif df[cat][i] <= 4:
            df[cat][i] = 'Poor'

quality_to_cat(wine_train, 'quality')
quality_to_cat(wine_test, 'quality')
```

Another data tranformation that must be made before performing any modelling is converting the quality scores to the quality category which is specified by the table below.

Quality Category	Quality Score
Excellent	≥ 7
Good	6
Average	5
Poor	≤ 4

[10]:
```
# Training data
Xt = wine_train.drop(['quality'], axis=1)
yt = wine_train.quality
# Validation data
Xv = wine_test.drop(['quality'], axis=1)
yv = wine_test.quality
# Complete data
complete = pd.concat([wine_test, wine_train])
X = complete.drop(['quality'], axis=1)
y = complete.quality
```

1.3 3. Model Fitting and Tuning

Helper Functions For Testing Accuracy of Model

[11]:
```
def model_test_assess(model):
    # Use the model to predict test labels
    res = wine_test.copy()
```

```
    res["pred_label"] = model.predict(wine_test.drop(['quality'], axis=1))

    # Print report
    print(
        sklearn.metrics.classification_report(res.quality, res.pred_label)
    )
```

[12]:
```
qualities = ['Poor', 'Average', 'Good', 'Excellent']
```

1.3.1 Random Forest

I tested many different models including Random Forest Classifiers, Decision Tree Classifiers, nearest neighbors, logistic regression testing both ovr and multiclass as the multiclass, and support vector machines. After testing all of these models, I found that Random Forest Classifiers was the most accurate model with the least number of false positives and least number of false negatives.

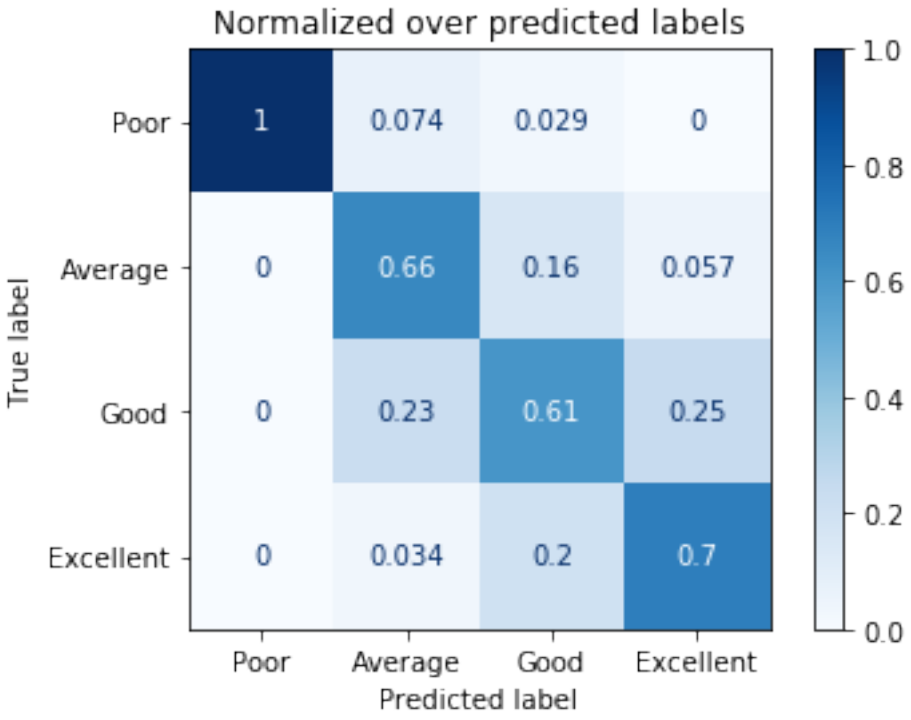
[13]:
```
@interact(n_estimators = (1,50), max_depth=[None] + list(np.arange(1,50,5)))

def fit_rf(n_estimators, max_depth):
    m_rf = sklearn.ensemble.RandomForestClassifier(
        n_estimators=n_estimators, max_depth=max_depth
    ).fit(Xt,yt)

    res = wine_test.copy()
    res["pred_label"] = m_rf.predict(wine_test.drop(['quality'], axis=1))

    print(
        sklearn.metrics.classification_report(res.quality, res.pred_label)
    )
```

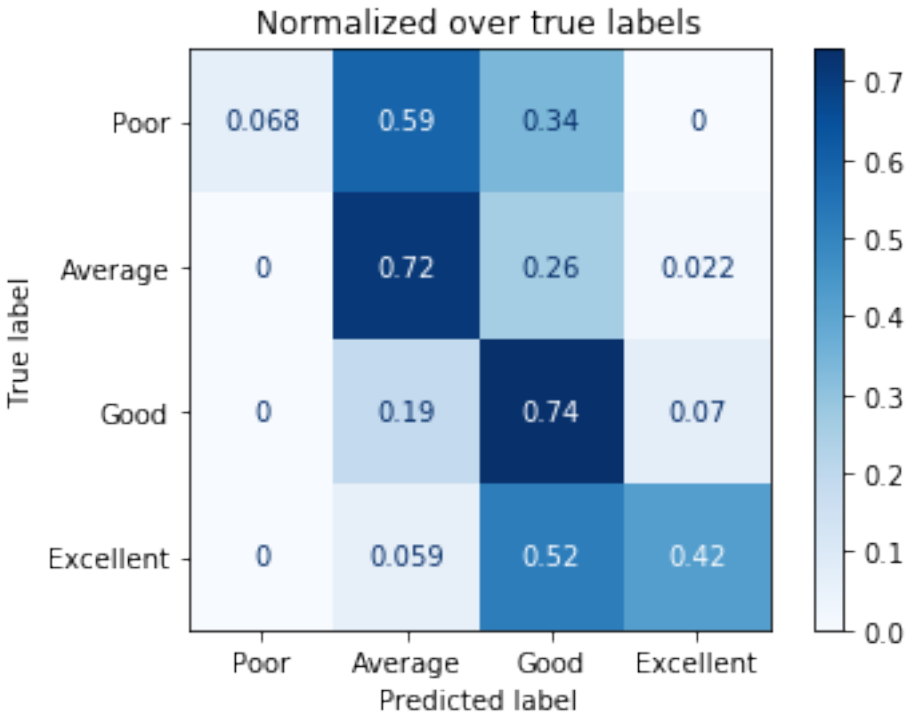
```
interactive(children=(IntSlider(value=25, description='n_estimators', max=50, min=1), Dropdown
```

[14]:
```
param = {'n_estimators':(1,50), 'max_depth':[None] + list(np.arange(1,50,5))}
# Used Grid Search to find the best parameters for the random forest classifier␣
 ↪model

m_rf = GridSearchCV(
    sklearn.ensemble.RandomForestClassifier(),
    param_grid=param,
    cv=5,).fit(Xt, yt)

print( "best n_estimators:", m_rf.best_params_['n_estimators'])
print( "best max_depth:", m_rf.best_params_['max_depth'])
```

```
best n_estimators: 50
best max_depth: 26
```

[16]:
```
m_rf = sklearn.ensemble.RandomForestClassifier(
        n_estimators=50, max_depth=26
    ).fit(Xt,yt)
disp = plot_confusion_matrix(m_rf, Xv, yv,labels=qualities, cmap=plt.cm.
 ↪Blues,normalize='pred')
disp.ax_.set_title('Normalized over predicted labels')
disp = plot_confusion_matrix(m_rf, Xv, yv, labels=qualities,cmap=plt.cm.
 ↪Blues,normalize='true')
disp.ax_.set_title('Normalized over true labels')
plt.show()
```





1.4 4. Discussion

Tree based algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based algorithms empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. For modelling the quality of different wines, the Random Forest Classifier for tree based algorithms worked quite well. The algorithm is an extension of the decision tree frame work whereby a number decision tree models are fit to random sub-sample of the features. When making predictions each tree predicts a label and the most common label across all of the trees is used as the final prediction.

Using my processed data, and using grid search to find the best paramaters to use for different models, I found that the Random Forest classifier had the highest accuracy of around 65%. Fitting the model to the training data and predicting the quality of the wine for the test data, the random forest classifier accurately predicted wines as their respective qualities with the probabilites in the table below. The model did not have any poor false-positives. The confusion matrices above reaveal how accurate my model was. The first matrix normalized over the predicted columns is good for comparing false negatives, and the second matrix normalized over the true labels is good for comparing false positives.

Quality Category	Ratio of True Positives	AUC
Excellent	0.42	0.46
Good	0.74	0.46
Average	0.71	0.86

Quality Category	Ratio of True Positives	AUC
Poor	0.11	0.36

Compared with the some of the other models I tested, Random Forest Classifiers has the highest overall accuracy and lowest number of false positives and false negatives.

Model	Overall Accuracy	Percentage Mislabeled
Random Forest Classifier	65 %	24.1 %
Decision Tree Classifier	56 %	45.4 %
Nearest Neighbors	55 %	44.5 %
Logistic Regression (ovr)	53 %	32.5 %
Logistic Regression (multiclass)	53 %	34.0 %
Support Vector Machines	58 %	65.5 %

The Random Forest Classifier is the most accurate overall. Regarding the types of errors that tend to occur with this model, they do not incur immense losses by the missclassification. 2.5% of excellent wines are mislabeled as poor, and 56% of excellent wine are mislabeled as good. The confusion table normalized by the true labels reveals that good wines are mislabelled as average 22% of the time and excellent 7.3% of the time, average wines are predicted to be Good 25% of the time and poor 1.5% of the time. 27% of poor wines are mislabeled as good, 59% mislabeled as average, and 2.3% as excellent. The Random Forest Classifier also has the least amount of wines that are mislabeled compared with the other models that I tested. This model also has a high AUC for each quality, and very high true positive rate for average quality wines. This is illustrated in the figure 5: ROC plot, located in the supp_mat notebook.

1.5 5. Model Validation

[17]:
```
wine_holdout = pd.read_csv("wine_qual_holdout.csv")

# Data Preprocessing
wine_holdout['wine_white'] = wine_holdout['wine'].isin(['white']).astype(int)
wine_holdout = wine_holdout.drop(['wine'], axis=1)
df_transform(wine_holdout)
quality_to_cat(wine_holdout, 'quality')

X_holdout = wine_holdout.drop('quality', axis=1)
y_holdout = wine_holdout.quality
```

[18]:
```
param = {'n_estimators':(1,50), 'max_depth':[None] + list(np.arange(1,50,5))}

m_rf = GridSearchCV(
    sklearn.ensemble.RandomForestClassifier(),
    param_grid=param,
    cv=5,).fit(X, y)
```

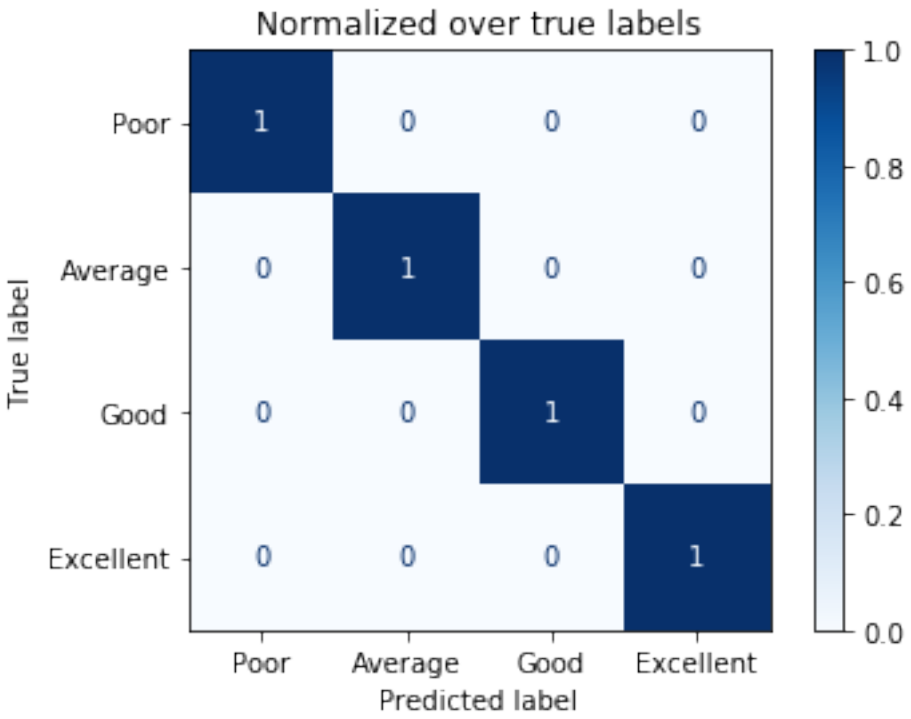
```
n = m_rf.best_params_['n_estimators']
maxd = m_rf.best_params_['max_depth']
```

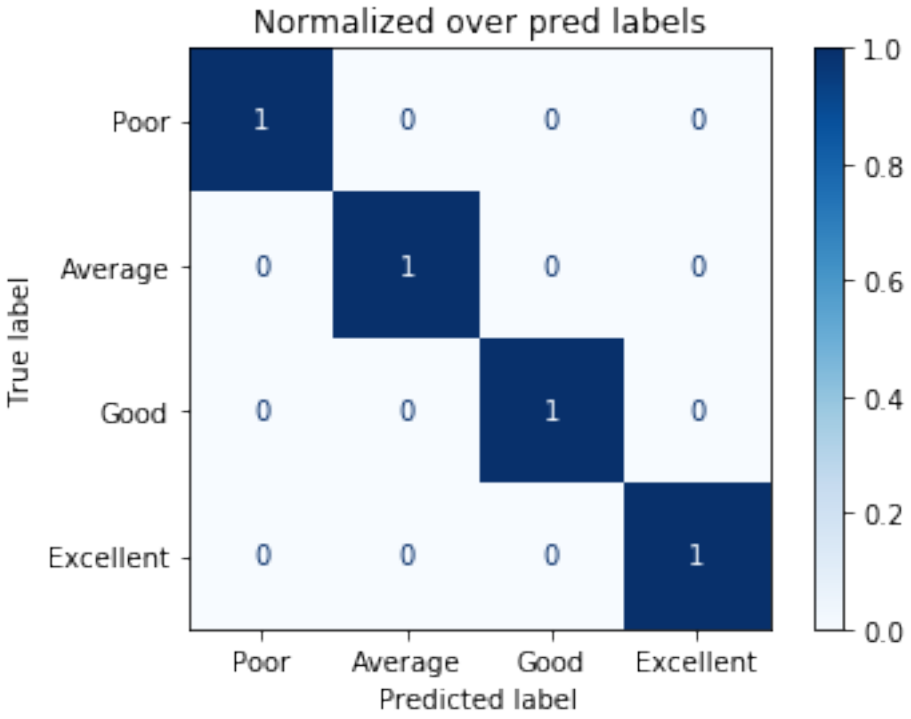
[19]:
```
m_rf = sklearn.ensemble.RandomForestClassifier(
        n_estimators=n, max_depth=maxd).fit(X,y)
```

[20]:
```
sklearn.metrics.confusion_matrix(y_holdout, m_rf.predict(X_holdout))
```

[20]:
```
array([[325,   0,   0,   0],
       [  0, 202,   0,   0],
       [  0,   0, 429,   0],
       [  0,   0,   0,  44]])
```

[21]:
```
# Plot non-normalized confusion matrix
titles_options = [("Normalized over true labels", 'true'),
                  ("Normalized over pred labels", 'pred')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(m_rf, X_holdout, y_holdout,
                                 display_labels=qualities,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)
plt.show()
```





[22]:
```
print(
    sklearn.metrics.classification_report(y_holdout, m_rf.predict(X_holdout))
)
```

```
              precision    recall  f1-score   support

     Average       1.00      1.00      1.00       325
   Excellent       1.00      1.00      1.00       202
        Good       1.00      1.00      1.00       429
        Poor       1.00      1.00      1.00        44

    accuracy                           1.00      1000
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000
```

My random forest classifier model is 65% accurate. It was the most accurate model out of all the models that I tested. There are some implications in the uncertainty, since an excellent quality wine could be categorized as average or good quality. Similar implications for all other categories. However, compared to all the other models, random forest classifier had the least amount of false positives and false negatives, making it the best model. If this model is used to categorize wines by their chemical features and then put prices on those wines based on the quality that my model gives it, then the price of the wine will only be accurate 65 percent of the time.