Final Project Report

Wenhao Chen

Introduction

Red wine is one of the most popular alcohol in the world. The quality of wine on the market varies and we all want to know how to identify the quality of red wine. The goal of this project is to judge the quality of red wine by using the data of composition and physical characteristics of red wine.

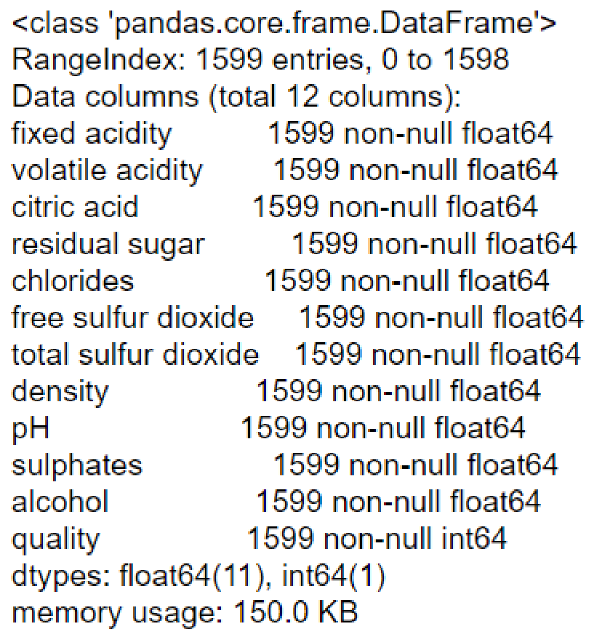
The dataset is downloaded from Kaggle (https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009). Following shows the variables of this dataset:

Input variables (based on physicochemical tests):
1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 - alcohol
12 - quality (score between 0 and 10)

In order to achieve this goal of this project, I will build several machine learning algorithms (Python code and sklearn package) and compare those methods to find a best one since each algorithm has its inherent biases.

Data Preprocessing

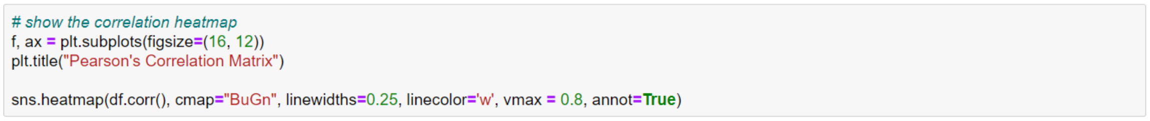
Let’s check the information of the dataset first:



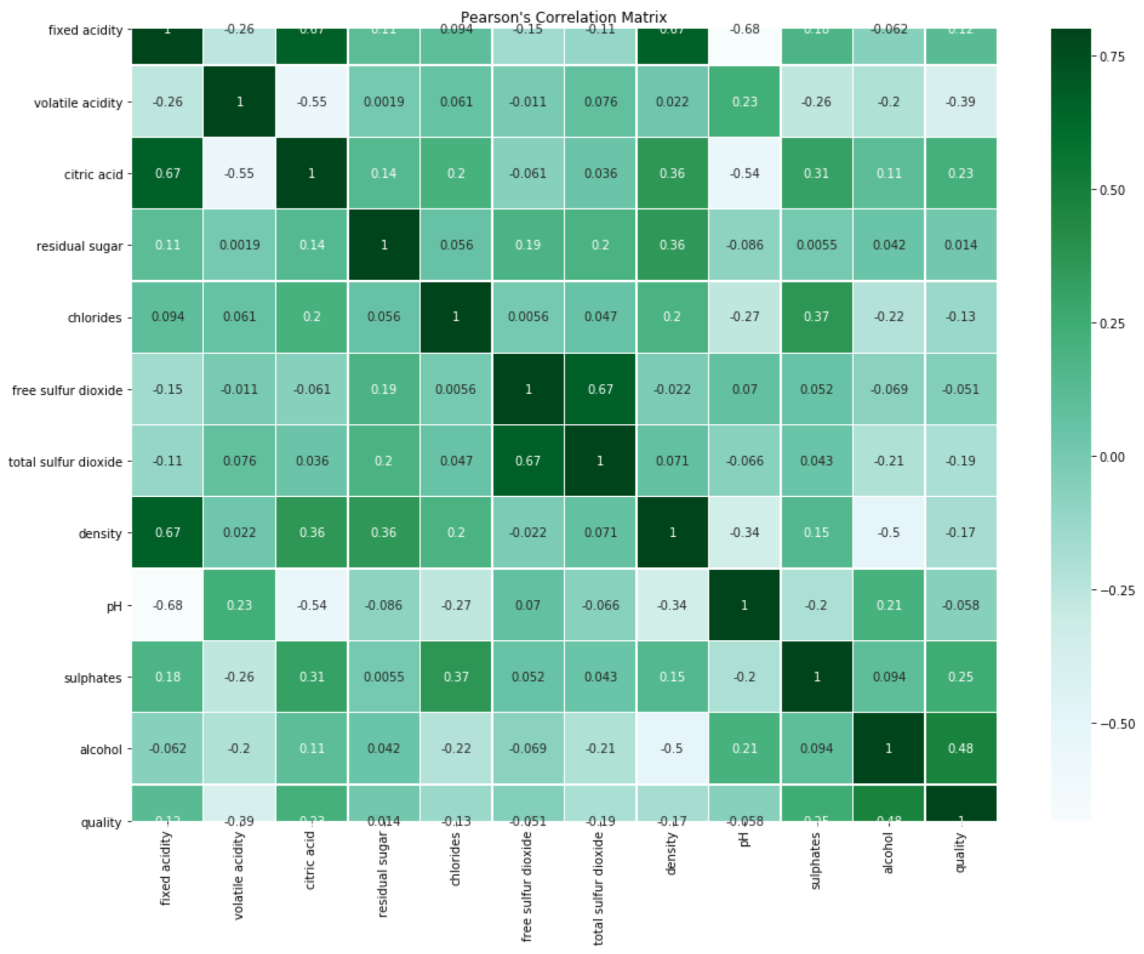
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
fixed acidity           1599 non-null float64
volatile acidity        1599 non-null float64
citric acid             1599 non-null float64
residual sugar          1599 non-null float64
chlorides               1599 non-null float64
free sulfur dioxide     1599 non-null float64
total sulfur dioxide    1599 non-null float64
density                 1599 non-null float64
pH                      1599 non-null float64
sulphates               1599 non-null float64
alcohol                 1599 non-null float64
quality                 1599 non-null int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

We can see that there is no null value in this dataset and all the variables are numerical.

Let’s check the correlation between the features by plotting heatmap:



```
# show the correlation heatmap
f, ax = plt.subplots(figsize=(16, 12))
plt.title("Pearson's Correlation Matrix")

sns.heatmap(df.corr(), cmap="BuGn", linewidths=0.25, linecolor='w', vmax = 0.8, annot=True)
```



The heatmap plot shows that there are some variables which are strongly correlated. Also, since this is a high-dimensional dataset, I decide to standardize the data and use PCA algorithm to reduce the dimension of the data.

The standardization and PCA method are written in pipeline when building models below.

Data splitting

The next step is split our data, Since the dataset is not big, cross validation is appropriate for evaluating our models. Therefore, I decide to split the data into training data (70%) and test data (30%).

```
X, y = df.drop(['quality'], axis=1), df['quality']

X_train, X_test, y_train, y_test =\
    train_test_split(X, y,
                     test_size=0.3,
                     random_state=2020,
                     stratify=y)
```

Model Building & Model Evaluation

We can regard the outcome variable (quality) as a numerical variable first and perform multiple linear regression model:

```
pipe_lm = make_pipeline(StandardScaler(),
                        PCA(n_components=0.9),
                        LinearRegression())

pipe_lm.fit(X_train, y_train)

print('Training Accuracy: %.3f' % pipe_lm.score(X_train, y_train))
print('Test Accuracy: %.3f' % pipe_lm.score(X_test, y_test))
```

```
Training Accuracy: 0.353
Test Accuracy: 0.314
```

The result shows that the accuracy of multiple linear regression model is low. Thus, it is not appropriate to consider the outcome variable (quality) as a numerical variable. Let's consider it as categorical variable, and then, use the classification algorithms.

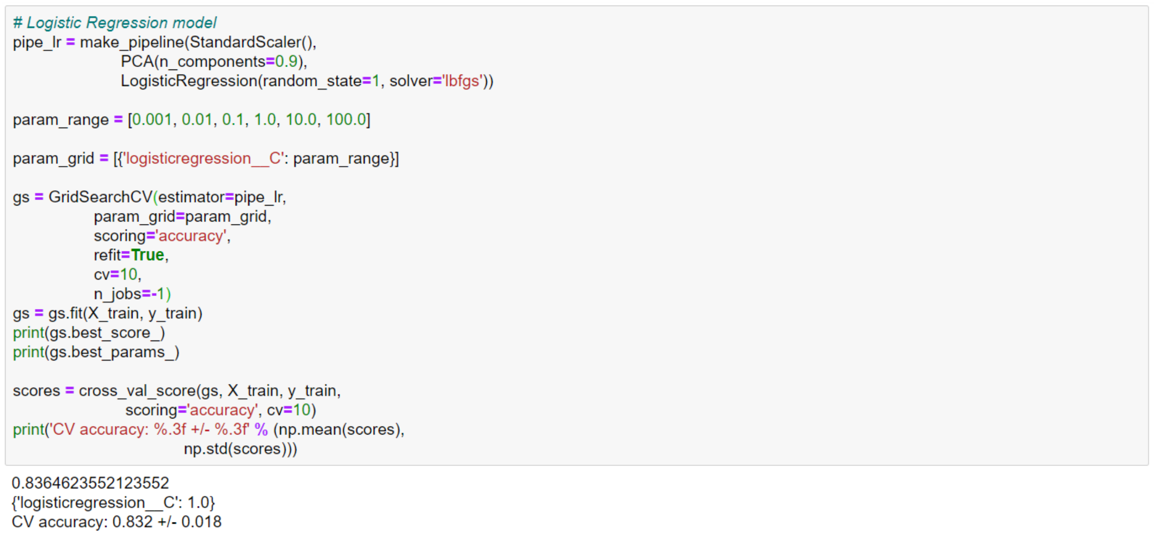
Let's rescale the outcome variable first:

```
df.loc[(df['quality']==3),'quality']=1
df.loc[(df['quality']==4),'quality']=1

df.loc[(df['quality']==5),'quality']=2
df.loc[(df['quality']==6),'quality']=2

df.loc[(df['quality']==7),'quality']=3
df.loc[(df['quality']==8),'quality']=3

X, y = df.drop(['quality'], axis=1), df['quality']

X_train, X_test, y_train, y_test =\
    train_test_split(X, y,
                     test_size=0.3,
                     random_state=2020,
                     stratify=y)
```

I rescale it to three types: good, fair and bad, which are represented by 3, 2 and 1.

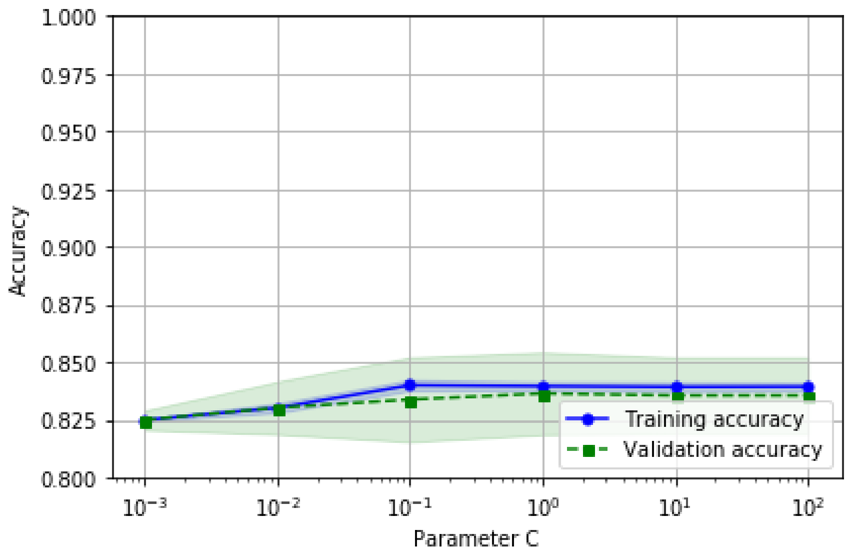
The model I used are Logistic Regression model, Decision Tree model, SVM model and K-nearest Neighbors model. Let's look at the performance of those models.

■ Logistic Regression model


```
# Logistic Regression model
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=0.9),
                        LogisticRegression(random_state=1, solver='lbfgs'))

param_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]

param_grid = [{'logisticregression__C': param_range}]

gs = GridSearchCV(estimator=pipe_lr,
                  param_grid=param_grid,
                  scoring='accuracy',
                  refit=True,
                  cv=10,
                  n_jobs=-1)
gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)

scores = cross_val_score(gs, X_train, y_train,
                         scoring='accuracy', cv=10)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),
                                      np.std(scores)))
```

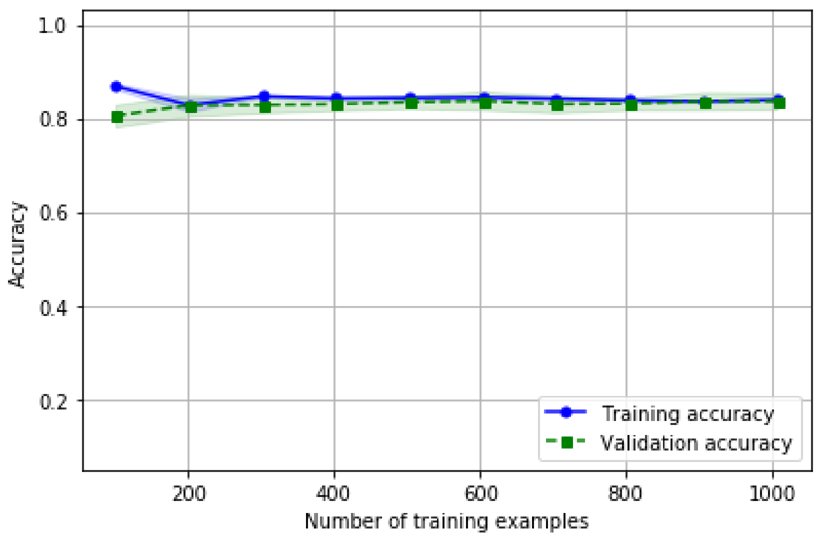
```
0.8364623552123552
{'logisticregression__C': 1.0}
CV accuracy: 0.832 +/- 0.018
```

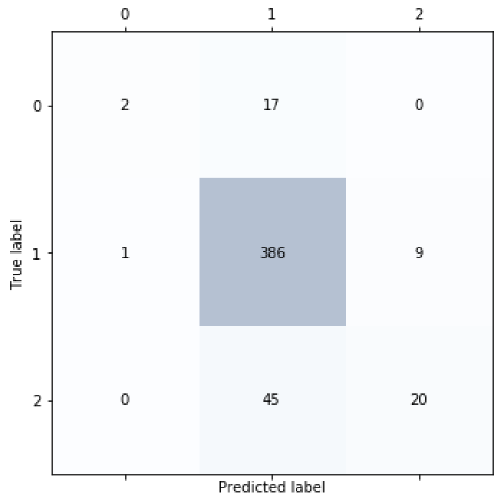
I use grid search with 10-fold cross-validation to find the hyperparameter of the model and the best parameter C is 1.0. Let’s find the hyperparameter by using validation curve.



We can find the same conclusion to grid search.

Let’s look at the performance of the model by plotting learning curve and confusion matrix.





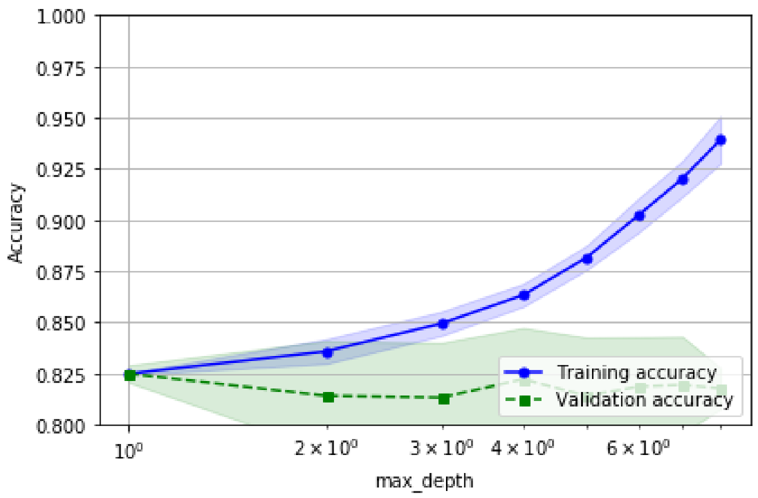
The performance of Logistic Regression model is good!

I will find hyperparameters and evaluate them in similar way.

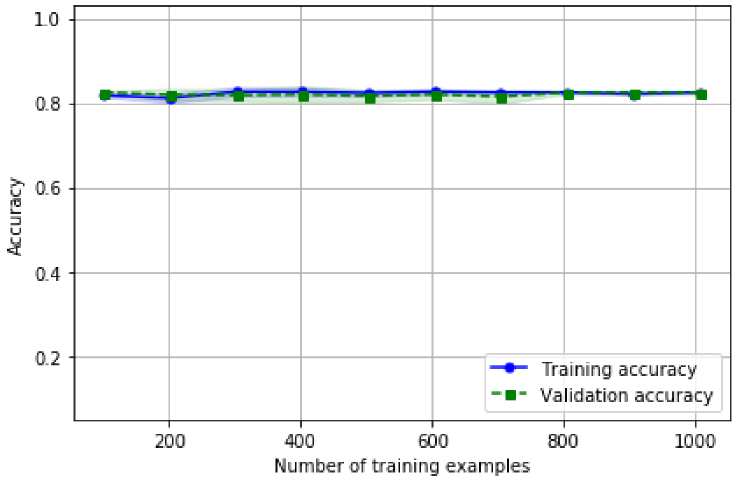
■ Decision Tree model

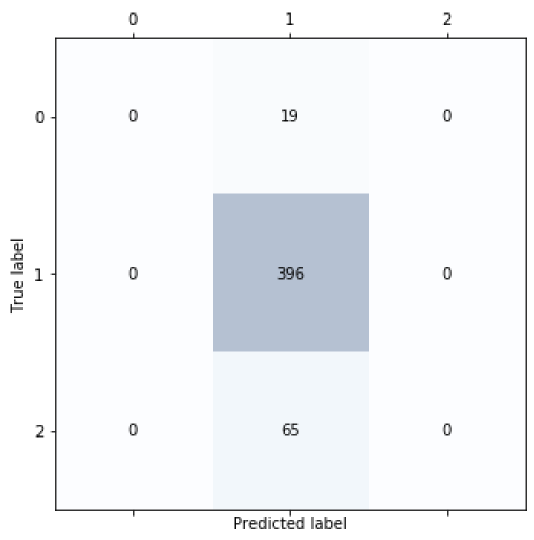
```
# Decision Tree model
pipe_dt = make_pipeline(StandardScaler(),
                        PCA(n_components=0.9),
                        DecisionTreeClassifier(criterion='gini',
                            max_depth=8))

param_grid = [{'decisiontreeclassifier__max_depth': [1, 2, 3, 4, 5, 6, 7, 8]}]

gs = GridSearchCV(estimator=pipe_dt,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=10)
gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)

scores = cross_val_score(gs, X_train, y_train,
                         scoring='accuracy', cv=10)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),
                                      np.std(scores)))
```

```
0.8248471685971686
{'decisiontreeclassifier__max_depth': 1}
CV accuracy: 0.811 +/- 0.031
```



Both grid search and validation curve show that the best max depth is 1.



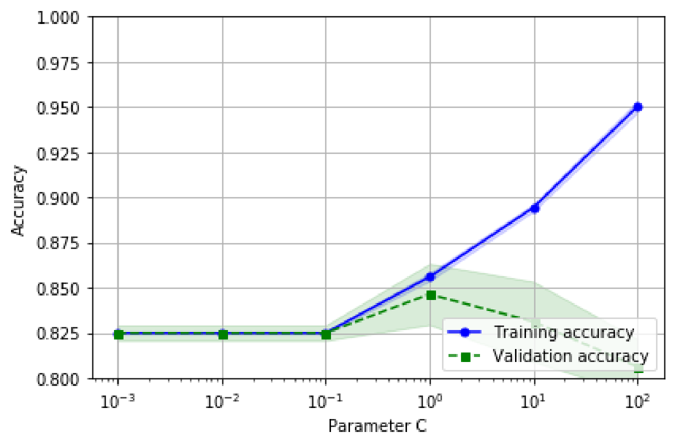


Still no bias but it is worse than Logistic Regression model.

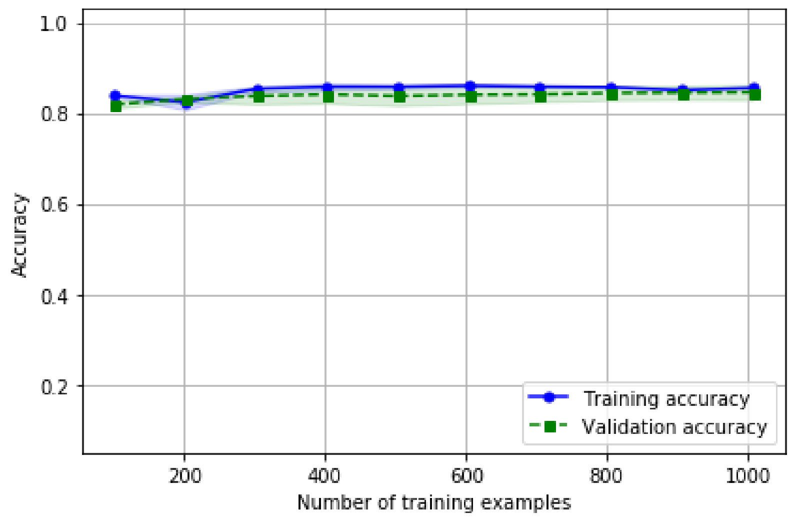
- SVM model

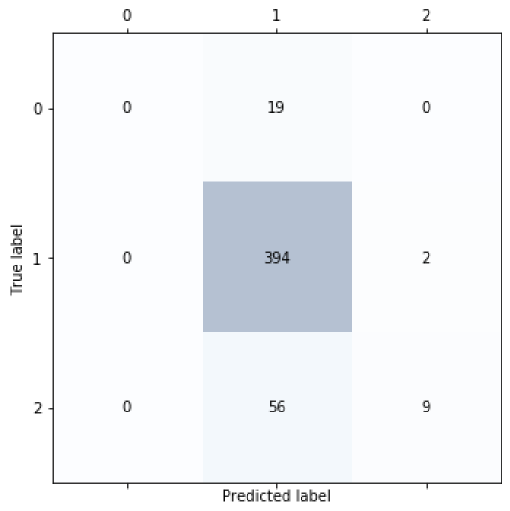
```
# SVM model
pipe_svc = make_pipeline(StandardScaler(),
                         PCA(n_components=0.9),
                         SVC())

param_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
param_grid = [{'svc__C': param_range}]

gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=10)
gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)

scores = cross_val_score(gs, X_train, y_train,
                         scoring='accuracy', cv=10)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),
                                      np.std(scores)))
```

```
0.8462837837837839
{'svc__C': 1.0}
CV accuracy: 0.846 +/- 0.017
```



The best parameter C is 1.



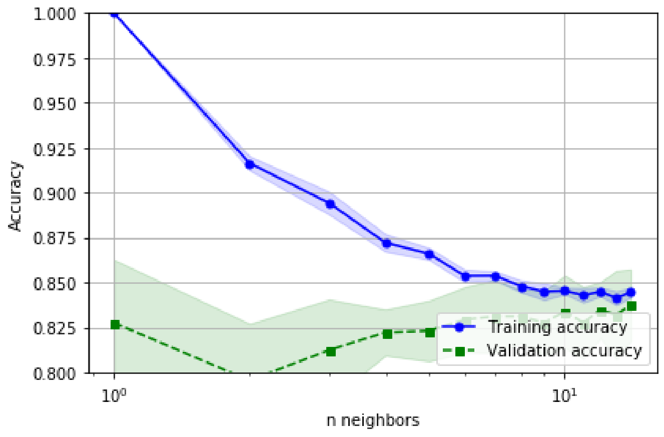


Also not overfitting and performs well. It is better than Logistic Regression.

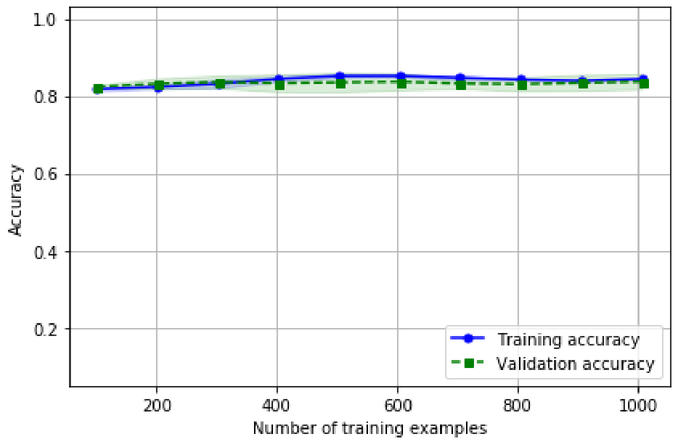
- K-nearest Neighbors model

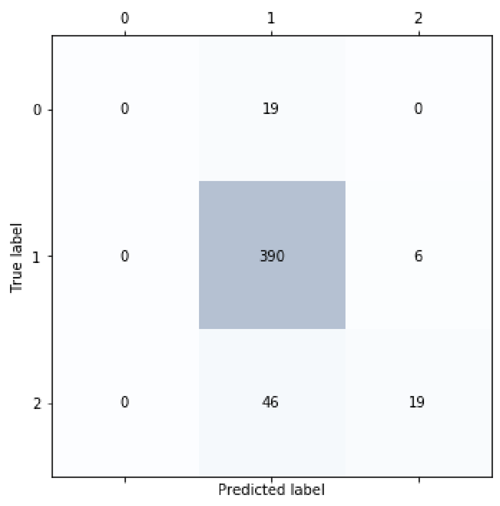
```
# K-nearest Neighbors model
pipe_knn = make_pipeline(StandardScaler(),
                PCA(n_components=0.9),
                KNeighborsClassifier(n_neighbors=10))

param_grid = [{'kneighborsclassifier__n_neighbors': [i for i in range(30)]}]

gs = GridSearchCV(estimator=pipe_knn,
            param_grid=param_grid,
            scoring='accuracy',
            cv=10)
gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)

scores = cross_val_score(gs, X_train, y_train,
                scoring='accuracy', cv=10)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),
                np.std(scores)))
```

```
0.8373471685971687
{'kneighborsclassifier__n_neighbors': 14}
CV accuracy: 0.829 +/- 0.022
```



The best n_neighbors is 14.





The performance is good, but SVM is better.

Conclusion

After the model building and evaluation, we find the best model is SVM (support vector machine) model. The accuracy of the model is good, meaning that we can use the model to judge whether the red wine is good when giving such composition and features.