Red Wine Quality- Machine Learning with Python

January 15, 2022

0.0.1 Introduction

The goal of this exercise is to classify the red wines listed in the data set into “good” or “bad wines. This will be done by using the”quality” field, which gives each wine a rating from 1 to 8. The red wines that are “bad” will have a quality of 6 or below, whereas the “good” wines will have a quality of 7 or 8. The goal is to generate a list of red wines that we want to taste. Once we transform the quality field, we will create three different models using machine learning algorithms in the *scikit-learn* library from Python. This will be done by splitting the data into **training** and **testing**. The **training** data will be used to create the models, and the **testing** data will tell us how well these models hold up aaginst the actual wine qualities. We want to see which algorithm best predicts the classification of the wine quality.

Necessary Library Installations:

```
[1]: import pandas as pd
     import seaborn as sns #added features from matplotlib
     import matplotlib.pyplot as plt
     from sklearn import svm
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.svm import SVC #support vector classifier
     from sklearn.neural_network import MLPClassifier
     from sklearn.linear_model import SGDClassifier
     from sklearn.metrics import confusion_matrix, classification_report #metrics
     from sklearn.preprocessing import StandardScaler, LabelEncoder
     from sklearn.model_selection import train_test_split #split into test and train␣
      ↪data
     %matplotlib inline
```

Before we begin the model-constructing process, we must be sure the data is clean and without null values:

```
[14]: redwine = pd.read_csv('/Users/andrewlevine/Downloads/redwinequality.csv')
      redwine.info()
      redwine.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
```

```
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

[14]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Thankfully for us, there are no nulls in the data set. Had there been some, we could attack this issue by omitting the records with null data, or by replacing these values with a measure of central tendency. Alternatively we could build a regression model to predict these null values based on the response variable and predictors for that record.

Now, we can proceed with transforming the data to build for the models. We will use the binning technique to replace the numeric values in "quality" with "good" and "bad" based on their value:

[15]:
```
#Preprocessing data:
bins = (2, 6.5, 8)
group_names = ['bad', 'good']
redwine['quality'] = pd.cut(redwine['quality'], bins = bins, labels =␣
 ↪group_names)
redwine['quality'].unique()
```

[15]:
```
['bad', 'good']
Categories (2, object): ['bad' < 'good']
```

We are replacing the "quality" field with **two** bins named "good" and "bad". The cutoff for these bins is set at **6.5**; this means that any wine whose quality score is less than 6.5 (1 to 6) will be classified as a “bad wine”, and any wine whose quality score is above 6.5 (7 to 8) will be classified as a “good wine”.

Take a look at the updated “quality” column; notice how the values changed from 1 to 8 to 0 or 1. “0” denotes a bad wine, and “1” denotes a good wine.

[16]:
```
label_quality = LabelEncoder()
redwine['quality'] = label_quality.fit_transform(redwine['quality'])
redwine.head()
```

[16]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        0
1      9.8        0
2      9.8        0
3      9.8        0
4      9.4        0
```

The first five attributes are shown above. So far, we do not observe any good wines.

How many good wines are there in the data set? How many bad wines are there?

[17]:
```
redwine['quality'].value_counts()
```

[17]:
```
0    1382
1     217
Name: quality, dtype: int64
```

There are 1382 bad wines, compared to just 217 good wines. That does not sound promising for wine enthusiasts!

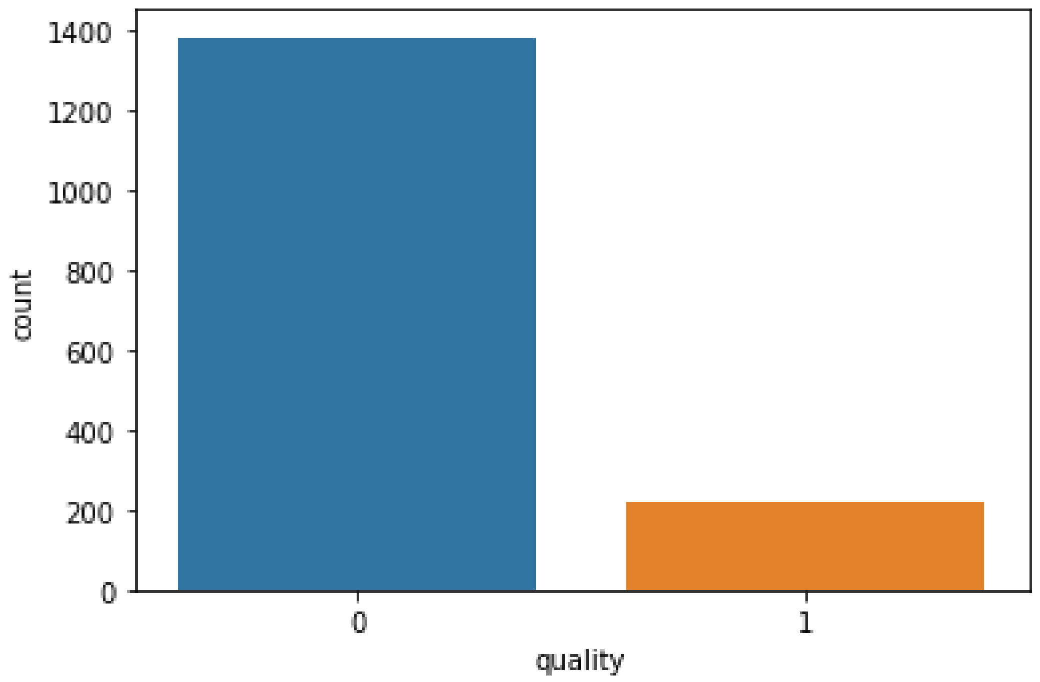
Alternatively, we can look at this discrepency using a bar chart:

[21]:
```
sns.countplot(redwine['quality'])
```

```
/Users/andrewlevine/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable
```

```
as a keyword arg: x. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

[21]:
```
<AxesSubplot:xlabel='quality', ylabel='count'>
```



It is now time to build the model! First, we will initialize the variables. Since we are trying to predict whether a wine quality is good or bad, this will be our y-value. Everything else will be a predictor/explanatory variable, so it will remain in the X.

[22]:
```
X = redwine.drop('quality', axis = 1)
y = redwine['quality']
```

Now we will *randomly* split the data into training and testing parts. The training part is the data used to create the models. It is used to teach the machine learning algorithm to detect any patterns/relationships between the explanatory variables and response variable in the data set. The testing part is the data used to see how well the model holds up. How accurate did it perform? The testing size for this data set will be 20%, which means 80% of the data will be selected at random to be included in the training.

[23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
```

Additionally, we need to scale the data. Some values for certain predictors are in the 100s, while others are in the 0.01's. StandardScaler() scales the data, so the means

are at 0 with a standard deviation of 1.

[28]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

0.0.2 Random Forest Classifier

The first algorithm we will use to test the data is the random forest classifier. We will run it 200 times, which means it has 200 "trees" in the forest. We usually start high, and try to bring this number lower to see if the model gets worse.

[30]:
```
rfc_wine = RandomForestClassifier(n_estimators = 200)
rfc_wine.fit(X_train, y_train)
predict_rfc_wine = rfc_wine.predict(X_test)
```

How does our model perform? We will look at the classification report and confusion matrix to see:

[32]:
```
print(classification_report(y_test, predict_rfc_wine))
print(confusion_matrix(y_test, predict_rfc_wine))
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.95       273
           1       0.76      0.53      0.62        47

    accuracy                           0.91       320
   macro avg       0.84      0.75      0.79       320
weighted avg       0.90      0.91      0.90       320

[[265   8]
 [ 22  25]]
```

Additionally, we can run these lines to find the accuracy:

[36]:
```
from sklearn.metrics import accuracy_score
precision_rfc = accuracy_score(y_test, predict_rfc_wine)
print(precision_rfc)
```

```
0.90625
```

The overall accuracy of this model is 90.625%, which means it classified the wines correctly 90.625% of the time. Broken down further, the accuracy for bad wines was 92%, and the accuracy for good wines was 76%.

0.0.3 Support Vector Model

The next algorithm we will use to test the data is the support vector model classifier. It is used better for smaller data sets and for raw data (no binning, which is what we did here).

[33]:
```
clf_wine = svm.SVC()
clf_wine.fit(X_train, y_train)
predict_clf_wine = clf_wine.predict(X_test)
```

How does our model perform? We will look at the classification report and confusion matrix to see:

[35]:
```
print(classification_report(y_test, predict_clf_wine))
print(confusion_matrix(y_test, predict_clf_wine))
```

```
              precision    recall  f1-score   support

           0       0.88      0.98      0.93       273
           1       0.71      0.26      0.37        47

    accuracy                           0.88       320
   macro avg       0.80      0.62      0.65       320
weighted avg       0.86      0.88      0.85       320

[[268   5]
 [ 35  12]]
```

Again, we can use the following line of code to find this value:

[37]:
```
precision_clf = accuracy_score(y_test, predict_clf_wine)
print(precision_clf)
```

```
0.875
```

The overall accuracy of this model is 87.5%, which means it classified the wines correctly 87.5% of the time. Broken down further, the accuracy for bad wines was 88%, and the accuracy for good wines was 71%. It did not perform as well as the random forest classifier did, but this was expected, due to the reasons described above.

0.0.4 Neural Network Model

The final algorithm we will use to test the data is the neural network model. It is best used for large cohorts of data, and is fantastic for text data as well. We will use three hidden layers to build this model, with sizes of 11 for each (there are 11 explanatory variables used in the model, so it is a fine number to use). We will pass the data through the algorithm 500 times.

[38]:
```
mlpc_wine = MLPClassifier(hidden_layer_sizes = (11,11,11), max_iter = 500)
mlpc_wine.fit(X_train, y_train)
predict_mlpc_wine = mlpc_wine.predict(X_test)
```

```
/Users/andrewlevine/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

How does our model perform? We will look at the classification report and confusion matrix to see:

[40]:
```python
print(classification_report(y_test, predict_mlpc_wine))
print(confusion_matrix(y_test, predict_mlpc_wine))
```

```
              precision    recall  f1-score   support

           0       0.94      0.93      0.94       273
           1       0.64      0.68      0.66        47

    accuracy                           0.90       320
   macro avg       0.79      0.81      0.80       320
weighted avg       0.90      0.90      0.90       320

[[255  18]
 [ 15  32]]
```

Again, we can use the following line of code to find this value:

[44]:
```python
precision_mlpc = accuracy_score(y_test, predict_mlpc_wine)
print(precision_mlpc)
```

```
0.896875
```

The overall accuracy of this model is 89.7%, which means it classified the wines correctly 89.7% of the time. Broken down further, the accuracy for bad wines was 94%, and the accuracy for good wines was 64%. It did a great job at predicting bad wines, but an awful one at predicting good wines.

The last thing we will do here is make a wine by filling in random values for each predictor. From there we will use each algorithm to see if it is a bad or a good wine.

[136]:
```python
Xnew = [[7.4, 0.88, 0.01, 4.1, 0.065, 18.0, 43, 0.9978, 3.31, 0.50, 9.44]]
Xnew_values = [7.4, 0.88, 0.01, 4.1, 0.065, 18.0, 43, 0.9978, 3.31, 0.50, 9.44,␣
 ↪"?"]
Xnew = sc.transform(Xnew)
yrfc_new = rfc_wine.predict(Xnew)
yclf_new = clf_wine.predict(Xnew)
ymlpc_new = mlpc_wine.predict(Xnew)
for ind, col in enumerate(redwine.columns):
    print(col, ":", Xnew_values[ind])
if yrfc_new == 1:
    print("The random forest algorithm predicts my wine to be good.")
else:
    print("The random forest algorithm predicts my wine to be bad.")
if yclf_new == 1:
    print("The support vector model predicts my wine to be good.")
else:
    print("The support vector model predicts my wine to be bad.")
```

```python
if ymlpc_new == 1:
    print("The neural network model predicts my wine to be good.")
else:
    print("The neural network model predicts my wine to be bad.")
```

```
fixed acidity : 7.4
volatile acidity : 0.88
citric acid : 0.01
residual sugar : 4.1
chlorides : 0.065
free sulfur dioxide : 18.0
total sulfur dioxide : 43
density : 0.9978
pH : 3.31
sulphates : 0.5
alcohol : 9.44
quality : ?
The random forest algorithm predicts my wine to be bad.
The support vector model predicts my wine to be bad.
The neural network model predicts my wine to be bad.
```

Each of the algorithms predicted the wine I created to be a bad wine. Guess I won't quit my day job…