Final_Project

December 9, 2019

1 Final Project_STP 598 Topic: Machine Learning / Statistical Learning

Team Members: Chi Duan, Huai Shen and Chengyu hong

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from itertools import combinations
import matplotlib.pyplot as plt
from nltk.tokenize import RegexpTokenizer
```

[2]:
```
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.svm import SVC
from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_squared_error, accuracy_score, f1_score,␣
↪confusion_matrix
```

[3]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, MaxPooling1D, Conv1D
```

```
Using TensorFlow backend.
```

[4]:
```
# Wine data cleaning
def wine_data_cleaning(wine_data):
    cleaned_data = []
    column_name = wine_data.columns.values[0].split(';')
    for i in range(len(column_name)):
        column_name[i] = column_name[i].strip('"')
    for i in range(len(wine_data)):
        cleaned_data.append(list(map(float, wine_data.iloc[i].values[0].split(';
↪'))))
```

```
    cleaned_data = pd.DataFrame(cleaned_data, columns=column_name)
    return cleaned_data
```

[5]:
```
def merge_quality(y,label_to_merge=[5,6]):
    for i in range(len(y)):
        if y[i] in label_to_merge:
            y[i]=label_to_merge[0]
    return y
```

2 Red Wine Data

[6]:
```
# Read the Red wine data
red_wine_raw = pd.read_csv('winequality-red.csv')
red_wine_cleaned = wine_data_cleaning(red_wine_raw)
print('Does each atrribute column have null data? \n', red_wine_cleaned.
 ↪isnull().any())
# Show the data statistics
red_wine_cleaned.describe()
```

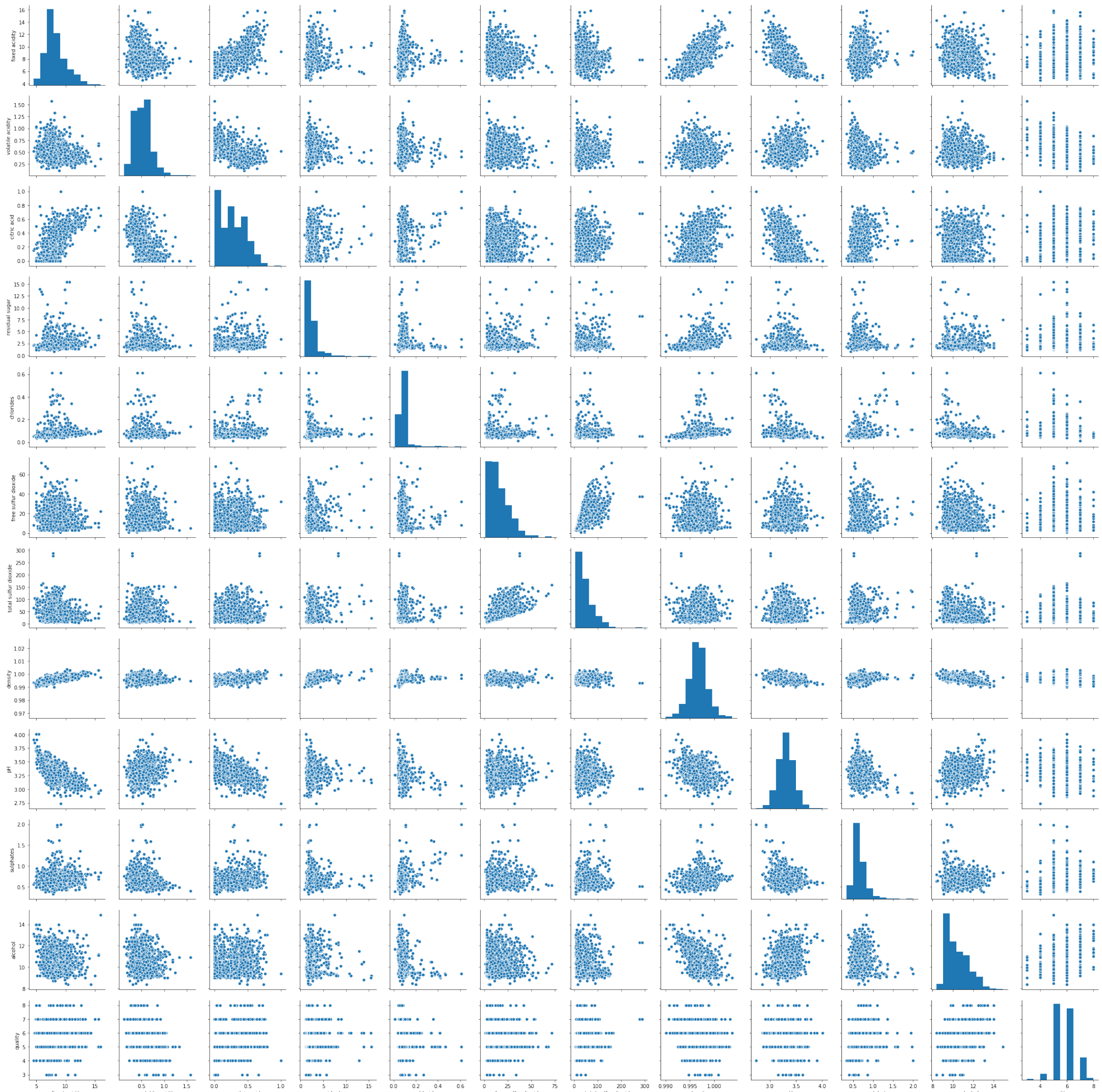
```
Does each atrribute column have null data?
 fixed acidity           False
volatile acidity        False
citric acid             False
residual sugar          False
chlorides               False
free sulfur dioxide     False
total sulfur dioxide    False
density                 False
pH                      False
sulphates               False
alcohol                 False
quality                 False
dtype: bool
```

[6]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000       1599.000000  1599.000000     1599.000000
mean        8.319637          0.527821     0.270976        2.538806
std         1.741096          0.179060     0.194801        1.409928
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1599.000000          1599.000000           1599.000000  1599.000000
```

```
mean      0.087467            15.874922             46.467792      0.996747
std       0.047065            10.460157             32.895324      0.001887
min       0.012000             1.000000              6.000000      0.990070
25%       0.070000             7.000000             22.000000      0.995600
50%       0.079000            14.000000             38.000000      0.996750
75%       0.090000            21.000000             62.000000      0.997835
max       0.611000            72.000000            289.000000      1.003690

                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

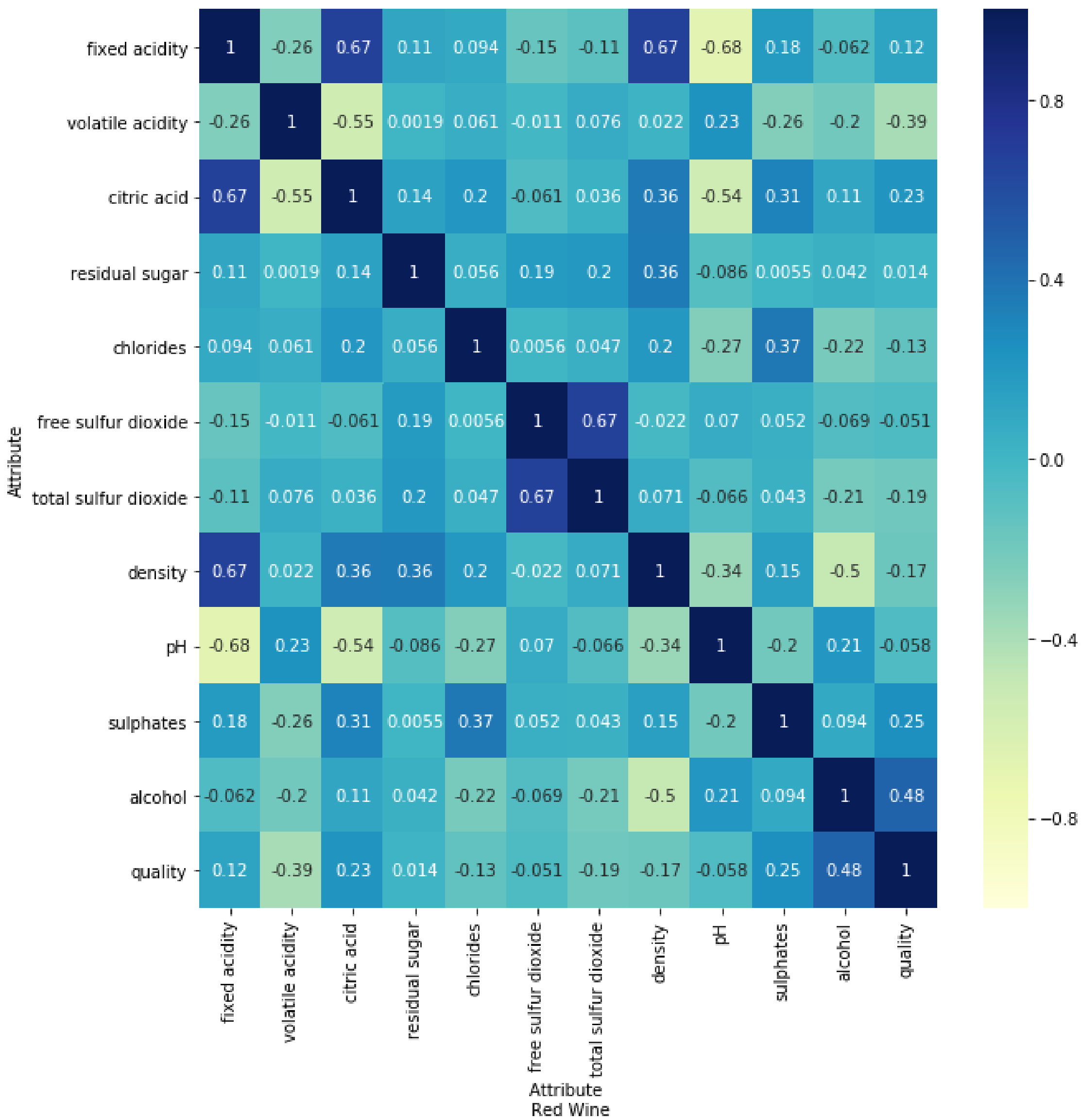
[7]:

```python
# Show the paire-wise attribute correlation
g = sns.pairplot(red_wine_cleaned)
```



[8]:
```
# Plot the attribute correlation coefficient as in heat map
plt.figure(figsize=(10, 10))
ax = sns.heatmap(red_wine_cleaned.corr(),annot=True,vmin=-1,cmap='YlGnBu')
ax.set(xlabel='Attribute \n Red Wine', ylabel='Attribute')
```

[8]: [Text(69.0, 0.5, 'Attribute'), Text(0.5, 69.0, 'Attribute \n Red Wine')]



[9]:
```
# Feature selection by using kNN to find the most salient feature rleated to␣
↪wine quality
def feature_selection_red(wine_cleaned, n_neignbors=90):
    # Generate all possible combinations of attribute feature as index array␣
↪for feature selection
    feature_index = []
    for i in range(1, len(wine_cleaned.columns)-1):
        arr = range(len(wine_cleaned.columns)-1)
        combination = list(combinations(arr, i))
        for c in combination:
            feature_index.append(np.array(c))
    score = []
```

```
    for feature in feature_index:
        X = wine_cleaned.iloc[:,feature].values
        y = wine_cleaned.iloc[:, -1].values
        scaler = StandardScaler()
        scaler.fit(X)
        X_scaled = scaler.transform(X)
        X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,␣
 ↪test_size=0.2, random_state=3)
        # By trial, we used neighbors = 90
#          n_neighbors = 90
        kNN_clf = KNeighborsClassifier(n_neighbors=n_neignbors).fit(X_train,␣
 ↪y_train)
        y_predicted = kNN_clf.predict(X_test)
        score.append(accuracy_score(y_predicted, y_test))
    return feature_index[np.argmax(score)], max(score)
best_feature_red, feature_score_red = feature_selection_red(red_wine_cleaned,␣
 ↪n_neignbors=90)
print('The best feature selection score: ', feature_score_red)
print('The best feature combinations: ', best_feature_red)
```

```
The best feature selection score:  0.646875
The best feature combinations:  [ 1  4  6  9 10]
```

[19]:
```
# Use the best feature combiantion as trainning and testing data
X = red_wine_cleaned.iloc[:, best_feature_red]
y = red_wine_cleaned.iloc[:,-1].values

# If one range of labels need to be merged to one label, set the following
# variable merge_label to True. Else set merge_label=False.
merge_label = False
if merge_label:
    label_to_merge = [5, 6]
    y = merge_quality(y, label_to_merge= label_to_merge)

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=3)
```

[20]:
```
# Use Gaussian Navie Bayes classifier to model the data
def NB_clf(X_train, X_test, y_train, y_test):
    NB_clf = GaussianNB()
    NB_clf.fit(X_train, y_train)
    y_predicted = NB_clf.predict(X_test)
```

```
    NB_clf_error = [mean_squared_error(y_predicted, y_test),␣
↪accuracy_score(y_predicted, y_test),f1_score(y_predicted, y_test,␣
↪average='micro')]
    print('NB_clf')
    print('mean_squared_error: ', mean_squared_error(y_predicted, y_test))
    print('accuracy_score: ', accuracy_score(y_predicted, y_test))
    print('f1_score: ', f1_score(y_predicted, y_test, average='micro'))
    c_matrix = confusion_matrix(y_test, y_predicted)
    print('the prediction confusion matrix is: \n', c_matrix)
    return NB_clf_error
NB_clf_error_red = NB_clf(X_train, X_test, y_train, y_test)
```

```
NB_clf
mean_squared_error:  0.484375
accuracy_score:  0.609375
f1_score:  0.609375
the prediction confusion matrix is:
 [[ 0  1  1  0  0  0]
 [ 0  1  7  4  0  0]
 [ 0  7 97 33  1  0]
 [ 0  0 37 81 11  2]
 [ 0  0  1 16 16  3]
 [ 0  0  0  1  0  0]]
```

[18]:
```
# Use kNN classifier to model the data and use grid search to find the optimal␣
↪model parameters
parameter_tree ={'n_neighbors':range(10,100)}
def kNN_clf(X_train, X_test, y_train, y_test, parameter_tree={'n_neighbors':
↪range(10,100)}):
    kNN_clf = KNeighborsClassifier()
    clf = GridSearchCV(kNN_clf, parameter_tree, cv=5, iid=False, n_jobs=8)
    clf.fit(X_train, y_train)
    y_predicted = clf.predict(X_test)
    kNN_clf_error = [mean_squared_error(y_predicted, y_test),␣
↪accuracy_score(y_predicted, y_test),f1_score(y_predicted, y_test,␣
↪average='micro')]
    print('kNN_clf')
    print('mean_squared_error: ', mean_squared_error(y_predicted, y_test))
    print('accuracy_score: ', accuracy_score(y_predicted, y_test))
    print('f1_score: ', f1_score(y_predicted, y_test, average='micro'))
    c_matrix = confusion_matrix(y_test, y_predicted)
    print('the prediction confusion matrix is: \n', c_matrix)
    kNN_para = clf.best_params_
    print('The best parameter for kNN_clf: ', kNN_para)
    return kNN_clf_error
kNN_clf_error_red = kNN_clf(X_train, X_test, y_train, y_test,␣
↪parameter_tree=parameter_tree)
```

```
kNN_clf
mean_squared_error:  0.403125
accuracy_score:  0.653125
f1_score:  0.653125
the prediction confusion matrix is:
 [[  0   0   2   0   0   0]
 [  0   0   9   3   0   0]
 [  0   0 104  34   0   0]
 [  0   0  35  93   3   0]
 [  0   0   0  24  12   0]
 [  0   0   0   1   0   0]]
The best parameter for kNN_clf:  {'n_neighbors': 88}
```

[22]:

```
# Use Decision Tree classifier to model the data and use grid search to find␣
↪the optimal model parameters
parameter_tree ={'criterion': ['gini', 'entropy'], 'splitter': ['best',␣
↪'random'] ,
                'max_depth': range(1,25), 'min_samples_split': range(2,10),
                'min_samples_leaf': range(1,10)}
def tree_clf(X_train, X_test, y_train, y_test, parameter_tree={'criterion':␣
↪['gini', 'entropy']}):
    tree_clf = tree.DecisionTreeClassifier()
    clf = GridSearchCV(tree_clf, parameter_tree, cv=5, iid=False, n_jobs=8)
    clf.fit(X_train, y_train)
    y_predicted = clf.predict(X_test)
    tree_clf_error_red = [mean_squared_error(y_predicted, y_test),␣
↪accuracy_score(y_predicted, y_test),f1_score(y_predicted, y_test,␣
↪average='micro')]
    print('tree_clf')
    print('mean_squared_error: ', mean_squared_error(y_predicted, y_test))
    print('accuracy_score: ', accuracy_score(y_predicted, y_test))
    print('f1_score: ', f1_score(y_predicted, y_test, average='micro'))
    c_matrix = confusion_matrix(y_test, y_predicted)
    print('the prediction confusion matrix is: \n', c_matrix)
    decision_tree_para = clf.best_params_
    print('The best parameter for tree_clf: ', decision_tree_para)
    return tree_clf_error_red
tree_clf_error_red = tree_clf(X_train, X_test, y_train, y_test,␣
↪parameter_tree=parameter_tree)
```

```
tree_clf
mean_squared_error:  0.6125
accuracy_score:  0.646875
f1_score:  0.646875
the prediction confusion matrix is:
 [[  0   1   1   0   0   0]
 [  0   2   7   2   1   0]
```

```
 [  1   4 101  27   5   0]
 [  0   3  27  86  12   3]
 [  0   0   9   6  18   3]
 [  0   0   0   1   0   0]]
The best parameter for tree_clf:  {'criterion': 'entropy', 'max_depth': 20,
'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
```

[23]:

```
# Use Random Forest classifier to model the data and use grid search to find␣
 ↪the optimal model parameters
parameter_tree ={'n_estimators': range(35,95, 2), 'criterion': ['gini',␣
 ↪'entropy'],
                'max_depth': range(12,30,2), 'min_samples_split': range(2,5,1),
                'min_samples_leaf': range(1,2,1)}
def random_forest(X_train, X_test, y_train, y_test,␣
 ↪parameter_tree={'n_estimators': range(35,85, 15)}):
    random_forest = RandomForestClassifier(verbose=1)
    clf = GridSearchCV(random_forest, parameter_tree, cv=5, iid=False,␣
 ↪n_jobs=8, verbose=1)
    clf.fit(X_train, y_train)
    y_predicted = clf.predict(X_test)
    random_forest_error = [mean_squared_error(y_predicted, y_test),␣
 ↪accuracy_score(y_predicted, y_test),f1_score(y_predicted, y_test,␣
 ↪average='micro')]
    print('random_forest')
    print('mean_squared_error: ', mean_squared_error(y_predicted, y_test))
    print('accuracy_score: ', accuracy_score(y_predicted, y_test))
    print('f1_score: ', f1_score(y_predicted, y_test, average='micro'))
    c_matrix = confusion_matrix(y_test, y_predicted)
    print('the prediction confusion matrix is: \n', c_matrix)
    random_forest_para = clf.best_params_
    print('The best parameter for random_forest: ', random_forest_para)
    return random_forest_error
random_forest_error_red=random_forest(X_train, X_test, y_train, y_test,␣
 ↪parameter_tree=parameter_tree)
```

```
Fitting 5 folds for each of 1620 candidates, totalling 8100 fits
```

```
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  52 tasks      | elapsed:    0.9s
[Parallel(n_jobs=8)]: Done 352 tasks      | elapsed:    7.4s
[Parallel(n_jobs=8)]: Done 852 tasks      | elapsed:   19.0s
[Parallel(n_jobs=8)]: Done 1552 tasks      | elapsed:   37.6s
[Parallel(n_jobs=8)]: Done 2452 tasks      | elapsed:  1.1min
[Parallel(n_jobs=8)]: Done 3552 tasks      | elapsed:  1.6min
[Parallel(n_jobs=8)]: Done 4528 tasks      | elapsed:  2.3min
[Parallel(n_jobs=8)]: Done 5278 tasks      | elapsed:  3.0min
[Parallel(n_jobs=8)]: Done 6128 tasks      | elapsed:  3.7min
[Parallel(n_jobs=8)]: Done 7078 tasks      | elapsed:  4.5min
```

```
[Parallel(n_jobs=8)]: Done 8100 out of 8100 | elapsed:  5.3min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  55 out of  55 | elapsed:    0.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
random_forest
mean_squared_error:  0.353125
accuracy_score:  0.721875
f1_score:  0.7218749999999999
the prediction confusion matrix is:
 [[  0   0   2   0   0   0]
 [  0   0   8   4   0   0]
 [  0   2 119  17   0   0]
 [  0   0  27  94  10   0]
 [  0   0   1  13  18   4]
 [  0   0   0   1   0   0]]
The best parameter for random_forest:  {'criterion': 'entropy', 'max_depth': 18,
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 55}
```

```
[Parallel(n_jobs=1)]: Done  55 out of  55 | elapsed:    0.0s finished
```

[28]:

```python
#Turn the predicted probability into class labels
def translate_label(y):
    y_translated = []
    for i in y:
        y_translated.append(np.argmax(i))
    return y_translated
```

[27]:

```python
# Neural Network - Densely Connected
X = red_wine_cleaned.iloc[:,:-1]
y = red_wine_cleaned.iloc[:, -1].values

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
# print(X_scaled)
num_classes = 10
y = keras.utils.to_categorical(y, num_classes)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=7)
d = 0.15
model = Sequential()
model.add(Dense(30, input_dim=len(X_train[0]), activation='linear'))
model.add(Dropout(d))
model.add(Dense(50,  activation='tanh'))
model.add(Dropout(d))
model.add(Dense(50, activation='tanh'))
model.add(Dropout(d))
```

```
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adadelta(),
metrics=['accuracy'])

batch_size = 50
model.fit(X_train, y_train, batch_size=batch_size, epochs=300, verbose=0,␣
 ↪validation_data=(X_test, y_test))
y_predicted = model.predict(X_test)
```

[26]:
```
y_predicted_scalar = translate_label(y_predicted)
y_test_scalar = translate_label(y_test)
neural_network_red = [mean_squared_error(y_predicted_scalar, y_test_scalar),
                      accuracy_score(y_predicted_scalar, y_test_scalar),
                      f1_score(y_predicted_scalar, y_test_scalar,␣
 ↪average='micro')]
print('densely connected neuralwork')
print('mean_squared_error: ', mean_squared_error(y_predicted_scalar,␣
 ↪y_test_scalar))
print('accuracy_score: ', accuracy_score(y_predicted_scalar, y_test_scalar))
print('f1_score: ', f1_score(y_predicted_scalar, y_test_scalar,␣
 ↪average='micro'))
c_matrix = confusion_matrix(y_predicted_scalar, y_test_scalar)
print('the prediction confusion matrix is: \n', c_matrix)
```

```
densely connected neuralwork
mean_squared_error:  0.40625
accuracy_score:  0.675
f1_score:  0.675
the prediction confusion matrix is:
 [[  1   1   0   0   0]
 [  7  95  31   0   0]
 [  1  25 101  16   2]
 [  1   2  13  19   2]
 [  0   0   1   2   0]]
```

3 White Wine Data

[29]:
```
# Read the Red wine data
white_wine_raw  = pd.read_csv('winequality-white.csv')
white_wine_cleaned = wine_data_cleaning(white_wine_raw)
# We fitered out wine with quality=9 since there are only 5 instance for this␣
 ↪category, which is
# too less for a classifier
white_wine_cleaned = white_wine_cleaned[white_wine_cleaned['quality']!=9]
```

```python
print('Does each atrribute column have null data? \n', red_wine_cleaned.
 ↪isnull().any())
# Show the data statistics
white_wine_cleaned.describe()
```

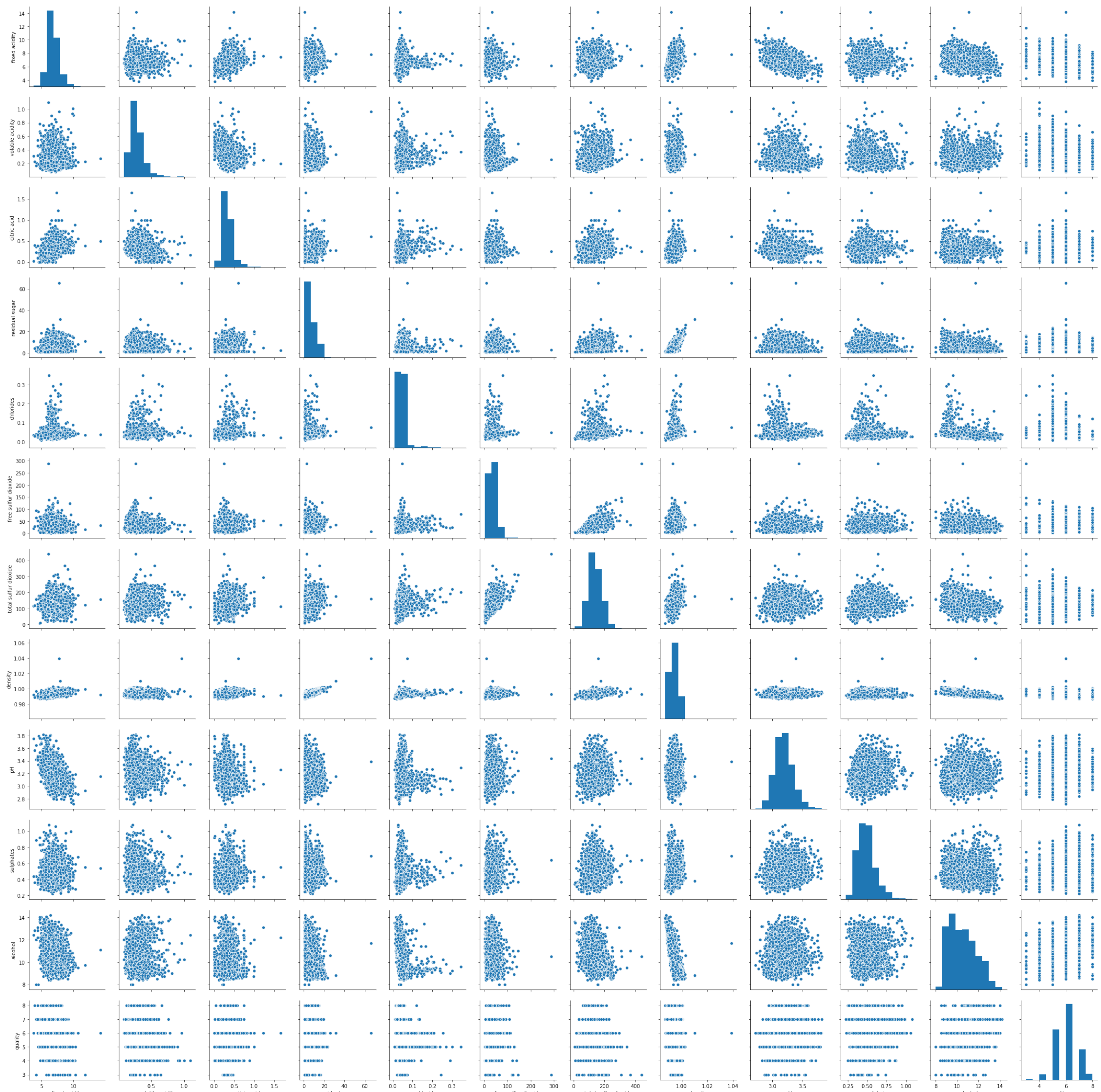
```
Does each atrribute column have null data?
 fixed acidity           False
volatile acidity        False
citric acid             False
residual sugar          False
chlorides               False
free sulfur dioxide     False
total sulfur dioxide    False
density                 False
pH                      False
sulphates               False
alcohol                 False
quality                 False
dtype: bool
```

[29]:

```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    4893.000000       4893.000000  4893.000000     4893.000000
mean        6.854210          0.278221     0.334139        6.393736
std         0.843637          0.100831     0.121048        5.072990
min         3.800000          0.080000     0.000000        0.600000
25%         6.300000          0.210000     0.270000        1.700000
50%         6.800000          0.260000     0.320000        5.200000
75%         7.300000          0.320000     0.390000        9.900000
max        14.200000          1.100000     1.660000       65.800000

         chlorides  free sulfur dioxide  total sulfur dioxide     density  \
count  4893.000000          4893.000000           4893.000000  4893.00000
mean      0.045791            35.310035            138.383507     0.99403
std       0.021850            17.011384             42.509982     0.00299
min       0.009000             2.000000              9.000000     0.98711
25%       0.036000            23.000000            108.000000     0.99173
50%       0.043000            34.000000            134.000000     0.99375
75%       0.050000            46.000000            167.000000     0.99610
max       0.346000           289.000000            440.000000     1.03898

                pH    sulphates      alcohol      quality
count  4893.000000  4893.000000  4893.000000  4893.000000
mean      3.188144     0.489871    10.512565     5.874719
std       0.151011     0.114151     1.229755     0.880446
min       2.720000     0.220000     8.000000     3.000000
25%       3.090000     0.410000     9.500000     5.000000
50%       3.180000     0.470000    10.400000     6.000000
```

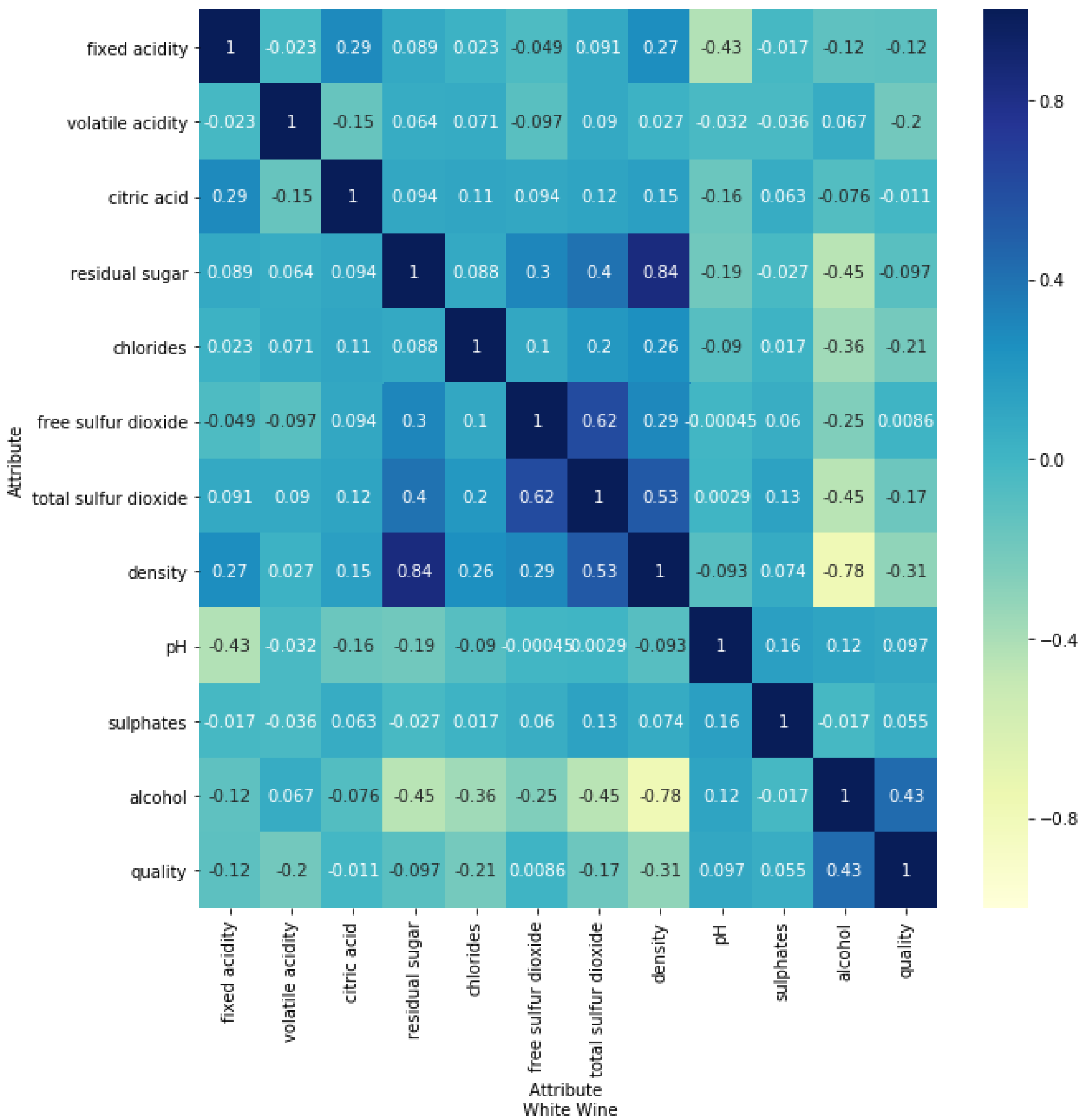
```
75%         3.280000     0.550000     11.400000      6.000000
max         3.820000     1.080000     14.200000      8.000000
```

[30]:
```python
# Show the paire-wise attribute correlation
g = sns.pairplot(white_wine_cleaned)
```



[31]:
```python
# Plot the attribute correlation coefficient as in heat map
plt.figure(figsize=(10, 10))
ax = sns.heatmap(white_wine_cleaned.corr(),annot=True,vmin=-1,cmap='YlGnBu')
ax.set(xlabel='Attribute \n White Wine', ylabel='Attribute')
```

[31]:
```
[Text(69.0, 0.5, 'Attribute'), Text(0.5, 69.0, 'Attribute \n White Wine')]
```




```
[32]: # Feature selection by using kNN to find the most salient feature rleated to
      ↪wine quality
      def feature_selection_white(wine_cleaned):
          # Generate all possible combinations of attribute feature as index array
      ↪for feature selection
          feature_index = []
          for i in range(1, len(wine_cleaned.columns)-1):
              arr = range(len(wine_cleaned.columns)-1)
              combination = list(combinations(arr, i))
              for c in combination:
                  feature_index.append(np.array(c))
          score = []
```

```
    for feature in feature_index:
        X = wine_cleaned.iloc[:,feature].values
        y = wine_cleaned.iloc[:, -1].values
        scaler = StandardScaler()
        scaler.fit(X)
        X_scaled = scaler.transform(X)
        X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,␣
 ↪test_size=0.2, random_state=3)
        # By trial, we used neighbors = 90
        n_neignbors=90
        kNN_clf = KNeighborsClassifier(n_neighbors=n_neignbors).fit(X_train,␣
 ↪y_train)
#         NB_clf = GaussianNB().fit(X_train, y_train)
        y_predicted = kNN_clf.predict(X_test)
        score.append(accuracy_score(y_predicted, y_test))
    print('Total number of feature combinations: ', len(score))
    return feature_index[np.argmax(score)], max(score)
best_feature_white, feature_score_white =␣
 ↪feature_selection_white(white_wine_cleaned)
print('The best feature selection score: ', feature_score_white)
print('The best feature combinations: ', best_feature_white)
```

```
Total number of feature combinations:  2046
The best feature selection score:  0.5556690500510725
The best feature combinations:  [ 0  1  2  4  5  8 10]
```

[33]:
```
# Use the best feature combiantion as trainning and testing data
# print(best_feature_white)
X = white_wine_cleaned.iloc[:,best_feature_white]
y = white_wine_cleaned.iloc[:, -1].values

# If one range of labels need to be merged to one label, set the following
# variable merge_label to True. Else set merge_label=False.
merge_label = False
if merge_label:
    label_to_merge = [5, 6]
    y = merge_quality(y, label_to_merge= label_to_merge)

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=3)
NB_clf = GaussianNB()
NB_clf.fit(X_train, y_train)
y_predicted = NB_clf.predict(X_test)
print('mean_squared_error: ', mean_squared_error(y_predicted, y_test))
```

```
print('accuracy_score: ', accuracy_score(y_predicted, y_test))
print('f1_score: ', f1_score(y_predicted, y_test, average='micro'))
```

```
mean_squared_error:  0.8784473953013279
accuracy_score:  0.48314606741573035
f1_score:  0.48314606741573035
```

[52]:
```
parameter_tree = {'n_neighbors':range(10,300,5)}
kNN_clf_error_red = kNN_clf(X_train, X_test, y_train, y_test,␣
↪parameter_tree=parameter_tree)
```

```
kNN_clf
mean_squared_error:  0.634320735444331
accuracy_score:  0.550561797752809
f1_score:  0.550561797752809
the prediction confusion matrix is:
 [[  0   0   2   0   0   0]
 [  0   2  20  13   0   0]
 [  0   1 185 108   7   0]
 [  0   0  93 302  37   0]
 [  0   0   8 112  50   0]
 [  0   0   2  25  12   0]]
The best parameter for kNN_clf:  {'n_neighbors': 25}
```

[53]:
```
# Use Decision Tree classifier to model the data and use  search to find the␣
↪optimal model parameters
parameter_tree ={'criterion': ['gini', 'entropy'], 'splitter': ['best',␣
↪'random'] ,
                'max_depth': range(1,35), 'min_samples_split': range(2,10),
                'min_samples_leaf': range(1,10)}
tree_clf_error_white = tree_clf(X_train, X_test, y_train, y_test,␣
↪parameter_tree=parameter_tree)
```

```
tree_clf
mean_squared_error:  0.8580183861082737
accuracy_score:  0.5985699693564862
f1_score:  0.5985699693564862
the prediction confusion matrix is:
 [[  0   0   1   1   0   0]
 [  1   5  10  17   2   0]
 [  2   9 196  70  17   7]
 [  4   9  81 278  51   9]
 [  2   2  14  51  94   7]
 [  0   2   2  12  10  13]]
The best parameter for tree_clf:  {'criterion': 'gini', 'max_depth': 23,
'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'random'}
```

[54]:
```python
# Use Random Forest classifier to model the data and use grid search to find␣
↪the optimal model parameters
parameter_tree ={'n_estimators': range(330,380, 5), 'criterion': ['gini',␣
↪'entropy'],
                'max_depth': range(12,20,2), 'min_samples_split': range(2,6,1),
                'min_samples_leaf': range(1,2,1)}

random_forest = RandomForestClassifier(verbose=1)
clf = GridSearchCV(random_forest, parameter_tree, cv=5, iid=False, n_jobs=8,␣
↪verbose=1)
clf.fit(X_train, y_train)
y_predicted = clf.predict(X_test)
random_forest_red = [mean_squared_error(y_predicted, y_test),␣
↪accuracy_score(y_predicted, y_test),f1_score(y_predicted, y_test,␣
↪average='micro')]
print('random_forest')
print('mean_squared_error: ', mean_squared_error(y_predicted, y_test))
print('accuracy_score: ', accuracy_score(y_predicted, y_test))
print('f1_score: ', f1_score(y_predicted, y_test, average='micro'))
c_matrix = confusion_matrix(y_test, y_predicted)
print('the prediction confusion matrix is: \n', c_matrix)
random_forest_para = clf.best_params_
print('The best parameter for random_forest: ', random_forest_para)
```

```
Fitting 5 folds for each of 320 candidates, totalling 1600 fits
```

```
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:   12.4s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:  1.0min
[Parallel(n_jobs=8)]: Done 434 tasks      | elapsed:  2.6min
[Parallel(n_jobs=8)]: Done 784 tasks      | elapsed:  4.8min
[Parallel(n_jobs=8)]: Done 1234 tasks      | elapsed:  8.6min
[Parallel(n_jobs=8)]: Done 1600 out of 1600 | elapsed: 12.1min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
random_forest
mean_squared_error:  0.45658835546475995
accuracy_score:  0.6710929519918284
f1_score:  0.6710929519918284
the prediction confusion matrix is:
 [[  0   0   2   0   0   0]
 [  0   8  16  11   0   0]
 [  0   2 194 101   3   1]
 [  0   0  64 346  22   0]
 [  0   0   3  69  97   1]
 [  0   0   0  20   7  12]]
The best parameter for random_forest:  {'criterion': 'entropy', 'max_depth': 18,
'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 345}
```

```
[Parallel(n_jobs=1)]: Done 345 out of 345 | elapsed:    2.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 345 out of 345 | elapsed:    0.0s finished
```

[36]:
```
# Neural Network - Densely Connected
X = white_wine_cleaned.iloc[:,best_feature_white]
y = white_wine_cleaned.iloc[:, -1].values
print(best_feature_white)
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
num_classes = 10
y = keras.utils.to_categorical(y, num_classes)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=7)
d = 0.05
model = Sequential()
model.add(Dense(50, input_dim=len(X_train[0]), activation='linear'))
model.add(Dropout(d))
model.add(Dense(80,  activation='tanh'))
model.add(Dropout(d))
model.add(Dense(50, activation='tanh'))
model.add(Dropout(d))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adadelta(),
metrics=['accuracy'])

batch_size = 50
model.fit(X_train, y_train, batch_size=batch_size, epochs=300, verbose=0,␣
 ↪validation_data=(X_test, y_test))
y_predicted = model.predict(X_test)
```

```
[ 0  1  2  4  5  8 10]
```

[37]:
```
y_predicted = translate_label(y_predicted)
y_test = translate_label(y_test)
neural_network_red = [mean_squared_error(y_predicted, y_test),␣
 ↪accuracy_score(y_predicted, y_test),f1_score(y_predicted, y_test,␣
 ↪average='micro')]
print('densely connected neuralwork')
print('mean_squared_error: ', mean_squared_error(y_predicted, y_test))
print('accuracy_score: ', accuracy_score(y_predicted, y_test))
print('f1_score: ', f1_score(y_predicted, y_test, average='micro'))
c_matrix = confusion_matrix(y_test, y_predicted)
print('the prediction confusion matrix is: \n', c_matrix)
```

```
densely connected neuralwork
```

```
mean_squared_error:  0.5413687436159347
accuracy_score:  0.6016343207354443
f1_score:  0.6016343207354443
the prediction confusion matrix is:
 [[  1   0   3   0   0   0]
 [  0  12   9   6   0   0]
 [  3  11 187  87   7   0]
 [  0   2  85 291  58   2]
 [  0   0   9  72  85   8]
 [  0   0   1  12  15  13]]
```

[ ]:

[ ]: