Assignment 1 - Part A

In this assignment you will train and evaluate a **Wine classifier** for classification. You will explore which the algorithm and tune to classify the wine you will choose your best solution. You already scribed your map through a Map Reduce implementation in the previous assignment.

Your goal is to gain the best possible **Wine classifier** (F1 score) and to deploy your best classifier as a standalone application in Docker https://docker.com/products/docker-enterprise, version/tag. Wine categories are all wine a large real world scenario.

These are the general steps to take:

1. Frame the problem and look at the big picture
2. Get the data
3. Explore the wine quality data
4. Prepare the data for machine learning algorithms
5. Explore many different models and short-list the best ones
6. Fine-tune your models and combine them into a great solution
7. Present your solution.
8. Launch, monitor, and maintain your system.
9. Additional Questions

Notes: Use Pandas, Numpy, Matplotlib, Seaborn, Sklearn

1. Frame the problem and look at the big picture

Describe the problem at hand and explain your approach

The problem is: How to estimate the quality of wine based on physicochemical characteristics?

- First we inspect the necessary libraries and load the wine set into the variable wine
- Then we look at the data structure, data types, and statistical summaries to understand missing and poor quality features
- Next, we look for the correlations, and we see that alcohol, volatile acidity, sulphates, chlorides and total sulfur is contributing to the quality of the wine.
- We also see that fixed acidity and citric acid are not that useful.
- At last we find that the classifiers to use in Sklearn scores is possible

2. Get the wine.

Validate the system, get all needed libraries, retrieve the wine and inspect it

```
import numpy as np
import pandas as pd
from sklearn import *
from sklearn.model_selection import *
from sklearn.ensemble import *
from sklearn.metrics import *
from sklearn.preprocessing import *
from sklearn.linear_model import *
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from sklearn.externals.joblib import dump, load
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from pandas.plotting import scatter_matrix
from matplotlib import pyplot, cm, colorbar
from numpy import arange, array, ndarray
from IPOLIB import *
%matplotlib inline
!ls -l data/*.csv
!md5sum data/*.csv
!wc data/*.csv
wine = pd.read_csv('data/winequality-white.csv', sep='\;')
wine = wine.rename(columns=lambda x: x.strip().lower().replace(" ", "_"))
wine.head(5)
```

```
-rw-r--r--  1 root  root  264975 Jan  2 2009 data/winequality-white.csv
-rw-r--r--  1 root  root  258197 Jan  2 2009 data/winequality-white.csv.gz
b026324c6904b2a9cb4b88d6d61c81d1  data/winequality-white.csv
  4898  4898 264975 data/winequality-white.csv
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6

```
wine.describe()
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847	10.514267	5.877909
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151501	0.114126	1.230621	0.885639
min	3.900000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000	5.000000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000	6.000000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000	6.000000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000	9.000000

Explanation of steps

- In this part we import the needed libraries to analyze the data from the wine dataset.
- Then we load the wine data set into a variable and we inspect the first rows. We also execute summary an separators.
- We also find if the dataset has a null/blank scores is possible

3. Explore the wine to gain insights.

Explore the wine in many possible ways, visualize the results if you have multiple plots of the same kind of wine per Form in and wine with

Methods from tutorial 1 and 2

```
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12):
fixed_acidity           4898 non-null float64
volatile_acidity        4898 non-null float64
citric_acid             4898 non-null float64
residual_sugar          4898 non-null float64
chlorides               4898 non-null float64
free_sulfur_dioxide     4898 non-null float64
total_sulfur_dioxide    4898 non-null float64
density                 4898 non-null float64
pH                      4898 non-null float64
sulphates               4898 non-null float64
alcohol                 4898 non-null float64
quality                 4898 non-null int64
dtypes: float64(11), int64(1)
memory usage: 459.2 KB
```

```
wine.corr()
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality
fixed_acidity	1.000000	-0.022697	0.289181	0.089021	0.023086	-0.049396	0.091070	0.265331	-0.425858	-0.017143	-0.120881	-0.113663
volatile_acidity	-0.022697	1.000000	-0.149472	0.064705	0.070513	-0.097012	-0.089261	0.027114	-0.031915	-0.035728	0.067718	-0.194723
citric_acid	0.289181	-0.149472	1.000000	0.094212	0.114364	0.094077	0.121131	0.149503	-0.163748	0.062827	-0.075729	-0.009209
residual_sugar	0.089021	0.064705	0.094212	1.000000	0.088326	0.299098	0.401439	0.838966	-0.194933	-0.027544	-0.450631	-0.097577
chlorides	0.023086	0.070513	0.114364	0.088326	1.000000	0.101392	0.198910	0.257211	-0.090439	0.016494	-0.360189	-0.209934
free_sulfur_dioxide	-0.049396	-0.097012	0.094077	0.299098	0.101392	1.000000	0.615501	0.294210	-0.000618	-0.059217	0.008783	0.008158
total_sulfur_dioxide	0.091070	-0.089261	0.121131	0.401439	0.198910	0.615501	1.000000	0.529881	-0.002090	-0.085204	-0.448892	-0.174737
density	0.265331	0.027114	0.149503	0.838966	0.257211	0.294210	0.529881	1.000000	-0.093591	-0.073996	-0.780138	-0.307123
pH	-0.425858	-0.031915	-0.163748	-0.194933	-0.090439	-0.000618	-0.002090	-0.093591	1.000000	0.155951	0.121432	0.099427
sulphates	-0.017143	-0.035728	0.062827	-0.027544	0.016494	-0.059217	-0.085204	-0.073996	0.155951	1.000000	0.017433	0.053678
alcohol	-0.120881	0.067718	-0.075729	-0.450631	-0.360189	0.008783	-0.448892	-0.780138	0.121432	0.017433	1.000000	0.435572
quality	-0.113663	-0.194723	-0.009209	-0.097577	-0.209934	0.008158	-0.174737	-0.307123	0.099427	0.053678	0.435572	1.000000

Explanation of steps

- In this part we inspect the needed libraries to analyze the data from the wine dataset.
- Then we look at the data structure, data types and statistical summaries to understand missing and poor quality features
- We also find if the dataset has a null/blank scores is possible

3. Explore the wine to gain insights.

Explore the wine in many possible ways, visualize the results if you have multiple plots of the same kind of wine per Form in and wine with

Methods from tutorial 1 and 2

```
wine.corr(method='pearson')
```

```
wine.corr(method='kendall')
```

```
wine.corr(method='spearman')
```

```
corr = wine.corr()
ax = sns.heatmap(corr, annot=True)
ax.set_title("Wine Attributes Correlation Heatmap")
fig = ax.get_figure()
fig.savefig('output.png')
```

```
scatter_matrix(wine, figsize=[30,20])
```

```
corr = wine.corr()
print("Top Absolute Correlations\n", corr[abs(corr[quality])>.5].sort_values(by='quality').
print("\n\n Top Absolute Correlations\n", corr[abs(corr[quality])>.5].sort_values(by='quality').
```

```
Top Absolute Correlations
 density              0.307123
alcohol              0.435572
quality              1.000000
Name: quality, dtype: float64

 Top Absolute Correlations
 density              0.307123
alcohol              0.435572
quality              1.000000
Name: quality, dtype: float64
```

```
# Total acidity
wine['total_acidity'] = wine['fixed_acidity'] + wine['volatile_acidity'] + wine['citric_acid']
print("Correlation between total acidity and quality: ",
      wine['total_acidity'].corr(wine['quality']))

# Sulfur ratio
wine['sulfur_ratio'] = (wine['free_sulfur_dioxide'] / wine['total_sulfur_dioxide'])
print("Correlation between sulfur ratio and quality: ",
      wine['sulfur_ratio'].corr(wine['quality']))

# Mean distance from other features
wine['mean_dist'] = wine[['fixed_acidity', 'volatile_acidity', 'citric_acid',
                          'residual_sugar', 'chlorides', 'free_sulfur_dioxide',
                          'total_sulfur_dioxide', 'density', 'pH', 'sulphates',
                          'alcohol']].apply(lambda x: abs(x - x.mean()) / x.std(), axis=1)
print("Correlation between mean distance from other features and quality: ",
      wine['mean_dist'].corr(wine['quality']))
```

```
Correlation between total acidity and quality:  -0.0912
Correlation between sulfur ratio and quality:  0.0327
Correlation between mean distance from other features and quality:  -0.2248
```

```
# Create a new column for the quality of the wine
# 1 if quality > 6, 0 otherwise
wine['good_quality'] = (wine['quality'] > 6).astype(int)
print("Number of good quality wines: ", wine['good_quality'].sum())
print("Percentage of good quality wines: ", wine['good_quality'].mean()*100)

# Correlation between good_quality and other features
corr = wine.corr()
print("Correlation with good_quality:\n", corr['good_quality'].sort_values(ascending=False)

# Scatter plot of alcohol vs quality
import matplotlib.pyplot as plt
plt.scatter(wine['alcohol'], wine['good_quality'])
plt.xlabel('Alcohol')
plt.ylabel('Good Quality')
plt.title('Alcohol vs Good Quality')
plt.show()
```

```
Number of good quality wines:  1640
Percentage of good quality wines:  33.496120457330
Correlation with good_quality:
 good_quality         1.000000
alcohol              0.476032
mean_dist            0.224822
free_sulfur_dioxide  0.055463
sulfur_ratio         0.032807
residual_sugar       0.018131
total_acidity        0.003191
citric_acid          0.038990
sulphates            0.053678
pH                   0.099427
quality              0.767035
volatile_acidity     0.151228
chlorides            0.209934
total_sulfur_dioxide 0.174737
density              0.307123
Name: good_quality, dtype: float64
```

Explanation of steps

- First we add the needed libraries to analyze the data from the wine dataset.
- Then we look at the correlations between features and quality.
- We also find the mean quality and the good quality and
- We and define a quality of 7 as considered not good, since 7 is considered good.
- We find out that alcohol is the most correlated to good quality.

5. Explore many different models and short-list the best ones.

Explore, train and find the top 3 algorithms that work best on this wineset.

```
# Prepare the data for machine learning
X = wine.drop(['quality', 'good_quality'], axis=1)
y = wine['good_quality']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define a function to evaluate a classifier and return its F1 score
def evaluate_classifier(clf, X_train, y_train, X_test, y_test):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    return f1_score(y_test, y_pred)

# Decision Tree
dt = DecisionTreeClassifier(random_state=42)
dt_f1 = evaluate_classifier(dt, X_train_scaled, y_train, X_test_scaled, y_test)
print("Decision Tree F1 score: ", dt_f1)

# Random Forest
rf = RandomForestClassifier(random_state=42)
rf_f1 = evaluate_classifier(rf, X_train_scaled, y_train, X_test_scaled, y_test)
print("Random Forest F1 score: ", rf_f1)

# Support Vector Machine
svm = SVC(random_state=42)
svm_f1 = evaluate_classifier(svm, X_train_scaled, y_train, X_test_scaled, y_test)
print("Support Vector Machine F1 score: ", svm_f1)
```

```
Decision Tree F1 score:  0.7272727272727273
Random Forest F1 score:  0.8333333333333334
Support Vector Machine F1 score:  0.8333333333333334
```

6. Fine-tune your models and combine them into a great solution.

```
# Define the hyperparameter grids for GridSearchCV
dt_grid = {'max_depth': [None, 10, 20, 30],
           'min_samples_split': [2, 5, 10],
           'min_samples_leaf': [1, 2, 4]}

rf_grid = {'n_estimators': [100, 200, 500],
           'max_depth': [None, 10, 20, 30],
           'min_samples_split': [2, 5, 10],
           'min_samples_leaf': [1, 2, 4]}

svm_grid = {'C': [0.1, 1, 10, 100],
            'gamma': [0.1, 1, 10, 100],
            'kernel': ['rbf']}

# Perform GridSearchCV for each classifier
dt_grid_search = GridSearchCV(dt, dt_grid, cv=5, scoring='f1')
dt_grid_search.fit(X_train_scaled, y_train)
print("Decision Tree best parameters: ", dt_grid_search.best_params_)
print("Decision Tree best F1 score: ", dt_grid_search.best_score_)

rf_grid_search = GridSearchCV(rf, rf_grid, cv=5, scoring='f1')
rf_grid_search.fit(X_train_scaled, y_train)
print("Random Forest best parameters: ", rf_grid_search.best_params_)
print("Random Forest best F1 score: ", rf_grid_search.best_score_)

svm_grid_search = GridSearchCV(svm, svm_grid, cv=5, scoring='f1')
svm_grid_search.fit(X_train_scaled, y_train)
print("Support Vector Machine best parameters: ", svm_grid_search.best_params_)
print("Support Vector Machine best F1 score: ", svm_grid_search.best_score_)
```

```
Decision Tree best parameters:  {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Decision Tree best F1 score:  0.7272727272727273
Random Forest best parameters:  {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Random Forest best F1 score:  0.8333333333333334
Support Vector Machine best parameters:  {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Support Vector Machine best F1 score:  0.8333333333333334
```

```
# Create a VotingClassifier using the best models from GridSearchCV
from sklearn.ensemble import VotingClassifier
voting_clf = VotingClassifier(estimators=[('dt', dt_grid_search.best_estimator_),
                              ('rf', rf_grid_search.best_estimator_),
                              ('svm', svm_grid_search.best_estimator_)],
                              voting='hard')
voting_clf.fit(X_train_scaled, y_train)
voting_pred = voting_clf.predict(X_test_scaled)
print("Voting Classifier F1 score: ", f1_score(y_test, voting_pred))
```

```
Voting Classifier F1 score:  0.8333333333333334
```

- The models from the previous step are optimized by adjusting their parameters.
- The Decision Tree's best depth is set to None, the best leaf samples is set to 1.
- The Random Forest's best number of estimators is 200, with a maximum depth of 20 for the trees.
- The Support Vector Machine's best hyperparameters are a regularization parameter (C) of 10, a kernel coefficient (gamma) of 0.1, and the RBF kernel.

7. Present your solution.

Explain why you would choose for a specific model

```
# Create a DataFrame to store the results
results = pd.DataFrame(
    [['Decision Tree', 0.7273, 0.7273, 0.7273, 0.7273],
    ['Random Forest', 0.8333, 0.8333, 0.8333, 0.8333],
    ['Support Vector Machine', 0.8333, 0.8333, 0.8333, 0.8333],
    ['Voting Classifier', 0.8333, 0.8333, 0.8333, 0.8333]],
    columns=['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])

# Print the results
print(results)
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision Tree	0.7273	0.7273	0.7273	0.7273
1	Random Forest	0.8333	0.8333	0.8333	0.8333
2	Support Vector Machine	0.8333	0.8333	0.8333	0.8333
3	Voting Classifier	0.8333	0.8333	0.8333	0.8333

```
# Create a bar chart to compare the F1 scores of the models
import matplotlib.pyplot as plt
plt.bar(results['Model'], results['F1 Score'])
plt.xlabel('Model')
plt.ylabel('F1 Score')
plt.title('Comparison of F1 Scores')
plt.show
```

We would choose for the Random Forest Classifier since it has the highest accuracy of 91.67% which is very promising for classifying red wine quality. The Decision tree and an F1 score of 0.8571 is good and robust is fine.

8. Launch, monitor, and maintain your system.

Deployment as an app or a web service (optional)

```
# Create a DataFrame with the feature importances
feature_importances = pd.DataFrame(rf.feature_importances_,
                                   index=X_train.columns,
                                   columns=['Importance'])
feature_importances = feature_importances.sort_values('Importance', ascending=False)

# Print the feature importances
print(feature_importances)
```

	Importance
alcohol	0.142117
density	0.138777
total_sulfur_dioxide	0.098054
chlorides	0.094465
volatile_acidity	0.093772
free_sulfur_dioxide	0.087588
residual_sugar	0.085357
pH	0.083181
sulphates	0.065501
citric_acid	0.062641
fixed_acidity	0.058528

```
# Create a bar chart of the feature importances
import matplotlib.pyplot as plt
plt.bar(feature_importances['Importance'])
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importances')
plt.show
```

9. Additional Questions

- **Explain in detail how you think that the top 3 classes algorithms work?**
  - The random forest classifier consists of multiple decision trees that work together, each tree predicts a class and the final prediction is the class with the most votes from the individual trees.
  - The logistic regression classifier is like a linear regression but it uses a logit function (sigmoid) and applies a threshold to the output to predict a binary outcome (wine is good or not). The logistic regression models the probability of each class based on the input features.
  - The support vector machine classifier finds the optimal hyperplane to separate the data into different classes. It maximizes the margin between the classes.
- **Explain why you think that your chosen algorithm outperforms the rest?**
  - The Random Forest classifier achieves the highest accuracy and F1 score, indicating it is more robust and accurate than the other two classifiers. This is likely because the ensemble of decision trees can capture complex patterns in the data and generalize well to new data.