Predicting the Wine Quality

About the problem

his datasets is related to red and white variants of the Portuguese "Vinho Verde" wine. For more details, consult the reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

The datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are much more normal wines than excellent or poor ones).

This dataset is also available from the UCI machine learning repository, https://archive.ics.uci.edu/ml/datasets/wine+quality (https://archive.ics.uci.edu/ml/datasets/wine+quality)

Getting the data

In [1]:

```
mkdir -p Data/Wine
!wget -O Data/Wine/wine-red.csv https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/wineq
uality-red.csv
!wget -O Data/Wine/wine-white.csv https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/win
equality-white.csv
!ls *.csv
```

```
--2018-10-28 22:57:05--  https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality
/winequality-red.csv
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.249
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.249|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 84199 (82K) [text/csv]
Saving to: ‘Data/Wine/wine-red.csv’

Data/Wine/wine-red. 100%[===================>]  82.23K  67.0KB/s    in 1.2s

2018-10-28 22:57:07 (67.0 KB/s) - ‘Data/Wine/wine-red.csv’ saved [84199/84199]

--2018-10-28 22:57:07--  https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality
/winequality-white.csv
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.249
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.249|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 264426 (258K) [text/csv]
Saving to: ‘Data/Wine/wine-white.csv’

Data/Wine/wine-whit 100%[===================>] 258.23K  93.4KB/s    in 2.8s

2018-10-28 22:57:11 (93.4 KB/s) - ‘Data/Wine/wine-white.csv’ saved [264426/264426]

wine-red.csv  wine-white.csv
```

Imports

In [2]:

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

Extracting the data

In [3]:

```
train = []
with open('./Data/Wine/wine-red.csv') as fp:
    fp.readline()
    line = fp.readline()
    while line:
        train.append(list(map(float, line.strip().split(';'))))
        line = fp.readline()
with open('./Data/Wine/wine-white.csv') as fp:
    fp.readline()
    line = fp.readline()
    while line:
        train.append(list(map(float, line.strip().split(';'))))
        line = fp.readline()
train = np.array(train)
train.shape
```

Out[3]:

```
(6497, 12)
```

In [4]:

```
test = train[:, [-1]]
test = test.reshape(-1)
test = test > 6
test = test.astype(int)
```

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(train, test, test_size=0.2, random_state=42)
```

PCA for K = 6

In [6]:

```
pca = PCA(n_components=6)
pca.fit(X_train)
```

Out[6]:

```
PCA(copy=True, iterated_power='auto', n_components=6, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

In [7]:

```
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

Grid search to find the best parameters

In [8]:

```
tuned_parameters = [
  {'C': [1, 10, 100, 1000, 10000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000, 10000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
scores_list = ['precision', 'recall']
for score in scores_list:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(SVC(), tuned_parameters, cv=5,
                       scoring='%s_macro' % score)
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()
    print("Final accuracy =", accuracy_score(y_test, clf.predict(X_test)))
```

```
# Tuning hyper-parameters for precision


/home/sai/.virtualenvs/Tensorflow/lib/python3.6/site-packages/sklearn/metrics/classification.py
:1135: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/sai/.virtualenvs/Tensorflow/lib/python3.6/site-packages/sklearn/metrics/classification.py
:1135: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/sai/.virtualenvs/Tensorflow/lib/python3.6/site-packages/sklearn/metrics/classification.py
:1135: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/sai/.virtualenvs/Tensorflow/lib/python3.6/site-packages/sklearn/metrics/classification.py
:1135: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/sai/.virtualenvs/Tensorflow/lib/python3.6/site-packages/sklearn/metrics/classification.py
:1135: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
Best parameters set found on development set:

{'C': 1, 'kernel': 'linear'}

Grid scores on development set:

1.000 (+/-0.000) for {'C': 1, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 10, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 100, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 1000, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 10000, 'kernel': 'linear'}
0.918 (+/-0.002) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.401 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.995 (+/-0.005) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.970 (+/-0.010) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.999 (+/-0.002) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
1.000 (+/-0.000) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.999 (+/-0.004) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.999 (+/-0.002) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.999 (+/-0.004) for {'C': 10000, 'gamma': 0.001, 'kernel': 'rbf'}
0.999 (+/-0.002) for {'C': 10000, 'gamma': 0.0001, 'kernel': 'rbf'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

          0       1.00      1.00      1.00      1048
          1       1.00      1.00      1.00       252

avg / total       1.00      1.00      1.00      1300


Final accuracy = 1.0
# Tuning hyper-parameters for recall

Best parameters set found on development set:

{'C': 1, 'kernel': 'linear'}

Grid scores on development set:

1.000 (+/-0.000) for {'C': 1, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 10, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 100, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 1000, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 10000, 'kernel': 'linear'}
0.599 (+/-0.013) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.500 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.984 (+/-0.011) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.870 (+/-0.046) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.998 (+/-0.004) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
1.000 (+/-0.002) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.998 (+/-0.005) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.999 (+/-0.002) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.998 (+/-0.005) for {'C': 10000, 'gamma': 0.001, 'kernel': 'rbf'}
0.999 (+/-0.002) for {'C': 10000, 'gamma': 0.0001, 'kernel': 'rbf'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

          0       1.00      1.00      1.00      1048
          1       1.00      1.00      1.00       252

avg / total       1.00      1.00      1.00      1300


Final accuracy = 1.0
```

Now we see that for $C = 1$ and for kernel linear, the model performs best with 100% accuracy!

In [9]:

```
C = 1
clf1 = SVC(kernel='linear', C=C).fit(X_train, y_train)
```

Support Vectors

In [10]:

```
print("Number of support Vectors =", len(clf1.support_))
```

```
Number of support Vectors = 12
```

In [11]:

```
print("Number of suppport vectors for each class:", clf1.n_support_ )
alphas = np.absolute(clf1.dual_coef_)
msv = np.count_nonzero(alphas == C)
print("The margin support vectors =", clf1.dual_coef_.shape[1] - msv)
print("The non-margin support vectors =", msv)
```

```
Number of suppport vectors for each class: [9 3]
The margin support vectors = 10
The non-margin support vectors = 2
```

$$E[Out\_Sample\_Error] \leq \frac{E[Number\_of\_Support\_Vectors]}{N-1}$$

In [12]:

```
mean = np.array(means).mean()
if mean < len(clf1.support_) / (X_train.shape[0] - 1):
    print("The condition holds true")
else:
    print("The condition is false")
```

```
The condition is false
```

Classification report

In [13]:

```
print(classification_report(y_true, y_pred))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      1048
          1       1.00      1.00      1.00       252

avg / total       1.00      1.00      1.00      1300
```

PCA for K = 2

In [14]:

```
X_train, X_test, y_train, y_test = train_test_split(train, test, test_size=0.2, random_state=42)
```

In [15]:

```
pca = PCA(n_components=6)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

Grid search to find the best hyper-parameters with Cross validation

In [16]:

```
tuned_parameters = [
  {'C': [1, 10, 100, 1000, 10000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000, 10000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
scores_list = ['precision', 'recall']
for score in scores_list:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(SVC(), tuned_parameters, cv=5,
                       scoring='%s_macro' % score)
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()
    print("Final accuracy =", accuracy_score(y_test, clf.predict(X_test)))
```

```
# Tuning hyper-parameters for precision


/home/sai/.virtualenvs/Tensorflow/lib/python3.6/site-packages/sklearn/metrics/classification.py
:1135: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/sai/.virtualenvs/Tensorflow/lib/python3.6/site-packages/sklearn/metrics/classification.py
:1135: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/sai/.virtualenvs/Tensorflow/lib/python3.6/site-packages/sklearn/metrics/classification.py
:1135: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/sai/.virtualenvs/Tensorflow/lib/python3.6/site-packages/sklearn/metrics/classification.py
:1135: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/sai/.virtualenvs/Tensorflow/lib/python3.6/site-packages/sklearn/metrics/classification.py
:1135: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
Best parameters set found on development set:

{'C': 1, 'kernel': 'linear'}

Grid scores on development set:

1.000 (+/-0.000) for {'C': 1, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 10, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 100, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 1000, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 10000, 'kernel': 'linear'}
0.918 (+/-0.002) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.401 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.995 (+/-0.005) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.970 (+/-0.010) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.999 (+/-0.002) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
1.000 (+/-0.000) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.999 (+/-0.004) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.999 (+/-0.002) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.999 (+/-0.004) for {'C': 10000, 'gamma': 0.001, 'kernel': 'rbf'}
0.999 (+/-0.002) for {'C': 10000, 'gamma': 0.0001, 'kernel': 'rbf'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

          0       1.00      1.00      1.00      1048
          1       1.00      1.00      1.00       252

avg / total       1.00      1.00      1.00      1300


Final accuracy = 1.0
# Tuning hyper-parameters for recall

Best parameters set found on development set:

{'C': 1, 'kernel': 'linear'}

Grid scores on development set:

1.000 (+/-0.000) for {'C': 1, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 10, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 100, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 1000, 'kernel': 'linear'}
1.000 (+/-0.000) for {'C': 10000, 'kernel': 'linear'}
0.599 (+/-0.013) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.500 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.984 (+/-0.011) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.870 (+/-0.046) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.998 (+/-0.004) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
1.000 (+/-0.002) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.998 (+/-0.005) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.999 (+/-0.002) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.998 (+/-0.005) for {'C': 10000, 'gamma': 0.001, 'kernel': 'rbf'}
0.999 (+/-0.002) for {'C': 10000, 'gamma': 0.0001, 'kernel': 'rbf'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

          0       1.00      1.00      1.00      1048
          1       1.00      1.00      1.00       252

avg / total       1.00      1.00      1.00      1300


Final accuracy = 1.0
```

Now we see that for $C = 1$ and for kernel linear, the model performs best with 100% accuracy!

In [17]:

```
C = 1
clf2 = SVC(kernel='linear', C=C).fit(X_train, y_train)
```

Support Vectors

In [18]:

```
print("Number of support Vectors =", len(clf2.support_))
```

```
Number of support Vectors = 12
```

In [19]:

```
print("Number of suppport vectors for each class:", clf1.n_support_ )
alphas = np.absolute(clf1.dual_coef_)
msv = np.count_nonzero(alphas == C)
print("The margin support vectors =", clf1.dual_coef_.shape[1] - msv)
print("The non-margin support vectors =", msv)
```

```
Number of suppport vectors for each class: [9 3]
The margin support vectors = 10
The non-margin support vectors = 2
```

$$E[Out\_Sample\_Error] \leq \frac{E[Number\_of\_Support\_Vectors]}{N-1}$$

In [20]:

```
mean = np.array(means).mean()
if mean < len(clf1.support_) / (X_train.shape[0] - 1):
    print("The condition holds true")
else:
    print("The condition is false")
```

```
The condition is false
```

Classification report

In [21]:

```
print(classification_report(y_true, y_pred))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      1048
          1       1.00      1.00      1.00       252

avg / total       1.00      1.00      1.00      1300
```

Analysis

1. In both the cases for $k = 6$ and $k = 2$, we get a cent percent accuracy.
2. Due to the less number of dimesnions there maybe a chance of over-fitting
3. The generalisation error condition does not satisfy for both the K values as these models make no error
4. The number of support vectors learned are very less
5. Since, the support vectors are less, we in general can expect less generalisation error.
6. With RBF kernel a higher C value seems to perform better
7. With linear kernel, there is not much change with the change in C value
8. The mean accuracy across folds is almost always more than 90%

In [ ]:

```
```