Assignment 3 on SVM

By K. Sai Somanath, 18MCMT28

Get Data

Un-comment the following lines to download the data...

In [ ]:

```
!mkdir -p Data/Arcene
!wget -O Data/Arcene/test_labels https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/arcene_val
id.labels
!wget -O Data/Arcene/train_labels https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/ARCENE/ar
cene_train.labels
!wget -O Data/Arcene/train  https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/ARCENE/arcene_t
rain.data
!wget -O Data/Arcene/test https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/ARCENE/arcene_val
id.data
```

```
--2018-10-28 23:12:27--  https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/arcen
e_valid.labels
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: 256 [text/plain]
Saving to: ‘Data/Arcene/test_labels’

Data/Arcene/test_la 100%[===================>]     256  --.-KB/s    in 0s

2018-10-28 23:12:29 (6.57 MB/s) - ‘Data/Arcene/test_labels’ saved [256/256]

--2018-10-28 23:12:29--  https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/ARCEN
E/arcene_train.labels
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: 256 [text/plain]
Saving to: ‘Data/Arcene/train_labels’

Data/Arcene/train_l 100%[===================>]     256  --.-KB/s    in 0s

2018-10-28 23:12:30 (6.51 MB/s) - ‘Data/Arcene/train_labels’ saved [256/256]

--2018-10-28 23:12:30--  https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/ARCEN
E/arcene_train.data
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: 2715582 (2.6M) [text/plain]
Saving to: ‘Data/Arcene/train’

Data/Arcene/train   100%[===================>]   2.59M   266KB/s    in 12s

2018-10-28 23:12:43 (227 KB/s) - ‘Data/Arcene/train’ saved [2715582/2715582]

--2018-10-28 23:12:43--  https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/ARCEN
E/arcene_valid.data
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: 2723110 (2.6M) [text/plain]
Saving to: ‘Data/Arcene/test’

Data/Arcene/test     28%[====>               ] 760.00K   146KB/s    eta 15s
```

In [ ]:

```
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
```

Extracting data of Arcene Dataset

In [ ]:

```
DATA_PATH = 'Data/Arcene'
TRAIN_FILE = DATA_PATH + '/train'
TRAIN_LABELS_FILE = DATA_PATH + '/train_labels'
TEST_FILE = DATA_PATH + '/test'
TEST_LABELS_FILE = DATA_PATH + '/test_labels'
```

In [ ]:

```
file_handle = open(TRAIN_FILE)
train = np.array([list(map(int, file_handle.readline().strip().split(' '))) for _ in range(100)], dtype='float64')
train.shape
```

In [ ]:

```
file_handle = open(TEST_FILE)
test = np.array([list(map(int, file_handle.readline().strip().split(' '))) for _ in range(100)], dtype='float64')
test.shape
```

In [ ]:

```
file_handle = open(TRAIN_LABELS_FILE)
y_train = np.array([int(file_handle.readline().strip()) for _ in range(100)])
```

In [ ]:

```
file_handle = open(TEST_LABELS_FILE)
y_test = np.array([int(file_handle.readline().strip()) for _ in range(100)])
```

Apply PCA transformation to reduce the dimensions of the data

K = 100

In [ ]:

```
pca = PCA(n_components=100)
pca.fit(train)
```

In [ ]:

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

Applying Grid Search to find the best parameters

In [ ]:

```
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)))
```

Now, that its clear the $C = 1$ and linear kernel gives the best results, we used them to train the model. Also, the accuracy on the test set is about 83%

In [ ]:

```
clf_final = SVC(kernel='linear', C=1)
clf_final.fit(X_train, y_pred)
print(means)
```

Support Vectors

In [ ]:

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

In [ ]:

```
print("Number of suppport vectors for each class:", clf_final.n_support_ )
print("The margin support vectors =", clf_final.dual_coef_.shape[1])
print("The non-margin support vectors = 0")
```

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

In [ ]:

```
mean = np.array(means).mean()
if mean < len(clf_final.support_) / 99:
    print("The condition holds true")
else:
    print("The condition is false")
```

Classification report

In [ ]:

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

K = 10

In [ ]:

```
# k = 10
scaler = MinMaxScaler()
scaler.fit(train)
X_train_s = scaler.transform(train)
X_test_s = scaler.transform(test)
pca = PCA(n_components=10)
pca.fit(X_train_s)
X1_train = pca.transform(X_train_s)
X1_test = pca.transform(X_test_s)
```

Applying grid search for the best parameters

In [ ]:

```
tuned_parameters = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.01, 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(X1_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(X1_test)
    print(classification_report(y_true, y_pred))
    print()
    print("Final accuracy =", accuracy_score(y_test, clf.predict(X1_test)))
```

In [ ]:

```
c = 1000
clf_final_2 = SVC(kernel='rbf', C=c, gamma=0.001)
clf_final_2.fit(X1_train, y_train)
y_true, y_pred = y_test, clf_final_2.predict(X1_test)
print(classification_report(y_true, y_pred))
print("Final accuracy =", accuracy_score(y_test, clf_final_2.predict(X1_test)))
```

Now, that its clear the $C = 1000$, $\gamma = 0.001$ and RBF kernel gives the best results, we used them to train the model. Also, the accuracy on the test set is about 85%

Support Vectors

In [ ]:

```
print("Number of support Vectors =", len(clf_final_2.support_))
print("Number of suppport vectors for each class:", clf_final_2.n_support_ )
alphas = np.absolute(clf_final_2.dual_coef_)
msv = np.count_nonzero(alphas == c)
print("The margin support vectors =", clf_final_2.dual_coef_.shape[1] - msv)
print("The non-margin support vectors =", msv)
```

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

In [ ]:

```
mean = np.array(means).mean()
if mean < len(clf_final.support_) / 99:
    print("The condition holds true")
else:
    print("The condition is false")
```

Classification report

In [ ]:

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

Analysis

The training proves to be challenging as the data available for training is very less.

1. Grid search had to be used to find the best parameters. In each iteration we exhuastively search for the best parameters using the 5-fold Cross-Validation.
2. In both the cases; k=100, k=10; the formula for the generalisation error is satisfied.
3. The Number of support vectors for both k=10 and k=100 are very high as the data was reduced from a very high dimesion (10000) to a low dimension (100, 10) space.
4. Also, considering the fact that the number of training samples equals number of test sample, and also that the number is a mere 100 points, we get really bad accuracy rates (about 30%) in some folds of cross validation.
5. When k=100, there is not much effect to the accuracy when the C value is changed to 10, 100, 1000, 10000 respectively. The accuracy remains the same.
6. When using the linear kernel, the number of margin support vectors($\alpha = C$) is 0
7. The rbf kernel seems to perform better when the data is reduced to 10 dimensional PCA space.

In [ ]: