VIT UNIVERSITY, ANDHRA PRADESH
School of CSE
CSE3008 - Introduction to Machine Learning
Lab Experiment-10
(**Multiclass Classification using Support Vector Machine, Support vector regression**)

Faculty-**Dr. B. SRINIVASA RAO**

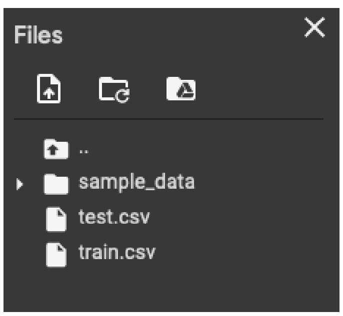
Name-Neeraj Guntuku

R.No-18MIS7071

Slot-L55+L56

Date- 10 April 2021

Multiclass Classification using Support Vector Machine



Training Support Vector Machines for Multiclass Classification

```
[1]
import numpy as np
import pylab as pl
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.utils import shuffle
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.model_selection import cross_val_score, GridSearchCV

import os
print(os.listdir())
```

```
['.config', 'train.csv', 'test.csv', '.ipynb_checkpoints', 'sample_data']
```

Load the Train and Test set

```
[2] train = shuffle(pd.read_csv("train.csv"))
    test = shuffle(pd.read_csv("test.csv"))
```

Check for missing values in the dataset

```
[3] print("Any missing sample in training set:",train.isnull().values.any())
    print("Any missing sample in test set:",test.isnull().values.any(), "\n")
```

```
Any missing sample in training set: False
Any missing sample in test set: False
```

Frequency Distribution of the Outome

```
[4] #Frequency distribution of classes"
    train_outcome = pd.crosstab(index=train["Activity"],  # Make a crosstab
                                columns="count")       # Name the count column

    train_outcome
```

col_0	count
Activity	
LAYING	1407
SITTING	1286
STANDING	1374
WALKING	1226
WALKING_DOWNSTAIRS	986
WALKING_UPSTAIRS	1073

Visualizing Outcome Distribution

```
[5] # Visualizing Outcome Distribution
    temp = train["Activity"].value_counts()
    df = pd.DataFrame({'labels': temp.index,
                       'values': temp.values
                      })

    #df.plot(kind='pie',labels='labels',values='values', title='Activity Ditribution',subplots= "True")

    labels = df['labels']
    sizes = df['values']
    colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral','cyan','lightpink']
    patches, texts = plt.pie(sizes, colors=colors, shadow=True, startangle=90, pctdistance=1.1, labeldistan
    plt.legend(patches, labels, loc="best")
    plt.axis('equal')
    plt.tight_layout()
    plt.show()
```

Normalize the Predictor(Feature Set) for SVM training

```
[6] # Seperating Predictors and Outcome values from train and test sets
    X_train = pd.DataFrame(train.drop(['Activity','subject'],axis=1))
    Y_train_label = train.Activity.values.astype(str)
    X_test = pd.DataFrame(test.drop(['Activity','subject'],axis=1))
    Y_test_label = test.Activity.values.astype(str)

    # Dimension of Train and Test set
    print("Dimension of Train set",X_train.shape)
    print("Dimension of Test set",X_test.shape,"\n")

    # Transforming non numerical labels into numerical labels
    from sklearn import preprocessing
    encoder = preprocessing.LabelEncoder()

    # encoding train labels
    encoder.fit(Y_train_label)
    Y_train = encoder.transform(Y_train_label)

    # encoding test labels
    encoder.fit(Y_test_label)
    Y_test = encoder.transform(Y_test_label)

    #Total Number of Continous and Categorical features in the training set
    num_cols = X_train._get_numeric_data().columns
    print("Number of numeric features:",num_cols.size)
    #list(set(X_train.columns) - set(num_cols))
```

```
names_of_predictors = list(X_train.columns.values)

# Scaling the Train and Test feature set
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
Dimension of Train set (7352, 561)
Dimension of Test set (2947, 561)

Number of numeric features: 561
```

Hyperparameter tuning using grid search and cross validation

```
[7] #Libraries to Build Ensemble Model : Random Forest Classifier
    # Create the parameter grid based on the results of random search
    params_grid = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                         'C': [1, 10, 100, 1000]},
                        {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
```

Training SVM model using radial kernel

```
[8] # Performing CV to tune parameters for best SVM fit
    svm_model = GridSearchCV(SVC(), params_grid, cv=5)
    svm_model.fit(X_train_scaled, Y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                          'kernel': ['rbf']},
                         {'C': [1, 10, 100, 1000], 'kernel': ['linear']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

Confusion Matrix and Accuracy Score

```
[9] # View the accuracy score
    print('Best score for training data:', svm_model.best_score_,"\n")

    # View the best parameters for the model found using grid search
    print('Best C:',svm_model.best_estimator_.C,"\n")
    print('Best Kernel:',svm_model.best_estimator_.kernel,"\n")
    print('Best Gamma:',svm_model.best_estimator_.gamma,"\n")

    final_model = svm_model.best_estimator_
    Y_pred = final_model.predict(X_test_scaled)
    Y_pred_label = list(encoder.inverse_transform(Y_pred))
```

```
Best score for training data: 0.9865340344159417

Best C: 1000

Best Kernel: rbf

Best Gamma: 0.001
```

```
[10] # Making the Confusion Matrix
     #print(pd.crosstab(Y_test_label, Y_pred_label, rownames=['Actual Activity'], colnames=['Predicted Activ
     print(confusion_matrix(Y_test_label,Y_pred_label))
     print("\n")
     print(classification_report(Y_test_label,Y_pred_label))

     print("Training set score for SVM: %f" % final_model.score(X_train_scaled , Y_train))
     print("Testing  set score for SVM: %f" % final_model.score(X_test_scaled  , Y_test ))

     svm_model.score
```

```
[[537   0   0   0   0   0]
 [  3 439  48   0   0   1]
 [  0  11 521   0   0   0]
 [  0   0   0 486   4   6]
 [  0   0   0   6 389  25]
 [  0   0   0  15   2 454]]
```

```
[[537   0   0   0   0   0]
 [  3 439  48   0   0   1]
 [  0  11 521   0   0   0]
 [  0   0   0 486   4   6]
 [  0   0   0   6 389  25]
 [  0   0   0  15   2 454]]


                    precision    recall  f1-score   support

            LAYING       0.99      1.00      1.00       537
           SITTING       0.98      0.89      0.93       491
          STANDING       0.92      0.98      0.95       532
           WALKING       0.96      0.98      0.97       496
WALKING_DOWNSTAIRS       0.98      0.93      0.95       420
  WALKING_UPSTAIRS       0.93      0.96      0.95       471

          accuracy                           0.96      2947
         macro avg       0.96      0.96      0.96      2947
      weighted avg       0.96      0.96      0.96      2947

Training set score for SVM: 1.000000
Testing  set score for SVM: 0.958941
<bound method BaseSearchCV.score of GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                          'kernel': ['rbf']},
                         {'C': [1, 10, 100, 1000], 'kernel': ['linear']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)>
```

Support vector regression

Support Vector Regression

Importing the libraries

```
[1] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
```

Importing the dataset

```
[2] dataset = pd.read_csv('SampleData.csv')
    X = dataset.iloc[:, 0].values
    y = dataset.iloc[:, 1].values
    y = np.array(y).reshape(-1,1)
    dataset.head(5)
```

	Hours of Study	Marks
0	32.502345	31.707006
1	53.426804	68.777596
2	61.530358	62.562382
3	47.475640	71.546632
4	59.813208	87.230925

Feature Scaling

```
[3] from sklearn.preprocessing import StandardScaler
    sc_X = StandardScaler()
    sc_y = StandardScaler()
    X = sc_X.fit_transform(X.reshape(-1,1))
    y = sc_y.fit_transform(y.reshape(-1,1))
```

Splitting the dataset into the Training set and Test set

```
[4] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

Training the Support Vector Regression model on the Training set

```
[5] from sklearn.svm import SVR
    regressor = SVR(kernel = 'rbf')
    regressor.fit(X_train.reshape(-1,1), y_train.reshape(-1,1))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A
  y = column_or_1d(y, warn=True)
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

Predicting the Results

```
[6] y_pred = regressor.predict(X_test)
    y_pred = sc_y.inverse_transform(y_pred)
    y_pred
```

```
array([78.86474326, 93.07233808, 82.04137655, 79.79709186, 76.72265591,
       90.54612743, 60.38192467, 58.55099588, 71.12798091, 62.11580563,
       69.89505864, 59.0668753 , 61.31339316, 84.32726724, 85.42956343,
       59.33656194, 74.69547341, 76.54528267, 53.41532363, 68.30603506])
```

Comparing the Real Values with Predicted Values

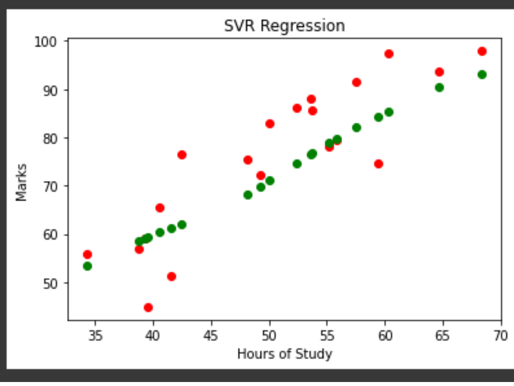
```
[7] df = pd.DataFrame({'Real Values':sc_y.inverse_transform(y_test.reshape(-1)), 'Predicted Values':y_pred})
    df
```

	Real Values	Predicted Values
0	78.211518	78.864743
1	97.919821	93.072338
2	91.486778	82.041377
3	79.550437	79.797092
4	85.668203	76.722656
5	93.576119	90.546127
6	65.562301	60.381925
7	56.877213	58.550996
8	82.905981	71.127981
9	76.617341	62.115806
10	72.111832	69.895059
11	59.171489	59.066875
12	51.391744	61.313393
13	74.765564	84.327267
14	97.379897	85.429563

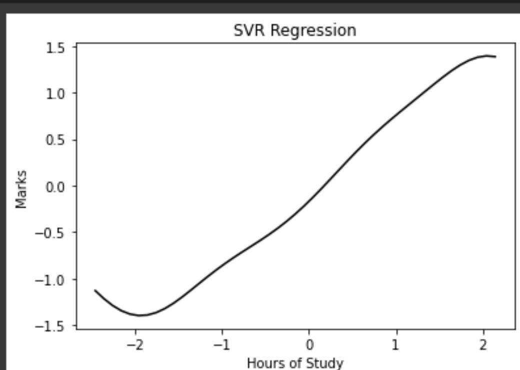
***

Visualising the SVR Results

```
[8] # Visualising the SVR results (for higher resolution and smoother curve)
    X_grid = np.arange(min(X), max(X), 0.1)
    X_grid = X_grid.reshape((len(X_grid), 1))
    plt.scatter(sc_X.inverse_transform(X_test), sc_y.inverse_transform(y_test.reshape(-1)), color = 'red')
    plt.scatter(sc_X.inverse_transform(X_test), y_pred, color = 'green')
    plt.title('SVR Regression')
    plt.xlabel('Hours of Study')
    plt.ylabel('Marks')
    plt.show()
```




```
[9] plt.plot(X_grid, regressor.predict(X_grid), color = 'black')
    plt.title('SVR Regression')
    plt.xlabel('Hours of Study')
    plt.ylabel('Marks')
    plt.show()
```



..
sample_data
SampleData.csv

***