Design and Implementation of Support Vector Machines

Aditi Helekar

March 24, 2017

Contents:

1. Problem Statement……..2
2. Approach…….....3
3. Implementation with code snippets…….4
4. Observations……....6
5. Conclusion……...8
6. References……...9

Abstract

A brief introduction to Support Vector Machines is provided and is implemented in Python. The implementation is demonstrated to identify the appropriate prediction on Testing data by training the models with Digit Recognition dataset and Amazon Baby products reviews dataset. There is a comparison mentioned about the kernel functions used to both the datasets.

1. Problem Statement:

- Implement or train an SVM model using the Digit Recognition and Amazon Reviews datasets.
- Submit your implementation and analysis results showing the various settings/parameters you experimented with.

2. Approach:

I have implemented SVM in Python using scikit-learn library. Scikit-learn is library designed for implementing Machine learning techniques in Python. Scikit-learn provides libraries Support Vector Machines.

Support Vector Machines(SVM):

The SVMs are set of supervise learning methods used for classification, regression and outliers detection. There are good advantages to using SVM such as it is very effective in high dimensional spaces. It is also effective when number of dimensions is greater than the number of samples. SVM also allows us to use different kernel functions. If the number of features is much greater than the number of samples, the methods is likely to give poor performances. SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None)

C: Penalty parameter of C for error term.

Kernel: Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape

degree: Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

3. Implementation:

The basic algorithm used for both implementations is described below:
1. Read the csv file using pandas.io library and store it as a Dataframe.
2. Classify the attributes as samples and features to apply it to SupportVectorMachines classifier. Features are also referred as predictors and samples as responses.
3. Create an object of Classifier imported from scikit learn library.
4. Using fit function, fit the dataframe to tree model. Fit function finds patterns in the given dataset.
5. For cross-validation, divide the training dataset in train and test data using train_test_split function.
6. Predict the behavior of model using predict function with test data.
7. After cross validation use the test dataset and predict again.
8. Calculate accuracy using score function.
9. Plot the graphs using matplotlib library.

For Amazon Dataset, I first removed all the stopwords using the nltk toolkit. Later on, I converted the words to vectors and found the occurrences of each meaningful words using CountVectorizer.

Support Vector Machines :

I have verified the results by modifying value of C and kernel fuction. Below code snippet shows the outline of SVM for Digit Recognition Dataset:

```
print("Test Sample 0 rbf kernel")

new_classifier0 = svm.SVC(kernel="rbf")
new_classifier0 = new_classifier0.fit(X_train, Y_train)

new_predictions0 = new_classifier0.predict(X_test)
sklearn.metrics.confusion_matrix(Y_test,new_predictions0)
print("Testing Accuracy 1:")
accuracy0 = sklearn.metrics.accuracy_score(Y_test, new_predictions0)
print(accuracy0)
```

The above code shows the implementation of ‘rbf’ kernel function.

```
classifier3 = svm.SVC(kernel="linear", C=0.001)
classifier3 = classifier3.fit(pred_train, tar_train)

predictions3 = classifier3.predict(pred_test)
#print(predictions.shape)

sklearn.metrics.confusion_matrix(tar_test,predictions3)
print("Training Accuracy 3:")
trainscore3 = sklearn.metrics.accuracy_score(tar_test, predictions3)
print(trainscore3)
c, r = targets.shape
targets = targets.values.reshape(c,)

scores = cross_val_score(classifier3,predictors, targets, cv=5)
print("Scores: ", scores)
```

The above code shows ‘linear’ kernel function with cross- validation giving different scores for 5 iterations.

4. **Observations:**

Support Vector Machines :

1. For Digit Recognition dataset:

Sample rbf
Training Accuracy 1:
0.490516677567
****
Sample 1
Training Accuracy 1:
0.982341399608
****
Sample 2
Training Accuracy 2:
0.982341399608
****
Sample 3
Training Accuracy 3:
0.982995421844
Scores: [0.97916667 0.98826597 0.96985583 0.98165138 0.98291721]
****
Sample 4
Training Accuracy 4:
0.904512753434
****
Test Sample 0 rbf kernel
Testing Accuracy 1:
0.561804008909
****
Test Sample 1
Testing Accuracy 1:
0.961024498886
****
Test Sample 2
Testing Accuracy 2:
0.961024498886
****
Test Sample 3
Testing Accuracy 3:
0.963808463252
****
Test Sample 4
Testing Accuracy 2:
0.951559020045

Sample rbf: classifier0 = svm.SVC(kernel="rbf")
Sample 1: classifier = svm.SVC(kernel="linear")
Sample 2: classifier2 = svm.SVC(kernel="linear", C=10)
Sample 3: classifier3 = svm.SVC(kernel="linear", C=0.001)
Sample 4: classifier4 = svm.SVC(kernel="linear", C=0.00001)

We get maximum accuracy for Sample 2 with C = 0.001, leaf size of 5, linear kernel function.

2. <u>For Amazon dataset:</u>

   The output with kernel function rbf and C=5.

```
[5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5]
Accuracy:
0.58
```

5. Conclusion:

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.

For Digit recognition dataset:

1. It was found that linear kernel suited best giving higher accuracy than rbf kernel.
2. There was a significant observation about the C term on the accuracy. For C = 1 to C = 10 or greater the accuracy remains constant. If we decrease C the at C = 0.001 we get maximum accuracy. But as we further decrease the value of C, the accuracy decreases.

For Amazon dataset:

1. It was observed that linear function kernel does not give better output rather polynomial kernel function and rbf kernel function give better accuracy.

6. References:

1. Support Vector machines, http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
2. Cross-validation, http://scikit-learn.org/stable/modules/cross_validation.html