INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Department of Computer Science and Engineering

CS60050 – Machine Learning

Project - 2

Predict Behavior using Support Vector Machines

Submitted By: (GROUP – 15)

- Kamal Kyal - 19CE10035
- Rishi Suman - 19EC39045
- Shailesh Chaudhary - 22CS60R37

Introduction

- **Support Vector Machines** (SVMs in short) are machine learning algorithms that are used for classification and regression purposes. SVMs are one of the powerful machine learning algorithms for classification, regression and outlier detection purposes. An SVM classifier builds a model that assigns new data points to one of the given categories. Thus, it can be viewed as a non-probabilistic binary linear classifier.

- SVMs can be used for linear classification purposes. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using the **kernel trick**. It enable us to implicitly map the inputs into high dimensional feature spaces.

Hyperplane

- A hyperplane is a decision boundary which separates between given set of data points having different class labels. The SVM classifier separates data points using a hyperplane with the maximum amount of margin. This hyperplane is known as the maximum margin hyperplane and the linear classifier it defines is known as the maximum margin classifier.

Support Vectors

- Support vectors are the sample data points, which are closest to the hyperplane. These data points will define the separating line or hyperplane better by calculating margins.

Margin

- A margin is a separation gap between the two lines on the closest data points. It is calculated as the perpendicular distance from the line to support vectors or closest data points. In SVMs, we try to maximize this separation gap so that we get maximum margin.

Kernel trick

- In practice, SVM algorithm is implemented using a kernel. It uses a technique called the kernel trick. In simple words, a kernel is just a function that maps the data to a higher dimension where data is separable. A kernel transforms a low-dimensional input data space into a higher dimensional space. So, it converts non-linear separable problems to linear separable problems by adding more dimensions to it. Thus, the kernel trick helps us to build a more accurate classifier. Hence, it is useful in non-linear separation problems

Algorithm

- The type of dataset used in recording performance entry

- Dataset for training, validation and test metric generation
- Indices for inputing in recorded table for validation scores
- Indices for inputing in recorded table for test scores of best performing on validation model
- Table for storing validation scores
- Table for storing test scores
- Try different kernel types by varying the appropriate hyperparameters of the classifier and compute the classification accuracy on the validation split.

  Type of Kernel used:
  a. Linear
  b. Poly
  c. Rbf
  d. Sigmoid
  e. Precomputed
- In last we are predicting the best model for given dataset.

<u>Result of validation and test on PCA dataset</u>

1. Table containing all accuracies for validation and test on PCA dataset:

Sr. No.	Data	C	Kernel	Gamma	Score
0	custom PCA data	1.0	linear	scale	0.855856
1	custom PCA data	1.0	linear	auto	0.855856
2	custom PCA data	1.0	poly	scale	0.855856
3	custom PCA data	1.0	poly	auto	0.855856
4	custom PCA data	1.0	rbf	scale	0.855856
5	custom PCA data	1.0	rbf	auto	0.855856
6	custom PCA data	10.0	linear	scale	0.855856
7	custom PCA data	10.0	linear	auto	0.855856
8	custom PCA data	10.0	poly	scale	0.855856

9	custom PCA data	10.0	poly	auto	0.855856
10	custom PCA data	10.0	rbf	scale	0.855856
11	custom PCA data	10.0	rbf	auto	0.855856
12	custom PCA data	100.0	linear	scale	0.855856
13	custom PCA data	100.0	linear	auto	0.855856
14	custom PCA data	100.0	poly	scale	0.855856
15	custom PCA data	100.0	poly	auto	0.855856
16	custom PCA data	100.0	rbf	scale	0.855856
17	custom PCA data	100.0	rbf	auto	0.855856
18	inbuilt PCA data	1.0	linear	scale	0.873874
19	inbuilt PCA data	1.0	linear	auto	0.873874
20	inbuilt PCA data	1.0	poly	scale	0.873874
21	inbuilt PCA data	1.0	poly	auto	0.873874
22	inbuilt PCA data	1.0	rbf	scale	0.873874
23	inbuilt PCA data	1.0	rbf	auto	0.873874
24	inbuilt PCA data	10.0	linear	scale	0.873874
25	inbuilt PCA data	10.0	linear	auto	0.873874
26	inbuilt PCA data	10.0	poly	scale	0.873874
27	inbuilt PCA data	10.0	poly	auto	0.873874
28	inbuilt PCA data	10.0	rbf	scale	0.873874
29	inbuilt PCA data	10.0	rbf	auto	0.873874
30	inbuilt PCA data	100.0	linear	scale	0.873874
31	inbuilt PCA data	100.0	linear	auto	0.873874

32	inbuilt PCA data	100.0	poly	scale	0.873874
33	inbuilt PCA data	100.0	poly	auto	0.873874
34	inbuilt PCA data	100.0	rbf	scale	0.873874
35	inbuilt PCA data	100.0	rbf	auto	0.873874

Accuracy of the Model = 85%

2. Test accuracy of Kernel using which the Validation accuracy is highest:

Sr. No.	Data	C	Kernel	Gamma	Score
0	custom PCA data	100.0	rbf	auto	0.848576
1	inbuilt PCA data	100.0	rbf	auto	0.851574

Accuracy of the Model = 87%

Result of validation and test on LDA dataset

1. Table containing all accuracies for validation and test on PCA dataset:

Sr. No.	Data	C	Kernel	Gamma	Score
0	custom LDA data	1.0	linear	scale	0.84985
1	custom LDA data	1.0	poly	scale	0.84985
2	custom LDA data	1.0	rbf	scale	0.84985
3	custom LDA data	10.0	linear	scale	0.84985
4	custom LDA data	10.0	poly	scale	0.84985
5	custom LDA data	10.0	rbf	scale	0.84985
6	inbuilt LDA data	1.0	linear	scale	0.855856
7	inbuilt LDA data	1.0	poly	scale	0.855856
8	inbuilt LDA data	1.0	rbf	scale	0.855856
9	inbuilt LDA	10.0	linear	scale	0.855856

	data				
10	inbuilt LDA data	10.0	poly	scale	0.855856
11	inbuilt LDA data	10.0	rbf	scale	0.852853

Accuracy of the Model = 84%

2. Test accuracy of Kernel using which the Validation accuracy is highest:

Sr. No.	**Data**	**C**	**Kernel**	**Gamma**	**Score**
0	custom LDA data	10.0	rbf	scale	0.86057
1	inbuilt LDA data	10.0	rbf	scale	0.836582

Accuracy of the Model = 85%

CONCLUSION

We implemented the SVM using Python from scratch, and find the accuracy of our model Using different kernel function for PCA and LDA. For PCA our model give the accuracy score around 85% on validation data set and for test dataset it give accuracy score around 87% . For LDA our model give the accuracy score around 84% on validation data set and for test dataset it give accuracy score around 85% .