CSE 569: Fundamentals of Statistical Learning and Pattern Recognition

Report on Project Part-2: Experimenting with SVM

by Rohan Sambidi

Introduction

This aim of this project is to understand the usage of a popular SVM library: LIBSVM. We use this library, in Python or MATLAB, to perform a series of classification tasks on a given dataset.

A dataset is provided that contains data of 50 classes. Therefore, we are performing multi-class classification. The training dataset has 4786 samples, while the testing dataset has 1833 samples. Each sample is described by three feature matrices: $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, and $\boldsymbol{X}_3$, and the category vector is described by $\boldsymbol{Y}$. Each of the three features of a sample of size 1x1000. All the three features are normalized histograms.

There are 3 classification tasks in this project:

- Step-0: Classification by individual features
- Step-1: Feature combination by fusion of classifiers
- Step-2: Feature combination by simple concatenation

For step-0, a multi-class linear SVM classifier is trained on each of the three features from the training set. We get three classifiers— corresponding to each of the three features— which are represented by $h_1(\mathbf{x})$, $h_2(\mathbf{x})$, and $h_3(\mathbf{x})$. These classifiers are then tested on the corresponding features from the testing set. The labels obtained from classifying the testing set are compared with the actual labels and the accuracy is reported.

For the next phase of this step, the SVM classifiers $h_k(\mathbf{x})$, k={1,2,3} are trained for probability estimates. SVM classifiers can be used to obtain the posterior probabilities of samples, for a given feature, that they belong to a certain class. This probability is represented by $p_k(w_i \mid \mathbf{x})$, i.e., the probability that the sample $\mathbf{x}$ belongs to the class w_i given the feature $\boldsymbol{X}_k$. The classifiers are then tested on the testing set and accuracy is reported.

For step-1, the probabilities obtained in the above step are fused as $p(w_i \mid \mathbf{x}) = \Sigma_k\, p_k(w_i \mid \mathbf{x}) / 3$. Then, for each sample in the test dataset, the maximum probability among the 50 probabilities is noted and the class corresponding to that probability is assigned to the sample. The accuracy is reported.

For step-2, the three features are concatenated to obtain a single feature $\boldsymbol{X}$. This feature is then used for training a multi-class linear SVM. The concatenated feature will be of size 1x3000. The model is tested on the test set for comparing with the models obtained in the initial part of step-0, and the accuracy is reported.

Method

LIBSVM is an integrated software for support vector classification, regression, and distribution estimation. This library is very useful for understating the workflow of SVM, especially when using the graphic interface, since it allows modifying the parameters for training. The library is supported by many languages and is comparatively efficient to work with.

For this project, I used Python programming language. I first installed the LIBSVM package using pip command and read the documentation for various classes and methods from GitHub directory. First thing I noticed is that this library requires a specific format for data to train the models. The given data for this project is a MATLAB file. Though *numpy* arrays can be used to format this data as required, I wanted to store the data into a separate file in the format specified in the documentation. Hence, I wrote a program to convert the data into the desired format and sorted it in a text file. I created a separate file for each of the three features in training and testing set. Thus, I obtained a total of six files. I verified the format of the data in all these files using the *checkdata.py* file mentioned in the directory. Then I used this data for training the necessary SVM models.

Step-0:
To read the data, I used svm_read_problem() function. The data file in proper format is passed as the argument to this function. It will return two lists- one with features and the other with labels in the dataset. To train a model, svm_train() function is used. This function has various parameters which allows the user to train the model as per their requirement. These parameters are used for various requirements training such as assigning cache memory, setting the training kernel, setting tolerance for termination, and other training parameters for SVM. As stated in the project instructions, I set the cost parameter '-c' to 10, and the kernel_type parameter '-t' to 0. The default value for cost parameter is 1. Since this value is increased to 10, the model will be heavily penalized for misclassification. A value of 0 for the kernel_type parameter ensures that the SVM training is done using a linear kernel. The rest of the parameters are left default. I used the quiet flag '-q' while training to stop getting output for each update made during the training. This will slightly help speed-up the training process, since the function doesn't have to print anything in the middle of training. The svm_train() function returns a trained SVM model, which is an instance of the svm_model class. Various functions are provided in the library to obtain useful data, such as the support-vectors, support-vector coefficients, etc., from this instance.

For testing, *svm_predict*() function is used. This functions has three arguments: actual labels of test data samples, features of the test data samples, and the *svm_model* instance. It has n optional argument that can be used to specify the options as in the *svm_train*() function. This function predicts the labels for the given samples and returns the predicted labels, performance metrics, and a list of decision values or probability estimates. To get the probability estimates from the SVM, set the '-b' parameter to 1 for both *svm_train*() and *svm_predict*() functions. Training the model for probabilities took longer time. The reason, as provided in the LIBSVM FAQ page, is that the function performs cross-validation on the predictions. I used the outputs of the *svm_predict*() function to report the accuracy of the classification. This will give the necessary outputs for the step-0.

Step-1:
The probability estimates are a list of probabilities that a sample belong to a certain class. Since there are 50 classes in the data, the probability estimates for each sample is a list of 50 real numbers. To fuse the probability estimates corresponding to the three features, I treated the probability estimates as *numpy* array and added them element-wise. Then I normalized them. Then, I obtained the class label corresponding to the maximum of the 50 probabilities for a sample. This will give a list of labels for samples in the test set. I compared these labels to the actual labels to get the accuracy of classification over the test set. The *evaluation*() function provided in the LIBSVM library can also be used to get the accuracy and other metrics such as MSE (Mean Squared Error) and SCC (Squared Correlation Coefficient).

Step-2:
To concatenate the features, I used the concatenate() function form the numpy module. This results in a feature of size 1x3000. Once the features are concatenated in training set and testing set, I converted them into the LIBSVM format. Then, similar to the approach in step-0, I trained the model and tested it and obtained the accuracy.

Results

For reporting the classification accuracy, I'm using the output of *svm_predict*() function.

- Step-0
  - Part-1:
    - When feature $\boldsymbol{X}_1$ is used, Accuracy = 10.7807% (203/1883) (classification)
    - When feature $\boldsymbol{X}_2$ is used, Accuracy = 16.6755% (314/1883) (classification)
    - When feature $\boldsymbol{X}_3$ its used, Accuracy = 8.92193% (168/1883) (classification)
  - Part-2:
    - When feature $\boldsymbol{X}_1$ is used, Accuracy = 27.9341% (526/1883) (classification)
    - When feature $\boldsymbol{X}_2$ is used, Accuracy = 27.0844% (510/1883) (classification)
    - When feature $\boldsymbol{X}_3$ its used, Accuracy = 26.5003% (499/1883) (classification)
- Step-1
  - Accuracy = 44.0255% (829/1883)
- Step-2
  - Accuracy = 37.0685% (698/1883) (classification)

Observation

From these results, it it clear that using the raw features for training gave the worst accuracy. When the models are trained for probability estimates, they performed better on the test set. This may not be the case every time. There is no guarantee for the probability model in part-2 of step-0 to perform better than the model in part-1 of step-0. This hugely depend on the features provided for training. The different accuracies is due to the fact that when the posterior probabilities are considered, the decision rule will be updated dynamically as the training progresses, which will help the model to make better predictions. If proper parameters are selected, both the models can give similar accuracy.

When the probabilities are fused to make predictions, the best accuracy was achieved. This accuracy is significantly better than the accuracies achieved previously. It is almost twice as much as the previous accuracies. This can be due to fact that the probabilities are fused, which mean the information from all the three features are encoded within this new feature. This might have helped the model to classify the samples based on better metrics, which were not available to the models in the previous step.

The model trained on a concatenation the three features gave better accuracy than the models trained on individual features. The accuracy is increased by more than three folds. This might be because having more features for samples helped the model distinguish between the samples clearly.

Conclusion

The LIBSVM library is very handy in understanding the working of SVM. It can help in better understanding of the SVM architecture by allowing the user to choose between wide range of parameters and utility functions.

For classification, posterior probabilities can help improve the performance of a classifier. As seen in the results, using posterior probabilities gave the best accuracy. Selecting the right number of features for classification is also important. Choosing the right number of features will help the classifier to perform better. However, too many features might also slow the inference process and lead to poor accuracies.