Machine Learning HW06

Siffi Singh – 0660819

Program to classify hand written digits using PCA for dimensionality reduction and SVM functions from LIBSVM library for data classification.

OUTLINE

In this report, I will be explaining:

1. Explanation of Code Logic

- ***a. Performing SVM and getting best C and best gamma using Grid Search***
- ***b. Performing PCA to reduce to 2D.***
- ***c. Plotting the data.***
- ***d. Plotting decision boundary and support vectors***

2. Testing Performance

3. Inferences from Result

4. Conclusion

1. Explanation of Code Logic

The program for support vector machine and visualization on 2D space has been done using LIBSVM for MATLAB.

Introduction to LIBSVM:
The functions used from LIBSVM in this homework are the libsvmread(), svmtrain() and svmpredict(). Using LIBSVM, we are also able to use model.SVs which helps us in plotting support vectors.

The code logic is straight forward. The overall homework is divided into four parts, which is explained separately one by one in this report. The performance in each case has been noted and results have been inferred for increasing the understanding of how the algorithm is working. MATLAB helps us with certain functions that makes it easier to plot and visualize the results. Also, searching for the best parameters 'C' for C-SVC and 'gamma' for RBF kernel using grid search ensures optimum performance in cross-validation of the algorithm, visualizing it also helps in understanding how different combination of parameters can affect the accuracy of the algorithm.

1.a. Performing SVM and getting best C and best gamma using Grid Search

Performing SVM in MATLAB using LIBSVM in MATLAB is done in the following way. The submitted code has just two files, one is **main.m** that is responsible for calling the functions that are needed to perform necessary operation, the second is the **support_vector.m** function which has the code for almost everything that is taking place inside this complete program.

From main.m, we pass the "csv files" given to us as the test case which has been done in the following way:

```
%calling support vector machine function for handwritten digits
%classification
[y,d] = support_vector('X_train.csv','X_test.csv', 'T_train.csv',
'T_test.csv');
```

Here, the output 'y' has the predicted labels for comparison and 'd' has the accuracy. Then, in support_vector.m the parameters are taken from main function to read the 'csv files' and store it into a double data type 2D-array, as shown below:

```
function [x,y] = support_vector(train, test, trainlabel, testlabel)

% Reading the data given in csv files
trainData = xlsread(train);
trainLabel = xlsread(trainlabel);
testData = xlsread(test);
testLabel = xlsread(testlabel);
```

After data has been read, we add the path to libsvm toolbox and the path to the data in LIBSVM library, downloaded from LIBSVM.

```
% addpath to the libsvm toolbox
addpath('C:/Users/Siffi Singh/Desktop/libsvm-3.22/libsvm-3.22/matlab');

% addpath to the data
dirData = 'C:/Users/Siffi Singh/Desktop/libsvm-3.22';
addpath(dirData);
```

Now, from here there are mainly two steps in this part:

(i) Grid Search to find the best 'C' and best 'gamma'.

(ii) Support vector classification using C-SVC, the soft margin SVM and RBF Kernel.

(i) Grid Search to find the best 'C' and best 'gamma'

In this step, we perform the Grid search, and how we do that is explained below. Unlike Python, there is no pre-defined function for tuning the parameters of Kernel SVM here, so manually performing grid search to find the best 'c' and best 'gamma' has been done in the following way:

The algorithm for Grid search is a brute force approach, we initially define a range for both 'C' and 'Gamma' on whose combination, we will perform the grid search. In each iteration, we use the 'svmtrain' and check if our current computed value 'cv' using svmtrain is better than or

equal to the old one. Every time we find the bestC = 2^valueC and bestG = 2^valueG and find for all combinations in the specified range. Once we find a 'cv' value that is greater than or equal to the current value, we update the 'bestcv', 'bestC' i.e. the best 'c' and the 'bestg' i.e. the best gamma and we keep on doing it until all the iterations and possible combination of values have been checked. The bestC and bestGamma are the ones, corresponding to which we have the highest cross-validation accuracy.

```
% Grid search to find best_C best_gamma
% grid of parameters
folds = 5;
bestcv = 0;
for log2c = -1:1:1
  for log2g = -1:1:1
    cmd = ['-q -c ', num2str(2^log2c), ' -g ', num2str(2^log2g)];
    cv = svmtrain(trainLabel, trainData, sprintf('-c %f -g %f -v %d',
2^log2c, 2^log2g, folds));
    if (cv >= bestcv)
      bestcv = cv; bestc = 2^log2c; bestg = 2^log2g;
    end
    fprintf('%g %g %g (best c=%g, g=%g, rate=%g)\n', log2c, log2g, cv, bestc,
bestg, bestcv);
  end
end
```

The grid search algorithm works efficiently with finding the bestC and bestGamma, and using this bestC and bestGamma, we do the further calculations:

```
% Training data with best_C and best_gamma
model = svmtrain(trainLabel, [(1:5000)' trainData*trainData'], sprintf('-c %f
-g %f -b 1 -t 4', best_C, best_gamma));
```

Next, we move to the Support vector classification part.

(ii) Support vector classification using C-SVC, the soft margin SVM and RBF Kernel

In this step, we chose to use LIBSVM library functions to perform C-SVC, i.e. soft-margin SVM. How we get the flexibility to choose that is explained below. An SVM classifies data by finding the best hyperplane that separates all data points of one class from those of the other class. The best hyperplane for an SVM means the one with the largest margin between the two classes. Margin means the maximal width of the slab parallel to the hyperplane that has no interior data points.

The support vectors are the data points that are closest to the separating hyperplane; these points are on the boundary of the slab.
When we call the 'svmtrain' function, we pass in the options along with it. These option give us flexibility to choose the kind of svm method, we want to use and the kind the kernel function, we want to use, for example:

```
options:
```

```
-s svm_type : set type of SVM (default 0)
        0 -- C-SVC
        1 -- nu-SVC
        2 -- one-class SVM
        3 -- epsilon-SVR
        4 -- nu-SVR
-t kernel_type : set type of kernel function (default 2)
        0 -- linear: u'*v
        1 -- polynomial: (gamma*u'*v + coef0)^degree
        2 -- radial basis function: exp(-gamma*|u-v|^2)
        3 -- sigmoid: tanh(gamma*u'*v + coef0)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default
0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 100)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking: whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates: whether to train a SVC or SVR model for probability
estimates, 0 or 1 (default 0)
-wi weight: set the parameter C of class i to weight*C, for C-SVC (default 1)
```

As we can see, that in the parameter passing of svmtrain, if we want to choose C-SVC, we either pass nothing, because by-default it considers C-SVC and for choosing the RBF kernel, we choose option 2, which is again a by-default option if user doesn't specify explicitly.

So, we train the model using 'svmtrain function' and using 'bestC' and 'bestgamma' in the following manner:

```
% using svmtrain to train the data
model = svmtrain(trainLabel, [(1:5000)' trainData*trainData'], sprintf('-c %f
-g %f -b 1', best_C, best_gamma));
```

With the above trained model, we use this model to classify out test data and record the output in

1. Predict Labels (the labels given to the test data based on the svm trained model.
2. Accuracy (the accuracy is calculated by comparing the predict_label and test_label.
3. Prob_values(these are the values of probability, a matrix of 2500*5, each row having 5 different probability values for each data, the one with the highest probability is assigned as it's class).

Following is how we perform classification using SVM model to classify the test data.

```
% Use the SVM model to classify the data
[predict_label, accuracy, prob_values] = svmpredict(testLabel, [(1:2500)'
testData*trainData'], cv, '-b 1'); % test the training data
```

Hence, above is the explanation of how grid search has been used to find the best parameters for training the svm model and using this trained model to classify data of testData (X_test.csv) and predict their classes.

1.b. Performing PCA to reduce to 2D

The next thing in this homework is to do dimensionality reduction and the method recommended is Principal component analysis (PCA), a technique used to emphasize variation and bring out strong patterns in a dataset. Its often used to make data easy to explore and visualize.

As in this case, we have taken our data i.e. of 2500*784 elements with each image having 784 different features, so plotting such high-dimensions is only possible in some other universe (:P) Here, to visualize our data we first transform the 784 different features to just 2 features that best represent these 784 features using Principle Component Analysis.

Now, MATLAB has this pre-defined function pca() that takes in various number of arguments depending upon how much information you want to extract i.e. how many dimensions do you want to project to as we are projecting the data from very-high dimension space to low-dimension space for visualization.

This is how we have done it in the code:

```
% PCA for dimentionalty reduction
[coeff, score] = pca(testData);
X = score(:,1:2);
```

FIGURE 1: TRAINING DATA POINTS PLOTTED IN 2D AFTER PCA

Now, as you can see here that as we call PCA function, we are getting two output parameters, or return parameters, those are the coeff and the score.
Now, coeff are the reduced coefficients should be easy to understand by the name. The second value is the 'score', now this score tells us the importance value as it sort of organizes our transformed data into decreasing order of variance. The ones with maximum variance are the first few vectors. So, from this we choose the first two columns and store it in 'X' for visualization.

Following is the plot for original data projection to 2D using PCA, and plotted based on the classes they belong to. This is labelled with the testLabel given in 'T_test.csv'.

1.c. Plotting the data

As it is important to visualize the data in order to fully understand it, we have plotted the data for:
1. Original testing data with 2D PCA
2. Predicted testing data (using predicted labels from C-SVC) with 2D PCA

The plot below is for the comparison of original data with original labels and for the predicted data with predicted labels using C-SVC and kernel function after training with SVMtrain and predicting labels with SVMpredict for test data.

1.Original Testing data with 2D PCA

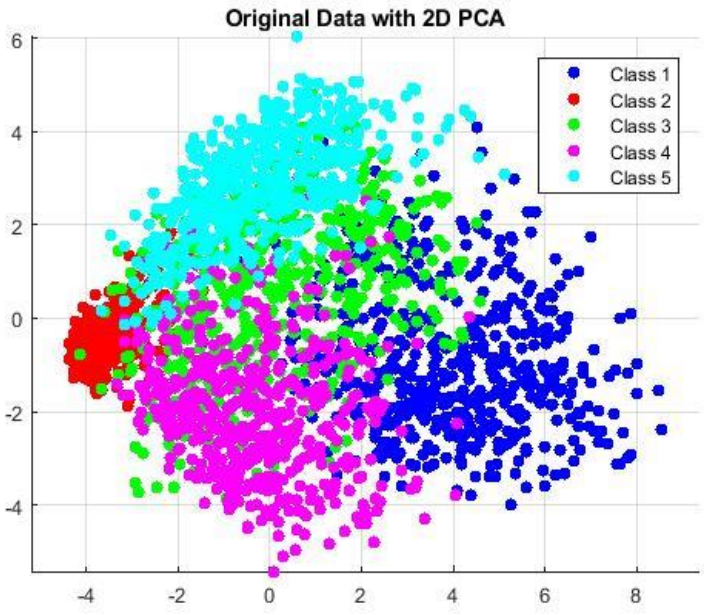

FIGURE 2: ORIGINAL TESTING DATA PLOTTED WITH GIVEN LABELS IN 2D AFTER PCA

Following is the code for how we are plotting the original test data with 2D PCA.

```
% Plotting the data before SVM, with given labels
[d,n] = size(X');
X = X';
testLabel = testLabel';
assert(n == length(testLabel));
color = 'brgmcyk';
m = length(color);
c = max(testLabel);

figure(1)
figure(gcf);
clf;
hold on;
view(2);
   for i = 1:c
       idc = testLabel==i;
       scatter(X(1,idc),X(2,idc),36,color(mod(i-1,m)+1), 'filled');
   end
       title('Original Data with 2D PCA');
       legend('Class 1', 'Class 2', 'Class 3','Class 4','Class 5');
%        decision(testData, testLabel);
axis equal
grid on
hold off
```

2.Predicted data (using predicted labels from C-SVC) with 2D PCA

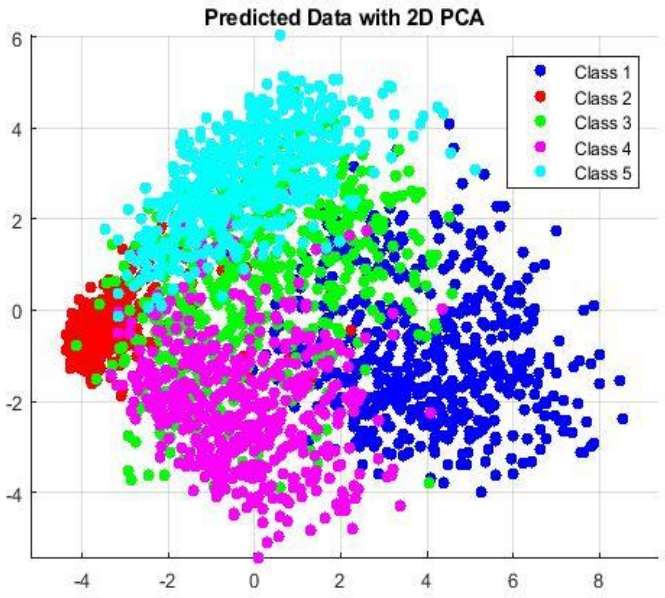

FIGURE 3: TESTING DATA WITH PREDICTED LABELS PLOTTED AFTER PCA IN 2D

The predict_label is the output from the above svmpredict function, that helps us in predicting the labels for the test data.

```
% Plotting the data after SVM, with predicted labels
predict_label = predict_label';
assert(n == length(predict_label));
c = max(predict_label);
figure(2)
figure(gcf);
clf;
hold on;
view(2);
   for i = 1:c
       idc = predict_label==i;
       scatter(X(1,idc),X(2,idc),36,color(mod(i-1,m)+1), 'filled');
   end
       title('Predicted Data with 2D PCA');
       legend('Class 1', 'Class 2', 'Class 3','Class 4','Class 5');
axis equal
grid on
hold off
```

As we can see, the predicted data with 2D PCA there are some data points that wrongly classified, as there is a red data marked right in the middle of green and blue data points, so using SVM gives us the accuracy of 96%. We will discuss more about its accuracy and its interpretation in the sections below.

1.d. Plotting decision boundary and support vectors

Further, we plot the decision boundary and the support vectors. For the sake of explanation, these are the vectors that define the hyperplane, so a lot depends on it, especially when you plot the decision boundary, you need to consider the support vectors to draw the best boundary separating your data. While, the decision boundary is surface that is maximally far away from any data point. This distance from the decision surface to the closest data point determines the margin of the classifier.

The geometric margin of the classifier is the maximum width of the band that can be drawn separating the support vectors of the two classes.

In all, we want to maximize the margin:

$$\underset{\mathbf{w},b}{\operatorname{argmax}}\{\min_{n}\frac{t_n(\mathbf{w}^\top\phi(\mathbf{x}_n)+b)}{\|\mathbf{w}\|}\}$$

Following the code for Decision boundary and Support vectors. LIBSVM provides us with the convinence of plotting support vectors easily by using the model.SVs function and model.sv_indices function for finsing the no. of support vectors and the index of the support vectors for each class.

```
%Support vectors
sv = full(model.SVs);
```

```
sv_idx = full(model.sv_indices);
figure(3)
figure(gcf);
clf;
hold on;

% Plot the training data along with the boundary
view(2);
   for i = 1:c
       idc = testLabel==i;
       scatter(X(1,idc),X(2,idc),36,color(mod(i-1,m)+1), 'filled');
       plot(sv(1,idc), sv(2,idc),36,color(mod(i-1,m)+1), 'black*');
   end
       title('Labelled Data with Support Vectors');
       legend('Class 1', 'Class 2', 'Class 3','Class 4','Class 5');
```

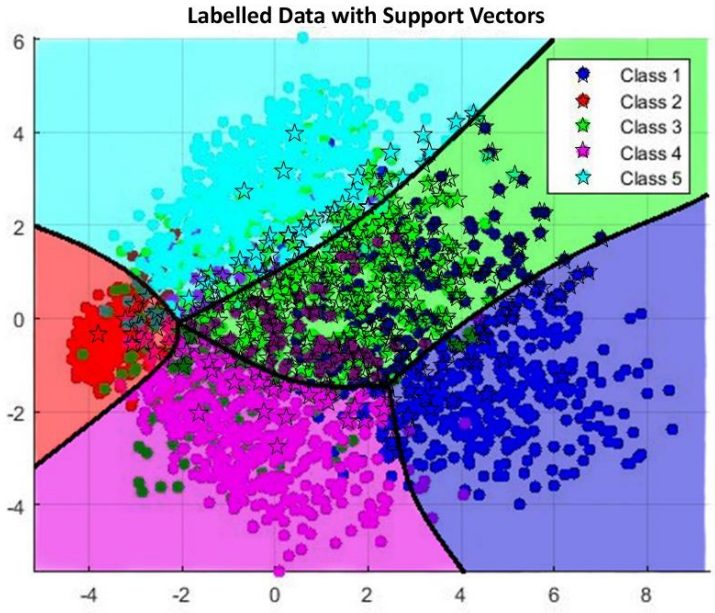

FIGURE 4: TESTING DATA WITH PREDICTED LABELS AND SUPPORT VECTORS MARKED WITH STARS

Following the code for Decision boundary. We make use of linspace() and meshgrid() function to make a grid of values for X and on that grid we use the contour function to plot the testing data and its labels along with the decision boundary over it.

```
figure; hold on

% Make classification predictions over a grid of values
xplot = linspace(min(features(:,1)), max(features(:,1)), 100)';
yplot = linspace(min(features(:,2)), max(features(:,2)), 100)';
[X, Y] = meshgrid(xplot, yplot);
vals = zeros(size(X));

for i = 1:size(X,2)
   x = [X(:,i),Y(:,i)];
   % Need to use evalc here to suppress LIBSVM accuracy printouts
```

```
   [T,predicted_labels, accuracy, decision_values] =
evalc('svmpredict(ones(size(x(:,1))), x, model)');
   clear T;
   vals(:,i) = decision_values;
end

% Plot the SVM boundary
colormap bone
if (size(varargin, 2) == 1) && (varargin{1} == 't')
    contourf(X,Y, vals, 50, 'LineStyle', 'none');
end
contour(X,Y, vals, [0 0], 'LineWidth', 2, 'k');
title('Labelled Data with Decision Boundary');
legend('Class 1', 'Class 2', 'Class 3','Class 4','Class 5');
```

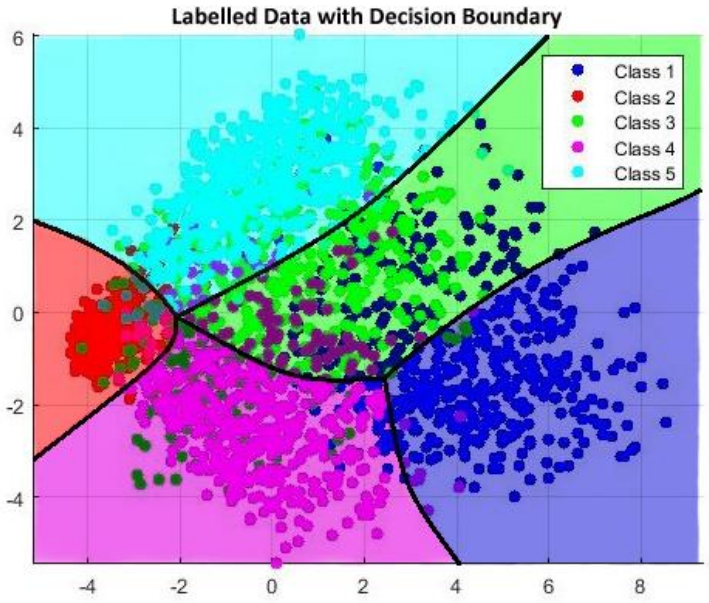

FIGURE 5: TESTING DATA WITH PREDICTED LABELS AND MARKED DECISION BOUNDARY AFTER SVM AND PCA

Above is the plot for decision boundary for multiclass SVM by creating a grid of values of X (the dimensionally reduced data after PCA) and 'contour' function for plotting the boundary and filling colors. Following is the plot for Kernel-SVM along with decision boundary and support vectors. In addition, there is the plot for predicted data for data points and the star marked points are the support vectors of each class. There are 701 support vectors are plotted on the predicted labelled diagram after SVM and PCA.

2. Testing Performance

During your cross validation, you will have both your original label and predicted label of your testing data in each fold. So in the end, the original label and predicted label in all folds will be concatenated. The overall accuracy is calculated as the number of correctly predicted label over the length of your original label. For the first case, where we are doing SVM and then PCA to

projecting data to 2D, for that case we are getting 2388 correct cases out of 2500, that gives us an accuracy of (~96%). Details below:

optimization finished, #iter = 1140
nu = 0.003083
obj = -3.083480, rho = 0.296820
nSV = 92, nBSV = 0
Total nSV = 701
Accuracy = 95.52% (2388/2500) (classification)

The performance is good when we do SVM and then PCA, approximately 96% and when we do PCA and then SVM, we find the accuracy to be 67%. So the difference is quite a lot and thus it is advisable to so SVM first and then PCA for both training and testing data.

The values of 'C' and 'Gamma' we have used here are computed using 5-fold cross validation as specified in the code:

```
% grid of parameters
folds = 5;
```

optimization finished, #iter = 800
nu = 1.000000
obj = -599.946801, rho = -0.001168
nSV = 701, nBSV = 701
Total nSV = 1995
Cross Validation Accuracy = 99.28%

If the testing data doesn't come with a label, there is no way we can evaluate the classification accuracy of our testing data. Hence, performance is largely affected by the selection of 'gamma' and 'c' and the choice of kernel function. The performance is the best with RBF kernel in most of the cases.

3. Inferences from Result

1. The ***gamma parameter*** defines how far the influence of a single training example reaches, with ***low values meaning 'far' and high values meaning 'close'*.** The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.
The ***C parameter*** trades off misclassification of training examples against simplicity of the decision surface. A ***low C*** makes the decision surface smooth, while a ***high C*** aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors.

2. Intuitively, the result for the accuracy of SVM and then PCA having ***better accuracy*** then doing PCA and then SVM is reasonable. As with training the model with less number of parameters, it makes sense to have less accuracy. As shown in the code below:

```
% PCA on Data for dimentionalty reduction
[coeff, score] = pca(testData);
X = score(:,1:2);
% Train the SVM
model = svmtrain(trainLabel, [(1:5000)' Xtrain*Xtrain'], '-c 0.7 -g 0.09 -b 1
-t 4');
% Use the SVM model to classify the data
[predict_label1, accuracy1, prob_values1] = svmpredict(testLabel, [(1:2500)'
X*Xtrain'], model, '-b 1'); % test the training data
```

The above code shows how we are performing PCA first and reducing the dimensions of the training and testing data, and then on the reduced data we are performing SVM. We are training with the projected data and thus finding labels for the testData after projection. So the accuracy by doing so, is much less than doing SVM first and then doing PCA.

4. Conclusion

1. A Support Vector Machine (SVM) performs classification by finding the hyperplane that maximizes the margin between the two classes. The vectors (cases) that define the hyperplane are the support vectors. C and Gamma are the parameters for a nonlinear support vector machine (SVM) with a Gaussian radial basis function kernel.

2. The idea of a "soft margin" SVM that allows some examples to be "ignored" or placed on the wrong side of the margin; this innovation often leads to a better overall fit. C is the parameter for the soft margin cost function, which controls the influence of each individual support vector; this process involves trading error penalty for stability.

3. SVM is intrinsically a linear separator when the classes are not linearly separable we can project the data into a high dimensionality space and with a high probability find a linear separation.

4. When data is not linearly separable the first choice is always a RBF kernel because they are very flexible. SVMs are effective when the number of features is quite large. It works effectively even if the number of features are greater than the number of samples.

5. Non-Linear data can also be classified using customized hyperplanes built by using kernel trick. It is a robust model to solve prediction problems since it maximizes margin.

6. The biggest limitation of Support Vector Machine is the choice of the kernel. The wrong choice of the kernel can lead to an increase in error percentage.

7. SVMs have good generalization performance but they can be extremely slow in the test phase. SVMs have high algorithmic complexity and extensive memory requirements.