**Machine Learning – HW 5**

Atieh Kashani

In this assignment a SVM classifier has been implemented to identify the gender of a crab from its physical measurements that are six physical features for each crab. We need to train a binary SVM model from the training samples and apply this model over testing samples to evaluate its performance.

The provided data set that is “crab.csv” contains 200 samples with gender labels (1: male and -1: female).

The following steps have been taken in this assignment:

Note: In order to make sure that all the model selection processes are performed with the exactly same training/validation data, I added the “data\_store.py” file that I used “pickle” module to store and load data. This file also contains the provided codes for step 1 : loading the dataset and getting feature matrix X and label vector Y, splitting 200 samples into 100 samples for training and validation and 100 samples for testing. Also, this file contains the code that I have implemented for step 2 for randomly dividing the training set that are 100 samples into two even subsets that 50 samples is used for training and another 50 samples is used for validation. This file “data\_store” runs one time and after running this script the training, validation and testing will be stored in the separate files that will be loaded in the “main\_svm” for SVM classifier implementation.

So, there are two python files “ data\_store” contains the related codes for loading, splitting data and storing data and ” and file “main\_svm” contains the provided codes and the sections that I have added for implementing the SVM classifier according to the following steps:

**Step-1 (data\_strore.py)** : Load data from the related file” crab.csv” and get feature matrix X and label vector Y. I added a file “data\_store\_load.py” that contains the provided codes for loading the dataset and getting feature matrix X and label vector Y. X is of 6 by 200 dimension, where each column represents a crab sample. The matrix Y is of 1 by 200 dimension, including the related gender labels (1 or -1). The provided codes randomly split these 200 samples into two even subsets: 100 samples for training and validation and another 100 samples for testing.

**Step-2 (data\_store.py)** : In the “data\_store.py” I have implemented the following code for randomly divided the training set that are 100 samples into two even subsets that one is used for training and another is used for validation. So, there are 50 samples for training and 50 samples for validation.

n2 = len(Xtr)  
S2 = np.random.permutation(n2)  
  
*# subsets for training models*x\_train = Xtr[S2[:50]]  
y\_train = Ytr[S2[:50]]  
*# subsets for validation*x\_validation = Xtr[S2[50:]]  
y\_validation = Ytr[S2[50:]]

After getting all the required data for training, testing, and validation, I have implemented the following code using pickle module to store the data into separate files.

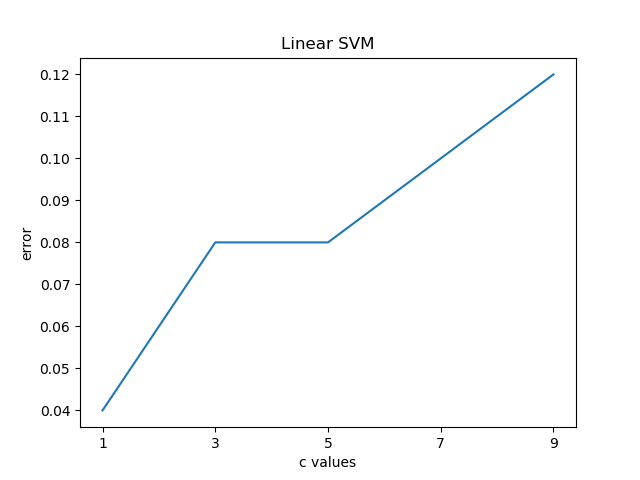
pickle.dump(x\_train, open(**"xTrain.dat"**, **"wb"**))  
pickle.dump(y\_train, open(**"yTrain.dat"**, **"wb"**))  
  
pickle.dump(x\_validation, open(**"xValidation.dat"**, **"wb"**))  
pickle.dump(y\_validation, open(**"yValidation.dat"**, **"wb"**))  
  
pickle.dump(X\_test, open(**"xTest.dat"**, **"wb"**))  
pickle.dump(Y\_test, open(**"yTest.dat"**, **"wb"**))

These files will be loaded in the “main\_svm” file in the step 3 to use the same training/validation/testing splitting while evaluating different models in the later steps.

x\_train = pickle.load(open(**"xTrain.dat"**, **"rb"**))  
y\_train = pickle.load(open(**"yTrain.dat"**, **"rb"**))  
  
x\_validation = pickle.load(open(**"xValidation.dat"**, **"rb"**))  
y\_validation = pickle.load(open(**"yValidation.dat"**, **"rb"**))  
  
  
x\_test = pickle.load(open(**"xTest.dat"**, **"rb"**))  
y\_test = pickle.load(open(**"yTest.dat"**, **"rb"**))

**Step – 3** : To train the SVM classifier first I consider different value for parameter C (the weighting parameters) with other hyper parameters fixed and train the model with training samples and apply the model over the validation samples. For each classifier I calculated and plot the validation errors . For parameters C, I consider different values in the range 1 to 10 with the step size 2 (the C value is 1,3,5,7,9) , while the kernel type is fixed (linear). The following plot represents the validation errors for different values of parameter C:

According to the following plot and results for the kernel type “Linear” C-value is best when it is ‘1’ as the validation error is least (0.040000000000000036)



C value: 1 , Validation error 0.040000000000000036

C value: 3 , Validation error 0.07999999999999996

C value: 5 , Validation error 0.07999999999999996

C value: 7 , Validation error 0.09999999999999998

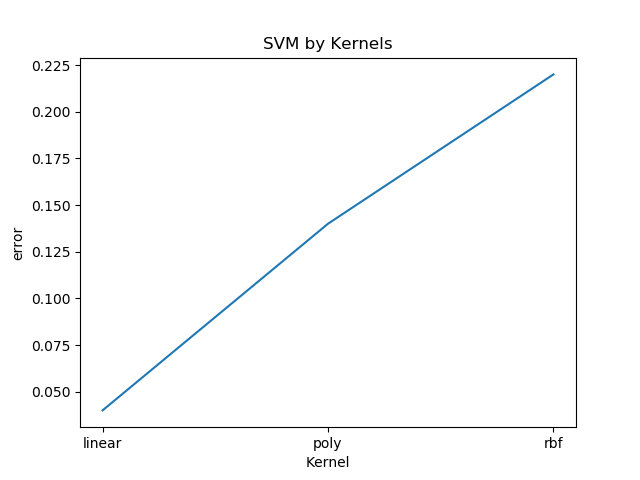
C value: 9 , Validation error 0.12

For the selection of kernels, I used different kernel type (Linear, RBF, and Polynomial) while other hyper parameters fixed for training the model with best C-value which is chosen from previous section using following code :

kernel\_types = [**'linear'**, **'poly'**, **'rbf'**]  
svm\_kernel\_error = []  
**for** kernel\_value **in** kernel\_types:  
 model = svm.SVC(kernel=kernel\_value, C=1)  
 model.fit(X=x\_train, y=y\_train)  
 error = 1. - model.score(x\_validation, y\_validation)  
 svm\_kernel\_error.append(error)

The following plot represents the validation errors for different kernel types:

The best Kernel type is Linear as the validation error is least.



Kernel type: linear , Validation error 0.040000000000000036

Kernel type: poly , Validation error 0.14

Kernel type: rbf , Validation error 0.21999999999999997

**Step - 4** : Based on the above results and plots the hyper parameters (C =1 and kernel type = linear) are the best. I have applied the best hyper parameters over the testing subsets.

**Step-5** : Evaluating the results using confusion matrix, average accuracy, Per-class Precision and Per-Class recall. Visualization of success and failure examples:

I evaluated the results using both the code that I have developed for confusion matrix, average accuracy, Per-class Precision and Per-Class recall for Homework 3 ( file “confusionMatrix.py” ) and also the provided code in Homework 5 for the confusion matrix and average accuracy, Per-class Precision and Per-Class recall. The results from both codes are the same:

**(Provided code in the homework folder for confusion matrix)**

Confusion Matrix:

[[47 5]

[ 0 48]]

Average Accuracy: 0.95

Per-Class Precision: [1. 0.90566038]

Per-Class Recall: [0.90384615 1. ]

**(My code from HW3 for confusion matrix)**

[[47, 5], [0, 48]]

Accuracy: 0.95

precision\_class\_1: 0.9056603773584906

precision\_class\_-1: 1.0

recall\_class\_1: 1.0

recall\_class\_-1: 0.9038461538461539

**According to the results:** the averageaccuracy is 0.95 for the hyper parameter values ( C=1 and Kernel type is “Linear”)

**Following are the visualization of both success and failure examples and the related codes:**

Failure Examples

Test samples: [[ 1. 9.1 8.1 18.5 21.6 7.7]

[ 1. 9.1 8.2 19.2 22.2 7.7]

[ 1. 10.8 9.5 22.5 26.3 9.1]

[ 1. 9.8 8.9 20.4 23.9 8.8]

[ 1. 7.2 6.5 14.7 17.1 6.1]]

Actual Label: [1. 1. 1. 1. 1.]

Predicted Label : [-1. -1. -1. -1. -1.]

Success Examples

Test samples: [[ 0. 20.6 17.5 41.5 46.2 19.2]

[ 0. 17.1 14.5 33.1 37.2 14.6]

[ 0. 18.9 16.7 36.3 41.7 15.3]

[ 0. 21.9 15.7 45.4 51. 21.1]

[ 0. 23. 16.8 47.2 52.1 21.5]]

Actual Label: [ 1. 1. 1. -1. -1.]

Predicted Label : [ 1. 1. 1. -1. -1.]

*## Visualize the success examples and failure examples***def** visualize\_failure\_result(xTest, true\_label,predict):  
 misclassified = np.logical\_not(np.equal(predict, true\_label))  
 misclassified\_crabs=xTest[misclassified]  
 misclassified\_crabs=misclassified\_crabs[0:5]  
 predicted\_label = predict[misclassified]  
 actual\_label = true\_label[misclassified]  
 print(**"Failure Examples"**)  
 print(**"Test samples: {} \n Actual Label: {} \n Predicted Label : {}\n"**.format(misclassified\_crabs, actual\_label[0:5],predicted\_label[0:5]))  
  
**def** visualize\_success\_result(xTest, true\_label,predict):  
 correct\_classified = (np.equal(predict, true\_label))  
 correct\_classified\_crabs=xTest[correct\_classified]  
 predicted\_label =predict[correct\_classified]  
 actual\_label = true\_label[correct\_classified]  
 print(**"Success Examples"**)  
 print(**"Test samples: {} \n Actual Label: {} \n Predicted Label : {}\n"**.format(correct\_classified\_crabs[0:5],actual\_label[0:5],predicted\_label[0:5]))