CPTS570 HW 2

Chandan Dhal

October 27, 2019

1 Programming Part

- **Question 1 a** Design linear SVM using Scikit.The Scikit learn package for linear SVM has no feature to compute support vectors.

  The training examples were split to Training and Validation set. A linear SVM was implemented on the Training set to observe the effect of the hyper-parameter C. The parameter C was varied accross 9 values described in the question. Three accuracy curve were computed for Training, Validation and Testing examples. Figure 1 depicts the accuracy curves as function of the parameter C.

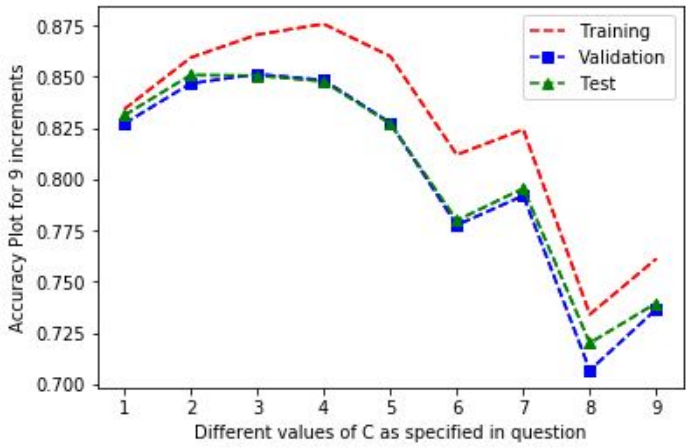

Figure 1: Linear SVM Accuracy Curve on Training, Validation and Test set.

Observations: Smaller value of parameter C, performs better on all of the accuracy curves. Based on the plot, the best value of parameter C is among the array of values specified in the question . The hyper-parameter is tuned to $C = 0.01$ for further analysis, although $C = 0.1$ is could perform very similar.

- **Question 1 b** Implement the linear SVM and compute the confusion matrix.

  The linear SVM was training on the complete example set, with parameter $C = 0.01$. The Scikit learn package was used to compute the confusion matrix based on the correct labels and predicted label. Three confusion matrix was computed below based on Training , Test and Validation set predictions.

Matrix.JPG

```
Training Confusion Matrix is:
 [[4519    7   18  102   18    1   96    0   39    1]
 [ 115 4566    8   73   10    0   13    0    2    0]
 [ 263   14 2882   56  874    1  618    0   60    0]
 [ 487   52    6 3999  180    0   65    0    8    0]
 [  79   10  215  160 3746    1  540    0   34    0]
 [  15    0    0    2    0 4677    3   87   12   39]
 [1366    9  308  124  457    0 2444    1  124    0]
 [   1    0    0    0    0  348    0 4386    5   73]
 [ 134   11   24  270   78  103   58   18 4091    1]
 [  21    1    0    2    0  335    2  199    2 4231]]
Validation Confusion Matrix is:
 [[1108    7    8   25    6    0   31    0   14    0]
 [  37 1137    1   25    3    0    7    0    3    0]
 [  73    5  712   16  239    0  173    0   14    0]
 [ 134   20    4  962   54    0   21    2    6    0]
 [  20    6   63   48  935    0  138    0    5    0]
 [   5    1    0    2    1 1080    3   33   14   26]
 [ 353   13   69   31  126    2  542    0   30    1]
 [   0    0    0    0    0  120    0 1040    2   25]
 [  32    6    8   76   37   43   14   11  980    5]
 [  13    1    1    0    0   85    1   61    1 1044]]
Test Confusion Matrix is:
 [[923   4   5  20   1   0  31   0  16   0]
 [ 33 945   4  11   0   1   5   1   0   0]
 [ 63   5 581  16 175   0 146   0  14   0]
 [104  25   4 809  39   0  15   2   2   0]
 [  8   2  42  33 793   0 117   0   5   0]
 [  6   2   0   1   0 909   2  49   9  22]
 [294   9  75  28  95   0 475   0  24   0]
 [  0   1   0   0   0  94   0 876   0  29]
 [ 25   2   7  55  17  34  27  11 818   4]
```

Figure 2: Confusion Matrix for $C = 0.01$

Observations: In lecture, we discussed that higher value across the diagonal means the classifier is not confused/wrongly predicts. It makes sense, since it implies that most predictions are correct. Based on the confusion matrix, the $C = 0.01$ outperforms $C = 0.1$. The test accuracy was computed in previous plot.

Matrix.JPG

```
Training Confusion Matrix is:
 [[2513    1    7   56    6    0 2199    0   19    0]
 [  22 4437    9   97    8    0  214    0    0    0]
 [  56    1 1343    6   90    0 3267    0    5    0]
 [  46   10    7 2958   20    0 1753    0    0    3]
 [   7    2  128   19  589    0 4036    0    4    0]
 [   8    1    0    0    0 4623   16  122   15   50]
 [ 143    3   49   20   26    0 4567    1   24    0]
 [   1    0    0    0    0  198    0 4544    4   66]
 [  35    2   34   37    3   49  814   19 3795    0]
 [   4    0    1    0    0  222   14  237    3 4312]]
Validation Confusion Matrix is:
 [[ 589    3    2   12    5    0  583    0    4    1]
 [   9 1119    4   32    4    0   44    0    1    0]
 [  25    1  305    2   25    0  874    0    0    0]
 [  14    5    6  703   12    0  454    2    1    6]
 [   1    1   37   12  146    0 1017    0    0    1]
 [   2    1    2    2    0 1063    8   45   11   31]
 [  52    2   20    8   11    1 1065    0    7    1]
 [   0    0    0    0    0   75    0 1085    2   25]
 [   8    5   10   15    2   25  186   13  943    5]
 [   3    1    2    0    0   53    9   75    2 1062]]
Test Confusion Matrix is:
 [[478   2   4  11   4   0 492   0   9   0]
 [  5 926   4  13   0   1  50   1   0   0]
 [  6   4 257   6  21   0 702   0   4   0]
 [ 15  12   5 603   9   0 350   3   0   3]
 [  2   0  36   3 121   0 838   0   0   0]
 [  9   3   2   0   0 904   2  47   8  25]
 [ 37   3  23   6  13   0 907   0  11   0]
 [  0   1   0   0   0  67   0 897   1  34]
 [  8   0   7  13   3  20 168  10 769   2]
 [  0   0   0   0   1  58   6  63   1 871]]
```

Figure 3: Confusion Matrix for $C = 0.1$

- **Question 1 c Implement polynomial Kernels SVM**

  A polynomial kernel was implemented on the Training set and the effect of the polynomial degree was analyzed. The accuracy curve was computed for the Training, Validation and Test set. Figure 4 depicts, accuracy as function of polynomial degree. Thus, a polynomial kernel of degree 2, is used for further analysis.

  Figure 5, depicts the number of support vectors for each polynomial kernels. Degree 2 has the least number of support vectors.

  Observations: The polynomial degree of 2 performs better than higher

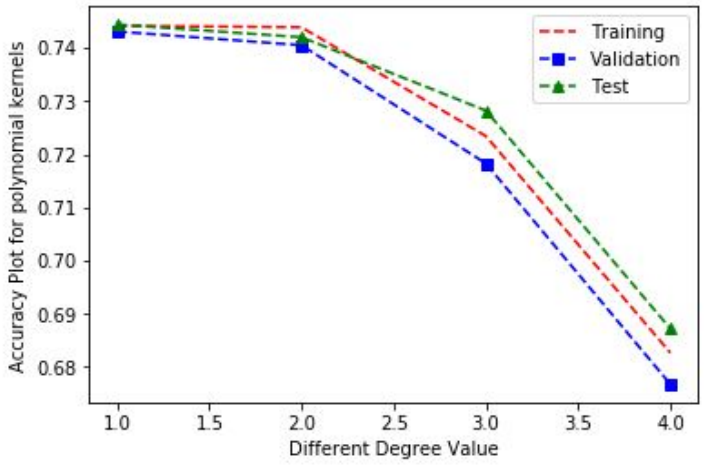

Figure 4: Accuracy curve for Polynomial kernel SVM

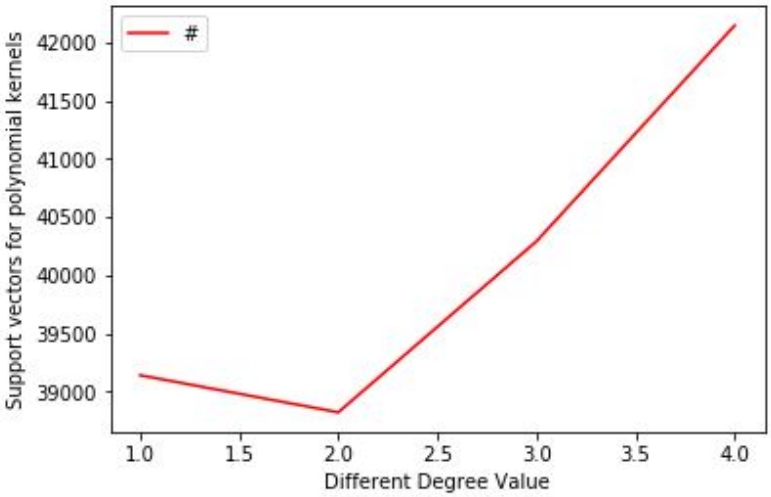

Figure 5: Support vectors for Polynomial kernel SVM

order degree. It can also be inferred from the plot that linear kernel is doing bit better in this Fashion example set.

- **Question 2: Multi-Class Kernelized Perceptron Classification**

  Remark 1: For further analysis, the data set entries are normalized to make the computation faster. For some reason, normalization had an effect on the plots for 1a and 1b.

A second order kernelized Perceptron was implemented on the Training examples. Similar analysis from homework 1 was performed. The Kernelized Perceptron was implemented for 5 Training iterations and mistakes over each iteration was computed. Accuracy curves of the Kernelized Perceptron was computed for training, validation and Test examples.

Remark 2: This question requires both large storage space and long execution time. The under-lining abstraction is similar to normal Perceptron Training set-up, but over Kernalized-Perceptron we are trying to over-parameterize the version-space. A Kernalized-Perceptron is estimating a large matrix **Alpha** of order k x $n(n = 48000$ Training examples, k classes), .

Remark 3: Due to the memory limitation, storing the large kernel matrix was not possible. Hence, the submitted code is computing the Kernal matrix for each Training example and the execution time is increased by $48000x$ (as computationally optimized as possible). If storage is not an issue, then the code can be changed to compute and store the Kernel matrix.

Figure 6, depicts the number of training mistakes for a second-order Kernalized Perceptron for 2 Training iterations.

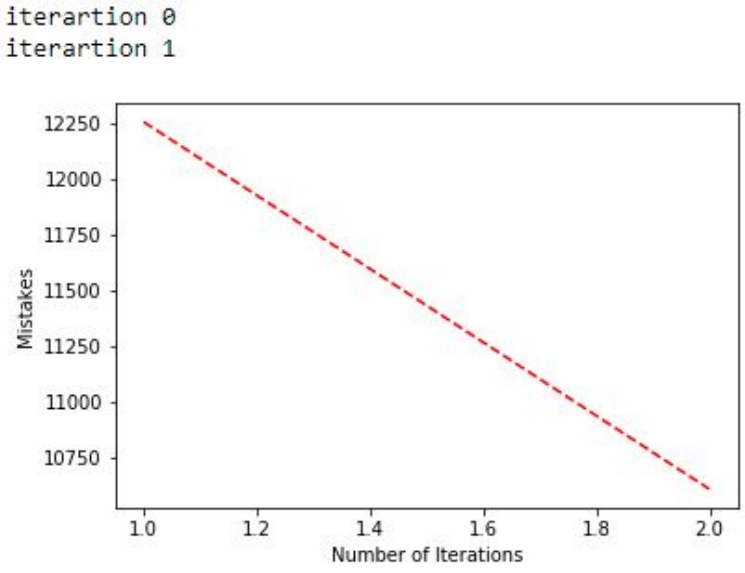

Figure 6: mistakes curve for second order kernelized Perceptron

Observations: From figure 4, the polynomial SVM accuracy plot [Training accuracy 75 %] and figure 6 mistakes curve of Kernelized-Perceptron

Training [Training accuracy min 75%]. It can be extrapolated that a second-order Kernelized-perceptron is performing better than second order SVM.

The code upon execution will generate both mistakes and accuracy plot for 5 Training iterations. The observations for this problem is based on the results of homework 1. As noted the abstract remaining same but the optimization version space is increased (from,kxd to kxn and here d<<n).

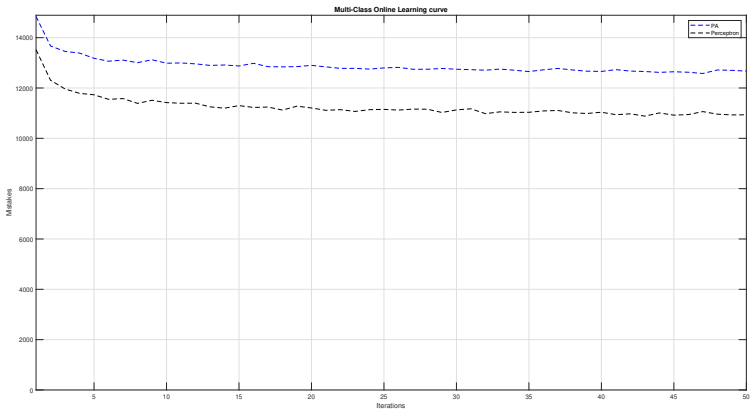

Figure 7: Mistakes curve for Perceptron from HW1

From figure 6 and figure 7, it can be inferred that similar observations as of perceptron set-up is expected. At each training iteration the mistakes are decreasing and the mistakes can be extrapolated to be close to normal Perceptron Training mistakes. Hence, similar argument holds for accuracy curve of different example set having similar characteristics of the Perceptron set-up.

Using kernels might be a very powerful tool on this example set (d<<n), and learning rate is constant.

- **Question 3a ID3 Decision tree**

  A decision tree based on the ID3 heuristic was implemented on Dataset. The pseudo-code depicts, the greedy heuristic for computation of Information gain.

  Psuedo-code for IG3 Tree

  - Initialize the first node with 490(70 % Training examples),v.
  - Initialize an array with all possible threshold value computed as described in question (length of array is k).

- Intialize an entropy matrix of dimensions d x k (d = # of features)
- For each impure node v, compute the conditional entropy of each of the possible thresholds and features (d x k matrix). Then find the minimum argument from entropy matrix. Since,the minimum conditional entropy results in maximum IG.
- Based on the minimum argument, split the impure node v with the respective feature and threshold condition. One of the sub-set is guaranteed to be a leaf node as the Information gain (entropy-reduction) due to conditional Expection is maximum when one sub-set is leaf node. This splitting is trying to achieve the best possible reduction of the uncertainty in an impure node.
- Repeat till all the nodes are Pure.

Remark 4: It was simpler to implement ID3 on Matlab with recursive function call. The submitted code needs $CPTS570HW2Q2_1.m$ saved before the ID3 program $CPTS570HW2Q2.m$ is executed. The function records all the boundary threshold values. The decision tree is based on these boundary values and the accuracy was computed based on these boundary conditions.

Observations: The Data set with missing rows removed is very small to notice over-fitting. The code will generate the decision boundary conditions for the complete tree. A test bench was programmed ***"manually"*** to check the efficiency of the ID3 decision trees. The implemented ID3 algorithm is classifying all the cases with label 4 as initial leaf node. At every impure node the algorithm splits one impure node and one leaf node of label 4. Thus, the algorithm produces 9 leaf nodes of label 4 and the last two leaf nodes as label 2. Figure 8 represents the decision tree based on maximum IG, the first column represents the feature element and the second column is the respective element in the sorted threshold array.

Figure 9 depicts the accuracy of the decision tree computed on Training, Test and Validation set with out pruning.

If any of the decision tree is merged with the lower impure branch, then the accuracy of all the example set decreases. Figure 10, depicts the drop in the accuracy when any decision branch are merged. Five randomly chosen decision layer were merged and the accuracy was computed for training, validation and testing set.

In pruning algorithm when last branch is dropped then the prediction accuracy is very low, since the tree can not identify the label 2s (Benign, majority examples). Other branches are classifying the label 4 based on the minimum conditional expectation. Hence if any other 9 branches are merged then the tree is mis-predicting few of the label 4 examples and accuracy drops on all example set.

The maximum testing accuracy (96.8 %) and validation accuracy (98.5 %) is achieved by the ID3 tree.

boundry

11x2 double

	1	2	3
1	0	0	
2	7	9	
3	7	8	
4	9	8	
5	9	7	
6	9	6	
7	1	7	
8	2	5	
9	8	3	
10	8	2	
11	1	1	

Figure 8: ID3 decision tree

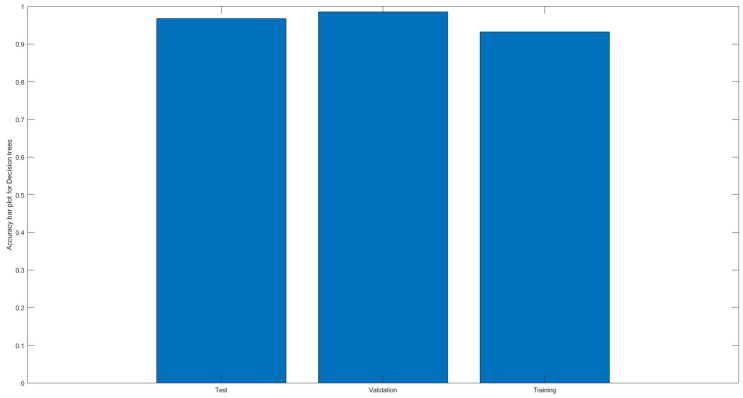

Figure 9: Accuracy of the decision tree

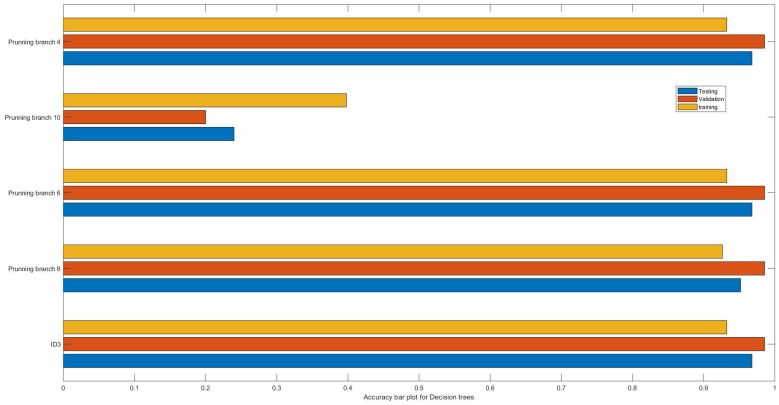

Figure 10: Effect of pruning on decision tree