CS5011: Programming Assignment #2

Girish Raguvir J, CS14B059, IIT Madras

October 2016

1 P1 - SVM

For this problem, I did feature extraction and used those features to train SVM Models with different kernels. These models were then used to classify various images. For this I used the SVC implementation of sklearn-libsvm which is built on top of the actual libsvm as it is well documented and more efficient.

I also come up with best model parameters for the different kernels by performing 5-fold cross validation on the train set.

Please look at p1.py for implementation.

1.1 Feature Extraction

Each pixel of an image is associated with a RGB triplet (r,g,b), indicating how much of each of the red, green, and blue is included. Each component varies from zero to 255. We separate the range into 32 bins for each of the 3 components. Now, for each pixel in image we bin the 3 components into one of the 32 bins of that color. Finally, this gives us 32 x 3 = 96 features for each image.

Please look at pre_process.py for implementation and further clarifications.

The datasets DS2_train.csv and DS2_test.csv can be found under the Dataset folder.

1.2 Linear Kernel - Results

Best fit parameters by varying C and evaluating performance on 5-fold cross validation:

C: 1e-09

1.2.1 Performance Metrics

Confusion Matrix:

$$\begin{vmatrix} 11 & 1 & 4 & 4 \\ 1 & 16 & 3 & 0 \\ 0 & 3 & 15 & 2 \\ 5 & 1 & 2 & 12 \end{vmatrix}$$

Accuracy: 67.5%

Class 0 (Coast)
Precision: 0.647
Recall: 0.55
F-measure: 0.595

Class 1 (Forest)
Precision: 0.762
Recall: 0.8
F-measure: 0.78

Class 2 (Inside City)
Precision: 0.625
Recall: 0.75
F-measure: 0.682

Class 3 (Mountain)
Precision: 0.67
Recall: 0.6
F-measure: 0.632

1.3 RBF Kernel - Results

Best fit parameters by varying C and evaluating performance on 5-fold cross validation:

C: 10
gamma: 1e-09

1.3.1 Performance Metrics

Confusion Matrix:

$$\begin{vmatrix} 10 & 0 & 3 & 7 \\ 0 & 18 & 2 & 0 \\ 0 & 3 & 17 & 0 \\ 9 & 1 & 3 & 7 \end{vmatrix}$$

Accuracy: 65%

Class 0 (Coast)
Precision: 0.526
Recall: 0.5
F-measure: 0.513

Class 1 (Forest)
Precision: 0.818
Recall: 0.9
F-measure: 0.857

Class 2 (Inside City)
Precision: 0.68
Recall: 0.85
F-measure: 0.755

Class 3 (Mountain)
Precision: 0.5
Recall: 0.35
F-measure: 0.412

1.4 Sigmoid Kernel - Results

Best fit parameters by varying C and evaluating performance on 5-fold cross validation:

C: 0.1
gamma: 1e-09

1.4.1 Performance Metrics

Confusion Matrix:

$$\begin{vmatrix} 7 & 4 & 0 & 9 \\ 1 & 19 & 0 & 0 \\ 4 & 7 & 0 & 9 \\ 3 & 1 & 0 & 16 \end{vmatrix}$$

Accuracy: 52.5%

Class 0 (Coast)
Precision: 0.467
Recall: 0.35
F-measure: 0.4

Class 1 (Forest)
Precision: 0.613
Recall: 0.95
F-measure: 0.745

Class 2 (Inside City)
Precision: nan
Recall: 0
F-measure: nan

Class 3 (Mountain)
Precision: 0.47
Recall: 0.8
F-measure: 0.593

1.5 Polynomial Kernel - Results

Best fit parameters by varying C and evaluating performance on 5-fold cross validation:

C: 0.0001
degree: 3
gamma: 0.0001

1.5.1 Performance Metrics

Confusion Matrix:

$$\begin{vmatrix} 14 & 2 & 2 & 2 \\ 1 & 19 & 0 & 0 \\ 0 & 6 & 14 & 0 \\ 8 & 0 & 3 & 9 \end{vmatrix}$$

Accuracy: 70%

Class 0 (Coast)
Precision: 0.61
Recall: 0.7
F-measure: 0.65

Class 1 (Forest)
Precision: 0.7
Recall: 0.95
F-measure: 0.81

Class 2 (Inside City)
Precision: 0.74
Recall: 0.7
F-measure: 0.72

Class 3 (Mountain)
Precision: 0.82
Recall: 0.45

F-measure: 0.581

2 P2 - Neural Networks

2.1 Introduction

Layers are numbered as 1,2 and 3.Layer1 is input layer, Layer2 in hidden layer and Layer3 is output layer.Layer1's output is thresholded by sigmoid to get Layer2 , Layer2's output is passed through softmax to get Layer3 and Layer3 is finally passed through argmax to predict the class.

$\delta^{(l)}$ - error at layer l

$Z^{(l)}$ - output of layer l

a^{l} - thresholded output by layer l

$L(\theta)$ - loss function

C^{l} - bias unit as layer l

2.2 Forward Propagation

$$Z^{(1)} = X \cdot \theta^{(1)} + C^{(1)}$$
$$a^{(1)} = sigmoid(Z^{(1)})$$
$$Z^{(2)} = a^{(1)} \cdot \theta^{(2)} + C^{(2)}$$
$$a^{(2)} = softmax(Z^{(1)})$$

2.3 Back Propagation

2.3.1 Gradients

$$\delta_i^{(3)} = a_i^{(2)} .* [(a_i^{(2)} - y_i) + \sum_{i=0}^{3}(a_i^{(2)} y_i) - \sum_{i=0}^{3}(a_i^{(2)})^2]$$

$$\frac{\partial}{\partial \theta^{(2)}}\big(L(\theta)\big) = a^{(1)T} \cdot \delta^{(3)} + \gamma\theta^{(2)}$$

$$\frac{\partial}{\partial C^{(2)}}\big(L(\theta)\big) = \sum_i \delta_i^{(3)}$$

$$\delta^{(2)} = (\delta^{(3)} \cdot \theta^{(2)T}) .* a^{(1)} .* (1 - a^{(1)})$$

$$\frac{\partial}{\partial \theta^{(2)}}\big(L(\theta)\big) = X^T \cdot \delta^{(2)} + \gamma\theta^{(1)}$$

$$\frac{\partial}{\partial C^{(1)}}\big(L(\theta)\big) = \sum_i \delta_i^{(2)}$$

2.3.2 Update Rules

$$\theta^{(1)} = \theta^{(1)} - \alpha \frac{\partial}{\partial \theta^{(1)}}\big(L(\theta)\big)$$

$$C^{(1)} = C^{(1)} - \alpha \frac{\partial}{\partial C^{(1)}}\big(L(\theta)\big)$$

$$\theta^{(2)} = \theta^{(2)} - \alpha \frac{\partial}{\partial \theta^{(2)}}\big(L(\theta)\big)$$

$$C^{(2)} = C^{(2)} - \alpha \frac{\partial}{\partial C^{(2)}}\big(L(\theta)\big)$$

3 Best Performance Metrics

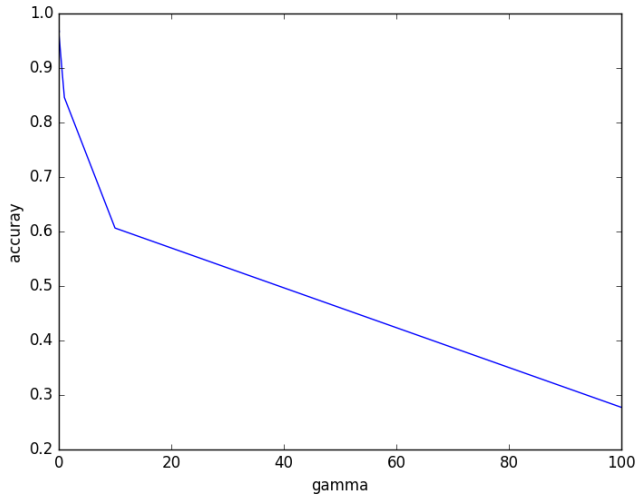

Figure 1: Accuracy vs Gamma for train data

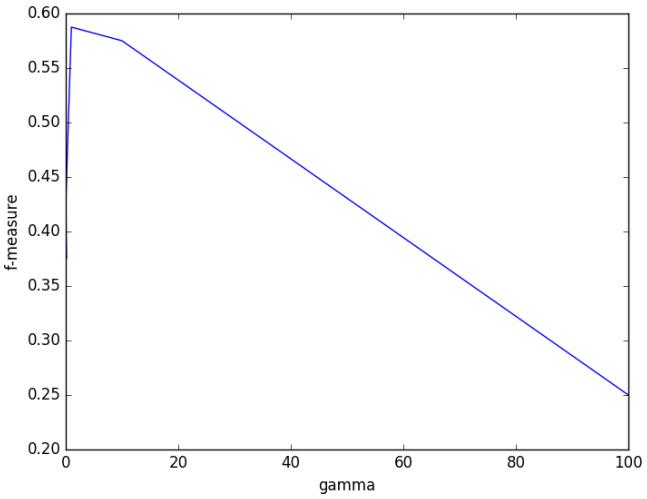

Figure 2: Accuracy vs Gamma for test data

Confusion Matrix:

$$\begin{vmatrix} 14 & 1 & 2 & 3 \\ 4 & 14 & 2 & 0 \\ 3 & 1 & 13 & 3 \\ 11 & 2 & 0 & 7 \end{vmatrix}$$

Accuracy: 60%

Class 0 (Coast)
Precision: 0.44
Recall: 0.7
F-measure: 0.54

Class 1 (Forest)
Precision: 0.78
Recall: 0.7
F-measure: 0.74

Class 2 (Inside City)
Precision: 0.76
Recall: 0.65
F-measure: 0.7

Class 3 (Mountain)
Precision: 0.54
Recall: 0.35
F-measure: 0.43

4 Implementation

Please look at p2.py for the implementation of above and further clarifications.

5 Conclusion

As gamma is increased we are increasing the panalty for large weights in between the layers. Increasing gamma may reduce accuracy on the train set but the trained model would generalise well.

As gamma increases, we observe that the weights learnt are smaller in magnitude due to larger penalty.

6 P3 - Decision Tree

6.1 Datasets

I split the given Mushroom dataset into train (7000 instances) and test (1124 instances) and converted each to ARFF format. You can find the datasets agaricus-lepiota_test.data.arff and agaricus-lepiota_train.data.arff under the Datasets folder.

6.2 Multiway Splits

6.2.1 Performance Metrics on Test Set - No pruning

Confusion Matrix:

$$\begin{vmatrix} 660 & 0 \\ 0 & 464 \end{vmatrix}$$

Accuracy: 100%

Class 0 (Poisonous)
Precision: 1.0
Recall: 1.0
F-measure: 1.0

Class 1 (Edible)
Precision: 1.0
Recall: 1.0
F-measure: 1.0

6.2.2 With Reduced Error Pruning

Confusion Matrix:

$$\begin{vmatrix} 660 & 0 \\ 0 & 464 \end{vmatrix}$$

Accuracy: 100%

Class 0 (Poisonous)
Precision: 1.0
Recall: 1.0
F-measure: 1.0

Class 1 (Edible)
Precision: 1.0
Recall: 1.0
F-measure: 1.0

minNumObj	Accuracy
1	100%
4	100%
8	100%
32	99.3722%
128	99.3722%
512	99.3722%
1024	99.3722%

Table 1: minNumObj vs Accuracy

As we can see, reduced error pruning in case of multiway splits does not seem to make a difference.But however as we increase minNumObj the accuracy, though not significantly, decreases.

6.3 Binary Splits

6.3.1 Performance Metrics on Test Set - No pruning

Confusion Matrix:

$$\begin{vmatrix} 660 & 0 \\ 8 & 456 \end{vmatrix}$$

Accuracy: 99.2883%

Class 0 (Poisonous)
Precision: 0.988
Recall: 1.0
F-measure: 0.994

Class 1 (Edible)
Precision: 1.0
Recall: 0.983
F-measure: 0.991

6.3.2 With Reduced Error Pruning

Confusion Matrix:

$$\begin{vmatrix} 660 & 0 \\ 13 & 451 \end{vmatrix}$$

Accuracy: 98.8434%

Class 0 (Poisonous)
Precision: 0.981
Recall: 1.0
F-measure: 0.990

Class 1 (Edible)
Precision: 1.0
Recall: 0.972
F-measure: 0.986

minNumObj	Accuracy
1	99.2883%
4	99.3772%
8	98.8434%
32	99.3772%
128	99.3722%
512	99.3722%
1024	99.3722%

Table 2: minNumObj vs Accuracy

As we can see, reduced error pruning in case of binary splits, though not by much, worsens performance. But however as we increase minNumObj the accuracy does not seems to change by much.

6.4 Conclusion

As you increase minNumObj, you are basically increasing the number of instances at the node before splitting stops. Thus as you increase minNumObj, you are increasing the number of instances that are in the leaf in the final decision tree and effectively decreasing the number of leaf nodes in the tree. Thus the decision would be forced to put dissimilar points together in one region when minNumObj is too high. But in general, increasing minNumObj may decrease accuracy on train set but it is expected to train a model which generalizes better. We may get a decision tree that might perform worse on the training data but generalization is the goal.

One of the simplest forms of pruning is reduced error pruning. Starting at the leaves, each node is replaced with its most popular class. If the prediction accuracy is not affected then the change is kept. In general, this is expected to train a model which generalizes better.

By observing the decision tree generated, we can say the features odor, spore-print-color, stalk shape, gill-size, gill-shape and population are some important features as they are used high up in the decision tree to split.