Aslı Aykan
2016400222

CMPE 462 | Assignment 3

PART 1

The first part of the assignment in about decision tree, and it has two steps. In the first step, we should implement a decision tree with information gain and apply on test data. In the second step, we should implement a decision tree with gain ratio and apply on the test data.

Code Summary:

I created a while loop to create every sub-nodes of a node. For each data set, I found the split feature by calculating necessary entropies and using the information gain or gain ratio formula. After splitting, two data lists are added to the iteration list for next node calculations. To select a threshold, I used the mean values of each feature. However, one feature can be used more than once, so I recalculated the mean values for each splitting node to be able to select features with different thresholds.

To save the decision tree and use it while testing with test data, I saved each node as a list in a list of nodes.

A node structure in my implementation is follows:

['0', '0', 'petal-length', 'S', 'V', 2.8900000000000006]

0) 0 is the id of the node.

1) 0 is the level of the node.

2) petal length is the splitting feature of that node.

3) 'S' is the left-leaf value of that node. ('S' means iris-setosa, entropy = 0)

4) 'V' is the right-leaf value of that node. ('V' means iris-versicolor, entropy=0)

5) 2.8900000000000006 is the threshold value for that feature.

For leaf values, 'Cont.' is used if the leaf should be also a feature (entropy>0)

I also created a list called *fromIds* to save the id's of each parent node of a node.

As explained, each node is saved in a list and that list is added to the *nodesList*. In the testing part, each sample in the input data is tested by using this *nodesList* and *fromIds.*

STEP 1

In step 1, we should implement a decision tree with information gain. I used the formula that we learned in class while calculating entropy and information gain. For each split node, I calculated its initial entropy, found the best feature that separates the data by subtracting each

entropy for the features from the initial entropy and selecting the highest value. After this step, I separated the data and added them as two data lists to the iteration list for next calculations.

The output is as follows:

DT petal-length 1.0

STEP 2

In step 1, we should implement a decision tree with gain ratio. I used the formula that we learned in class while calculating entropy, information gain and gain ratio. For each split node, I calculated its initial entropy, found the best feature that separates the data by subtracting each entropy for the features from the initial entropy, dividing them to the corresponding denominator of gain ratio formula and selecting the highest gain ratio. After this step, I separated the data and added them as two data lists to the iteration list for next calculations.

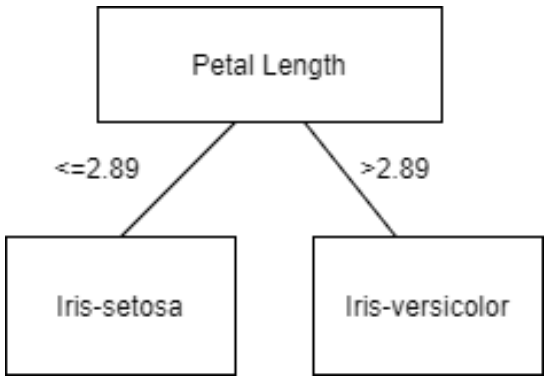
The output is as follows:

DT petal-length 1.0

As can be seen, the results for step 1 and step 2 are the same for the given data set. The decision trees are also the same.

Max depth is 6 levels. However, the decision tree does not need to have 6 levels, so there is no need to imply any pruning algorithm.

Here is the decision tree:



PART 2

The second part of the assignment is about support vector machines, and it has two steps. In the first step, we should SVM with 5 different C values. In the second step, we should apply SVM with different kernel types.

Aslı Aykan
2016400222

Before implementing the SVM on test data, I normalized the train and test data to not to get any warnings about the maximum number of iterations.

Additionally, I removed the id column and I also removed the last column on the given data set, which has “nan” values.

STEP 1

In step 1, the accuracy values of different C values should be examined for a fixed kernel type. To choose a fixed kernel type, I chose the type which has the highest accuracy value for the input data set. I compared these accuracies in step 2 and linear type gave the highest accuracy score. So, I decided to use linear kernel type in step 1.

My 5 different C values are 0.01, 0.1, 1, 10 and 100.

Here are the results:

0.01 -> ***acc***=0.7692307692307694 ***n***=346

0.1 -> ***acc***=0.7751479289940828 ***n***=346

1-> ***acc***=0.911242603550296 ***n***=244

10-> ***acc***=0.9289940828402367 ***n***=94

100 -> ***acc***=0.9349112426035504 ***n***=78

As we learned in the class, margin is the distance from a point to the hyperplane that we decide to use to separate our data set. The smaller c value means the larger margin. As can be guessed, it is difficult to have a large margin with completely separated data, so the correctness decreases as margin increases. (The data points in different classes can be very close, so a model with larger margin can fail to classify it correctly) So, it can be thought as a measure of correctness. Higher c value means lower margin, and a lower margin yields more accurate results. As can be seen from results, larger c value increases the accuracy score.

However, to prevent overfitting, we should choose our C value wisely so that we get a smooth hyperplane. In order words, if we choose a too high value for C, our model memorizes the training data set, which causes overfitting and reduces the accuracy score.

In this example, we add a penalty term (c value), so this is an example of soft-margin case. In soft-margin case, the support vectors are the ones which are inside the margin and on the margin. So, in larger margins (or lower c values), the area is larger, so there exist more support vectors, as can be seen from the output.

Aslı Aykan
2016400222

STEP 2

In step 2, the accuracy values of different kernel types should be examined for a fixed C value. I picked the c value that gave the highest accuracy score from the ones which I used in step 1. So, c=100.

The different kernel types that I used are linear, rbf(radial basis function), sigmoid and polynomial.

Here are the results:

Linear -> ***acc***=0.9349112426035504 ***n***=94

Rbf -> ***acc***=0.9289940828402367 ***n***=155

Sigmoid -> ***acc***=0.911242603550296 ***n***=181

Polynomial -> ***acc***=0.7692307692307694 ***n***=346

In svm models, kernel is used to map the input data to higher dimensional space to get more accurate predictions from the model. (the data set can be separated linearly of nonlinearly)

The polynomial kernel type is less popular than other kernel types because of its less accuracy scores. It collects some of the polynomial combinations of the features to create the model and is generally used in visual pattern recognition.

The rbf(radial basis function) is a widely used svm kernel type. (also the default type) It gives high accuracy scores in nonlinear data sets. It uses the Gaussian distribution for creating space for data.

The linear kernel type is very efficient when the data set can be separated linearly. It is a parametric model and simpler type than rbf and polynomial type.

The sigmoid kernel type is efficient in neural networks. SVM with sigmoid kernel is actually 2-layer perceptron.

There is no “best kernel type” that perfectly fits every input data. To determine the best type for a given data set, different kernel types should be tested, and their accuracy scores should be compared.

From the output, we can see that the number of vectors usually increases as the complexity of the kernel type increases. However, it is not always the case. With different c values, the ordering according to the number of vectors changes. For example, for c=10000000, the number of vectors in rbf is higher than the number of sigmoid.

Since I got the best accuracy score in linear kernel type, we can say that the data is linearly separable.