University of Nevada, Reno

Department of Computer Science and Engineering

CS 691: Machine Learning

Project 1

Cayler Miley

Instructor: Emily Hand

September 26, 2018

1 Decision Trees

1.1 Overview

The Decision Trees implementation is a collection of modular function built on a recursive algorithm. The tree is built as a dictionary of values, with accuracies at each level. If the current branch of the tree is not a leaf, then the dictionary contains two sub trees split on the most accurate feature. The dictionary structure is as follows for leaves:

```
node = {
    "accuracy": accuracy,
    "output": label
}
```

The dictionary structure is as follows for non-leaves:

```
node = {
        "accuracy": best_acc,
        "feature": feature_index,
        "threshold": best_threshold,
        (feature_index, 0): lower_tree_root,
        (feature_index, 1): upper_tree_root
    }
```

The implementation was chosen to include a threshold to allow the function stack to operate on both real values (continuous points) and discrete classification. The threshold is set to 0.5 for the discrete case. When ties occur, they are broken based on $num \leq threshold$. The most important function which recurses to build the tree is: `choose_best_tree(X, Y, feature_indices, current_depth, max_depth, is_binary=True)`. `choose_best_tree` takes in features, labels, a list of the features used up to the current depth, the current depth, the max depth, and whether the tree is binary (in terms of features). It then creates as single depth trees as possible for the remaining features and recurses on a subset of X and Y, with the recently used feature is split on appended to `feature_indices`. The best tree is chosen based on accuracy. If the labels are the same, the features are the same, or the current depth is as max depth, the recursion stops. Note that in the tree implementation, the accuracy is only the accuracy on the training level of that depth 1 tree, for debugging purposes. There are some helper functions to abstract different components and allow the required functions to be standalone. Note that the tuples serve as which feature we break on and correspond to the tree below the threshold (0) and above the threshold (1).

1.2 Implementation Details

1.2.1 `DT_train_binary(X,Y,max_depth,FLAG_print=True)`

This function sets the current depth to be 0 and the used feature set to be empty. Then it makes the initial call to `choose_best_tree`, which recurses and returns the full tree. There is also a print flag included to see the tree output.

1.2.2 `DT_test_binary(X,Y,DT)`

This function initializes the number correct from the test samples as zero and then uses the tree to make a prediction on the label value. If it is correct, the number correct is incremented. After all samples have been tested, the accuracy returned is simply $\frac{num\_correct}{total\_samples}$.

1.2.3 `DT_train_binary_best(X_train,Y_train,X_val,Y_val)`

This function sets the max depth to be $\arg\min(f, s-1)$ where f is the feature count and s is the sample count. Then the function builds a tree for each possible depth and computes the accuracy on the validation set. The tree that is returned is the tree that had the maximum accuracy.

1.2.4 `DT_train_real(X,Y,max_depth)`

This function trains a tree in the same way as the binary training, but instead tests all possible thresholds and returns the depth 1 tree with the highest accuracy based on all of the thresholds. It simply loops over the possible thresholds as well as the available features.

1.2.5 `DT_test_real(X,Y,DT)`

This function tests a real tree using its thresholds in the same way as the binary testing function. The accuracy on all of the samples is returned.

1.2.6 `DT_train_real_best(X_train, Y_train, X_val, Y_val)`

This function computes all possible trees on the training set and returns the tree with the highest accuracy on the validation set. This function works identically to the binary version, except with real values.

1.3 Performance Evaluation

The tree output for the first testing set is (with max depth):

```
tree = {
    'accuracy': 1.0,
    'feature': 1,
    'threshold': 0.5,
    (1, 0): {
        'accuracy': 1.0,
        'output': 0
    },
    (1, 1): {
        'accuracy': 1.0,
        'output': 1
    }
}
```

The accuracy value of this tree on the test set is 0.75. The best possible tree happens to be the same as the one output by the initial function (i.e. the tree above). The accuracy is also 0.75.

The tree output for the second set is (with max depth):

```
tree = {
    'accuracy': 0.7777777777777778,
    'feature': 3,
    'threshold': 0.5,
    (3, 0): {
        'accuracy': 0.75,
        'output': 0
    },
    (3, 1): {
        'accuracy': 0.8,
        'output': 1
    }
```

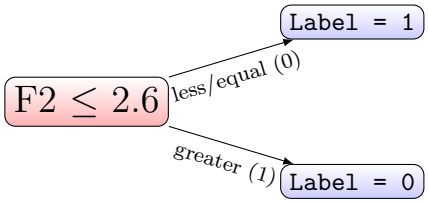
```
}
```

The accuracy value of this tree is 0.66666666. The best tree is shown below:

```
tree = {
    'accuracy': 0.7777777777777778,
    'feature': 3,
    'threshold': 0.5,
    (3, 0): {
        'accuracy': 0.75,
        'feature': 0,
        'threshold': 0.5,
        (0, 0): {
            'accuracy': 0.5,
            'output': 0
        },
        (0, 1): {
        'accuracy': 1.0,
        'output': 0
        }
    },
    (3, 1): {
        'accuracy': 1.0,
        'feature': 2,
        'threshold': 0.5,
        (2, 0): {
            'accuracy': 1.0,
            'output': 1
        },
        (2, 1): {
            'accuracy': 1.0,
            'output': 0
        }
    }
}
```

The accuracy for this tree is 0.7777777778.

1.3.1 Graduate Requirements



A depiction of the output tree for real values is shown.

2 K-NN Classifier

2.1 Overview

The K-NN classifier computes the L-2 norm to the closest K points and returns the most common label among the nearest neighbors.

2.2 Implementation Details

2.2.1 `KNN_test(X_train,Y_train,X_test, Y_test,K)`

The KNN test function implementation simply generates a label based on the test point using the `test_point(X_train, Y_train, point, K)` function. This function computes the distances (as the L_2 norm) to each training point and appends them to a list. The list is then sorted based on distance. Afterwards, the subset of the K closest points are chosen with their labels summed. The sign of the sum is output as the test point label.

2.2.2 `choose_K(X_train,Y_train,X_val, Y_val)`

The choose K function tests all possible K-NN accuracies on the validation set and returns the K with the greatest accuracy. This is implemented simply as a max of the accuracy output by the `KNN_test`.

2.3 Performance Evaluation

For the testing set, KNN achieves the following accuracies.

K	Accuracy
1	0.3
3	0.4
5	0.6

The best K for the test set however is computed to be 9.

3 Clustering

3.1 Overview

The K-means implementation initializes cluster centers at random sample points and recomputes cluster centers until convergence.

3.2 Implementation Details

3.2.1 `K_Means(X, K)`

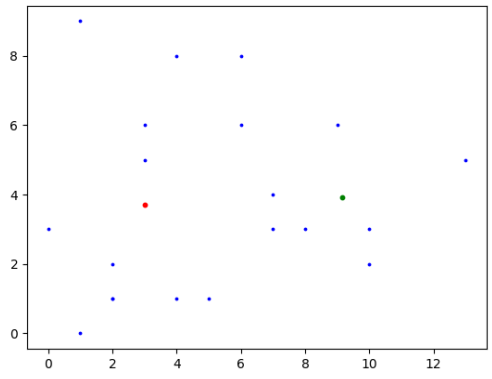
Cluster centers are initialized at random unique samples. If K is greater than the number of samples, `None` is returned. While convergence is not true, we compute clusters and then update centroids. Convergence returns true if two sets of clusters (stale and updated) are equivalent (meaning the centroids won't update).

3.2.2 `K_Means_better(X, K)`

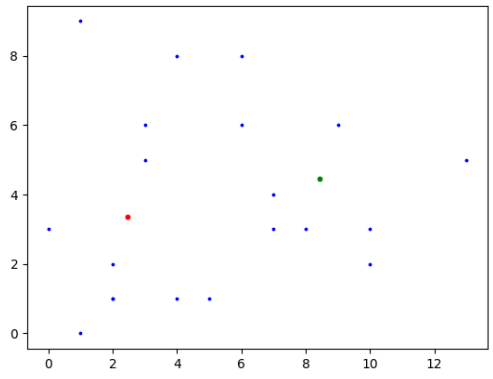
The `K_Means_better(X,K)` function runs the first K-Means implementation until the same cluster centers have occurred more than half of the time or 1000 trials have occurred. It also requires that a minimum of 10 trials occur. Initially Root-Mean-Square-Error was used to determine statistical convergence, but several data samples resulted in non-convergence. Thus the stopping condition is either maximum trials or a single set of centroids occurs after convergence for more than half of the trials.

3.3 Performance Evaluation

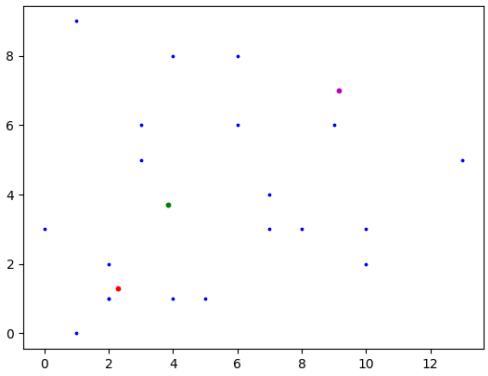
The plots show the cluster centers (in red, green, and magenta) with the data points (in blue).



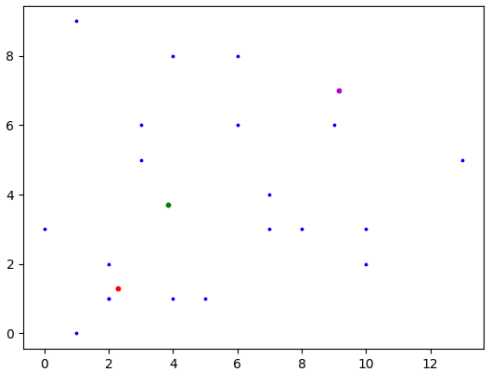
(a) K = 2



(b) K = 2 - better



(c) K = 3



(d) K = 3 - better

Figure 1: The cluster centers for the data as given by K-Means and K-Means_better respectively.