CSC 466 Lab 3 Report

Authors:
Sophia Chung, spchung@calpoly.edu
Anagha Sikha, arsikha@calpoly.edu

Introduction

In this report, we will go over our implementations of the C4.5, Random Forest, and KNN classifiers. For each method, we found the best hyperparameter values and calculated the accuracies for each of our three chosen datasets.

Implementations and Evaluation Procedures

C4.5

C4.5 is a recursive decision tree induction algorithm. Our implementation in InduceC45.py involves the creation of the DecisionTreeClassifier class, which contains the following functions: entropy, findBestSplit, selectSplittingAttribute, and C45. The file takes as input a formatted data file and an optional "restrictions" file. C45 follows three main steps. It first checks two termination conditions; if our dataset contains records with the same class label or there are no more attributes to consider, then we create a leaf node. Second, it selects the splitting attribute, whether categorical or continuous, using information gain. Third, it constructs the tree.

The evaluation of C4.5 is done in classify.py. The file takes as input a formatted data file and a .json file containing a decision tree output from InduceC45.py. The file has a function named classify that takes a row of data and classifies it based on the given decision tree. We keep track of the number of correct and incorrect classifications to report accuracy and error rate, and construct an overall confusion matrix using crosstab.

Random Forest

Random Forest is an ensemble learning method that builds a collection of decision trees using bagging or bootstrap aggregating. Our implementation takes as input a formatted data file, the number of attributes each decision tree built by the random forest classifier should contain (m, selected without replacement), the number of data points selected randomly with replacement (k), and the number of decision trees to build (N). Our randomForest.py file creates the RFClassifier class, which contains the following functions: data_selection, random_forest, and cross_validation.

We use 10-fold cross-validation, meaning that for each runthrough, 10% of the data is part of the holdout fold that we use for calculating accuracy and 90% of the data is part of the training data that we use to build the random forests. We use crosstab to create an overall confusion matrix, from which we are able to produce confusion matrices for each class label and accuracy.

KNN

K-nearest neighbors is a lazy evaluation algorithm requiring no training step. Our implementation takes as input a formatted data file, a value for k neighbors, and a distance metric of choice. Our knn.py file creates the KNNClassifier class, which contains the following functions: euclidean_distance, manhattan_distance, cosine_similarity, calculate_distance, and knn. To handle both categorical and continuous attributes, we use dummification to convert categorical attributes to continuous attributes. The algorithm selects the k-nearest points to the selected point and assigns the selected point to the plurality class. As in the other implementations, we use crosstab to create an overall confusion matrix, from which we are able to produce confusion matrices for each class label and accuracy.

Evaluation Procedures

Iris

The Iris dataset contains 151 records of data with 4 attributes each: sepal length, sepal width, petal length, and petal width (all in centimeters). We classify whether an iris is of species Iris-setosa, Iris-versicolor, or Iris-virginica based on these 4 attributes.

Model	**Hyperparameters**	**Accuracy**
C4.5	threshold=0.2	0.987
Random Forest	m=2, k=75, N=75, threshold=0.1	0.947
KNN	k=25, Cosine similarity	0.982

For the C4.5 algorithm, we tuned the threshold parameter, which determines how much the decision tree should split the data. We wanted to build a tree that was not too complex, which comes from a threshold that is too small, or too simple, which comes from a threshold that is too large. Also, if a threshold is too large, a root node may not even be selected, which would result in no decision tree. We started off with a threshold of 0.5, which resulted in an accuracy of 96%, but the decision tree was small, consisting of only 2 nodes and 3 leaves. We then reduced the threshold to 0.2 to increase the complexity of the tree and increase the accuracy. The threshold of 0.2 provided a decision tree that was descriptive but not overwhelmingly large. It consisted of 5 nodes and 5 leaves, and resulted in an accuracy of 99%. We also tried with a smaller threshold of 0.1, but this resulted in a larger and more complicated decision tree and an accuracy of 99%. Since the accuracy was similar to a threshold of 0.2, we decided to select a threshold of 0.2 which provided a descriptive and understandable decision tree with high accuracy.

For the Random Forest classifier, we tuned m (the number of attributes), k (the number of data points), N (the number of trees), and the threshold. We also used a cross validation with 10 folds. After doing some research, we decided to follow the logic of picking the optimal m value as being half of the number of attributes, the optimal k value as being half of the number of data points, and the N value as being no less than 50. Therefore, we tested m=2, k=75, N=50, and threshold=0.1, which resulted in an accuracy of 94%. When increasing the number of trees to 75 and 100 at m=2 and k=75, the accuracy resulted in a slightly higher value of 94.7%. Thus, we decided to select our hyperparameters as m=2, k=75, and N=75, since that resulted in the highest accuracy and was a good middle ground for the number of trees.

For KNN classifier, we tuned k (the number of neighbors to consider) and the distance metric (Euclidean, Manhattan, and Cosine similarity). We wanted to make sure we were not overfitting which is a result of a k value too small, or were not underfitting which is a result of a k value too large. We tested 3 different levels of k=3, 10, 25 with the Euclidean distance metric, resulting in the respective accuracies of 97%, 97.3%, 97.8%. Seeing that k=25 had the highest accuracy, we used that k value when testing for Manhattan and Cosine similarity distance metrics, resulting in the respective accuracies of 97.8% and 98.2%. Thus, we decided to go with looking at the 25 nearest neighbors and using a Cosine similarity distance metric, which resulted in the highest accuracy of 98.2% and made sense for the size of this dataset.

Out of all three methods, C4.5 performed the best and had the highest accuracy of 98.7%.

Heart

The Heart dataset consists of 919 records, with each record describing different characteristics of a patient such as their age, sex, resting bp, and cholesterol and measuring whether the patient had heart disease or not. This dataset contains a mix of continuous and categorical data.

Model	**Hyperparameters**	**Accuracy**
C4.5	threshold=0.18	0.825
Random Forest	m=6, k=460, N=50, threshold=0.05	0.867
KNN	k=25, Manhattan	0.763

We followed a similar process for tuning all implementations as we did with the Iris dataset. We tuned the same hyperparameters for each implementation.

For the C4.5 algorithm, we started off with a threshold of 0.2, which resulted in an accuracy of 81.3% but the decision tree was small, consisting of only 1 node and 3 leaves. We then reduced the threshold to 0.1 to increase the complexity of the tree and increase the accuracy. The threshold of 0.1 provided a decision tree that was very large and resulted in an accuracy of 88.6%. However, we chose a threshold in between the two values in order to reduce the complexity of the decision tree. Thus, we selected a threshold of 0.18 which resulted in a reasonably complex decision tree and an accuracy of 82.5%.

For the Random Forest classifier, we tested m=6, k=460, N=50, and threshold=0.05, which resulted in an accuracy of 86.7%. When selecting threshold values, we started from 0.2 and kept decreasing as the threshold value was too high to select a root node. When increasing the number of trees to 75 at m=6 and k=460, the accuracy resulted in a slightly lower value of 86.6%. Thus, we decided to select our hyperparameters as m=6, k=460, and N=50, since that resulted in the highest accuracy of 86.7%.

For KNN classifier, since the optimal k-value is the square root of the total number of data points, we chose larger k values for this dataset. First, we tested k=10 with all Euclidean, Manhattan, and Cosine similarity resulting in the respective accuracies of 70%, 75.6%, 71%. Seeing that Manhattan had the highest accuracy, we used that distance metric and tested at a higher k-value of 25 and 30, resulting in the respective accuracies of 76.3% and 76.1%. Thus, we selected our hyperparameters as k=25 and Manhattan distance metric, which resulted in the highest accuracy of 76.3% and made sense for the size of this dataset.

Out of all three methods, Random Forest performed the best and had the highest accuracy of 86.7%.

Credit

The Credit dataset consists of 690 records of credit approval decisions for individuals, including 15 features describing the applicants (the privacy of the individuals is protected). For each individual, a + for approved application or - for rejected application is recorded. This dataset contains a mix of continuous and categorical variables.

Model	**Hyperparameters**	**Accuracy**
C4.5	threshold=0.1	0.864
Random Forest	m=7, k=350, N=75, threshold=0.04	0.874
KNN	k=25, Manhattan	0.709

We followed a similar process for tuning all implementations as we did with the Iris and Heart datasets. We tuned the same hyperparameters for each implementation.

For the C4.5 algorithm, we started off with a threshold of 0.2, which resulted in an accuracy of 86.3%, but the decision tree was small. We then reduced the threshold to 0.1 to increase the complexity of the tree and increase the accuracy. The threshold of 0.1 provided a decision tree that was descriptive but not overwhelmingly large which consisted of 2 nodes and 3 leaves, and resulted in an accuracy of 86.4%. Thus, we selected a threshold of 0.1 which resulted in a reasonably complex decision tree and an accuracy of 86.4%.

For the Random Forest classifier, we tested m=7, k=350, N=50, and threshold=0.04, which resulted in an accuracy of 86%. When selecting threshold values we started from 0.2 and kept decreasing as the threshold value was too high to select a root node. When increasing the number of trees to 75 at m=7 and k=350, the accuracy resulted in a slightly higher value of 87.4%. Thus, we decided to select our hyperparameters as m=7, k=350, and N=75, since that resulted in the highest accuracy.

For KNN classifier, we tested the k values of 3, 10, 25 with the Euclidean distance metric, resulting in the respective accuracies of 66%, 64.9%, 67.6%. Seeing that k=25 had the highest accuracy, we used that k value when testing for Manhattan and Cosine similarity distance metrics, resulting in the respective accuracies of 70.9% and 69.2%. Thus, we selected our hyperparameters as k=25 and Manhattan distance metric, which resulted in the highest accuracy of 70.9% and made sense for the size of this dataset.

Out of all three methods, Random Forest performed the best and had the highest accuracy of 87.4%.

Comparative Study

Each method has various hyperparameters. In C4.5, threshold is a hyperparameter we had to tune. In Random Forest, m (the number of attributes), k (the number of data points), N (the number of trees), and threshold are hyperparameters we had to tune. In KNN, k (the number of neighbors to consider) and distance_metric (Euclidean, Manhattan, and Cosine similarity) are hyperparameters we had to tune. In the Random Forest classifier, we used 10-fold cross validation to split the data into training and validation (holdout) subsets. In this cross validation, the original dataset was divided into ten subsets and for each iteration one of the ten folds is used as a validation set with this process repeating until each fold has been used as a validation set exactly once. The purpose of the 10-fold cross validation is to detect overfitting or underfitting issues by evaluating the model on different data subsets.

Dataset	**Best Model**	**Accuracy**
iris	C4.5	0.987
heart	Random Forest	0.99
credit	Random Forest	0.86

C4.5 had the highest accuracy for one of the datasets, Iris, and the second highest accuracy for two of the datasets, Heart and Credit. Random Forest had the highest accuracy for two of the datasets, Heart and Credit, and the lowest accuracy for one of them, Iris. Meanwhile KNN had the second highest accuracy for one of the datasets, Iris, and the lowest accuracy for two of the datasets, Heart and Credit.

When selecting the best performing C4.5 threshold, we valued a decision tree that was sufficiently descriptive and had a sufficiently high accuracy. Depending on the size of the dataset, we found that lower thresholds ranging from 0.1 to 0.2 performed well. When selecting the best performing Random Forest hyperparameters, we found that selecting an m value that was half the number of attributes, a k value that was half the number of data points, an N value that was no less than 50, and threshold around 0.04 to 0.1 led to models with highest accuracies. When selecting the best performing KNN hyperparameters, we saw that choosing the 25 nearest neighbors and using Manhattan distance resulted in higher accuracies.

Overall, we saw that C4.5 and Random Forest performed the best and had the highest accuracies on all the datasets compared to KNN. Meanwhile, KNN had the lowest accuracy for two out of the three of the datasets, Heart and Credit.