COMPSCI 589 Homework 1 - Spring 2024

Due March 01, 2024, 11:55pm Eastern Time

1 Instructions

- This homework assignment consists of a programming portion. While you may discuss problems with your peers, you must answer the questions on your own and implement all solutions independently. In your submission, do explicitly list all students with whom you discussed this assignment.
- We strongly recommend that you use LaTeX to prepare your submission. The assignment should be submitted on Gradescope as a PDF with marked answers via the Gradescope interface. The source code should be submitted via the Gradescope programming assignment as a .zip file. Include with your source code instructions for how to run your code.
- We strongly encourage using Python 3 for your homework code. You may use other languages. In either case, you *must* provide us with clear instructions on how to run your code and reproduce your experiments.
- You may *not* use any machine learning-specific libraries in your code, e.g., TensorFlow, PyTorch, or any machine learning algorithms implemented in scikit-learn (though you may use other functions provided by this library, such as one that splits a dataset into training and testing sets). You may use libraries like numpy and matplotlib. If you are not certain whether a specific library is allowed, do ask us.
- All submissions will be checked for plagiarism using two independent plagiarism-detection tools. Renaming variable or function names, moving code within a file, etc., are all strategies that *do not* fool the plagiarism-detection tools we use. If you get caught, all penalties mentioned in the syllabus *will* be applied—which may include directly failing the course with a letter grade of "F".

  → Before starting this homework, please review this course's policies on plagiarism by reading Section 14 of the syllabus.

- The tex file for this homework (which you can use if you decide to write your solution in LaTeX) can be found here.
- The automated system will not accept assignments after **11:55pm on March 01**.

Programming Section (100 Points Total)

In this section of the homework, you will implement two classification algorithms: k-NN and Decision Trees. **Notice that you may not use existing machine learning code for this problem: you must implement the learning algorithms entirely on your own and from scratch.**

1. Evaluating the k-NN Algorithm

(50 Points Total)

In this question, you will implement the k-NN algorithm and evaluate it on a standard benchmark dataset: the Iris dataset. Each instance in this dataset contains (as attributes) four properties of a particular plant/flower. The goal is to train a classifier capable of predicting the flower's species based on its four properties. **You can download the dataset here.**

The Iris dataset contains 150 instances. Each instance is stored in a row of the CSV file and is composed of 4 attributes of a flower, as well as the species of that flower (its label/class). The goal is to predict a flower's species based on its 4 attributes. More concretely, each training instance contains information about the length and width (in centimeters) of the sepal of a flower, as well as the length and width (in centimeters) of the flower's petal. The label associated with each instance indicates the species of that flower: Iris Versicolor, Iris Setosa, or Iris Virginica. See Figure 1 for an example of what these three species of the Iris flower look like. In the CSV file, the attributes of each instance are stored in the first 4 columns of each row, and the corresponding class/label is stored in the last column of that row.

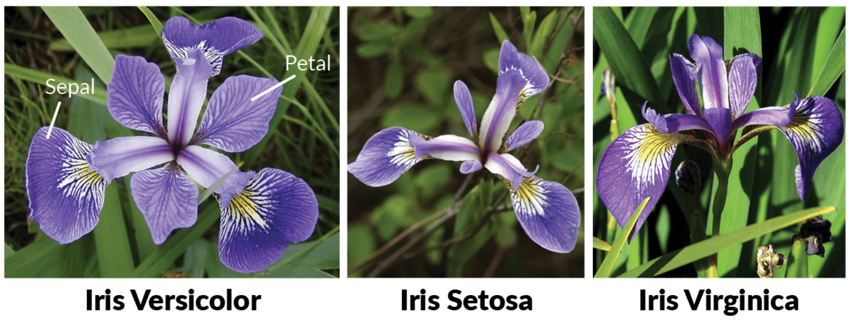


Figure 1: Pictures of three species of the Iris flower (Source: Machine Learning in R for beginners).

The goal of this experiment is to evaluate the impact of the parameter k on the algorithm's performance when used to classify instances in the training data, and also when used to classify new instances. For each experiment described below, you should use Euclidean distance as the distance metric and then follow these steps:

(a) shuffle the dataset to make sure that the order in which examples appear in the dataset file does not affect the learning process;[1]

(b) randomly partition the dataset into disjoint two subsets: a *training set*, containing 80% of the instances selected at random; and a testing set, containing the other 20% of the instances. Notice that these sets should be disjoint: if an instance is in the training set,

[1]If you are writing Python code, you can shuffle the dataset by using, e.g., the `sklearn.utils.shuffle` function.

it should not be in the testing set, and vice-versa.[2] The goal of splitting the dataset in this way is to allow the model to be trained based on just part of the data, and then to "pretend" that the rest of the data (i.e., instances in the testing set, which were *not* used during training) correspond to new examples on which the algorithm will be evaluated. If the algorithm performs well when used to classify examples in the testing set, this is evidence that it is generalizing well the knowledge it acquired after learning based on the training examples;

(c) train the k-NN algorithm using *only* the data in the training set;

(d) compute the *accuracy* of the k-NN model when used to make predictions for instances in the *training set*. To do this, you should compute the percentage of correct predictions made by the model when applied to the training data; that is, the number of correct predictions divided by the number of instances in the training set;

(e) compute the *accuracy* of the k-NN model when used to make predictions for instances in the *testing set*. To do this, you should compute the percentage of correct predictions made by the model when applied to the testing data; that is, the number of correct predictions divided by the number of instances in the testing set.

Important: when training a k-NN classifier, do not forget to normalize the features!

You will now construct two graphs. The first one will show the accuracy of the k-NN model (for various values of k) when evaluated on the training set. The second one will show the accuracy of the k-NN model (for various values of k) when evaluated on the testing set. You should vary k from 1 to 51, using only odd numbers $(1, 3, \ldots, 51)$. For each value of k, you should run the process described above (i.e., steps *(a)* through *(e)*) 20 times. This will produce, for each value of k, 20 estimates of the accuracy of the model over training data, and 20 estimates of the accuracy of the model over testing data.

Q1.1 (10 Points) In the first graph, you should show the value of k on the horizontal axis, and on the vertical axis, the average accuracy of models trained over the *training set*, given that particular value of k. Also show, for each point in the graph, the corresponding standard deviation; you should do this by adding error bars to each point. The graph should look like the one in Figure 2 (though the "shape" of the curve you obtain may be different, of course).

Q1.2 (10 Points) In the second graph, you should show the value of k on the horizontal axis, and on the vertical axis, the average accuracy of models trained over the *testing set*, given that particular value of k. Also show, for each point in the graph, the corresponding standard deviation by adding error bars to the point.

Ans 1.1/2 Given in Figure 3 is the graph for average accuracy over the training and testing and set and with standard deviation plotted as an error bar

Q1.3 (8 Points) Explain intuitively why each of these curves looks the way they do. First, analyze the graph showing performance on the training set as a function of k. Why do you think the graph looks like that? Next, analyze the graph showing performance on the testing set as a function of k. Why do you think the graph looks like that?

Ans 1.3 For smaller values of k, the decision boundary created will be over-fitting the training dataset but as the values of k increases, the model might be able to classify the data accurately and might misclassify data near the decision boundaries. For very large values of k, the model is

[2] If you are writing Python code, you can perform this split automatically by using the `sklearn.model_selection.train_test_split` function.

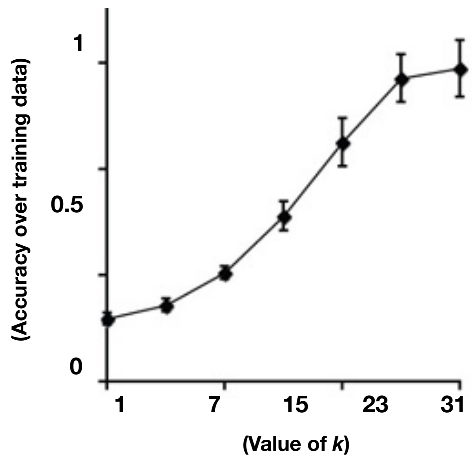


Figure 2: Example showing how your graphs should look like. The "shape" of the curves you obtain may be different, of course.

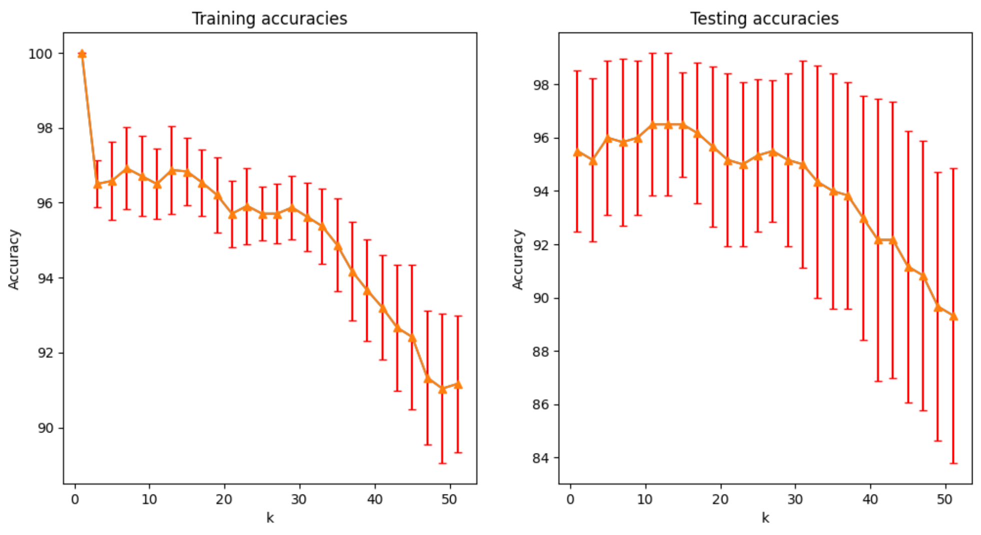


Figure 3: The graph for the average accuracy of the model trained over training set and testing set

prone to misclassifying data even further from the decision boundaries because there might be more labels for a specific class and for higher values of k, the majority labels might dominate the actual label(for reference on decision boundary check slide 02_CS589.pdf page 163).

For the training accuracy the graph starts with 100% accuracy for k = *1* because the nearest neighbour for a training data is the data itself(case for over-fitting). The accuracy then decreases and then almost remains the same till k = 15 but then keeps on decreasing for larger values of k, explained in the introduction of this answer.

For the testing accuracy the graph is a concave graph which slightly increases for k between *1 to 15* and then continue decreasing. This is again because for value of k for somewhere around *12-18*, the data is neither over-fitting(as seen in the case of k = *1* for training data) nor it is generalising the decision boundary to misclassify data further from the decision boundary. The only misclassified data in this range is the data near the decision boundary.

Q1.4 (6 Points) We say that a model is *underfitting* when it performs poorly on the training data (and most likely on the testing data as well). We say that a model is *overfitting* when it performs well on training data but it does not generalize to new instances. Identify and report the ranges of values of k for which k-NN is underfitting, and ranges of values of k for which k-NN is overfitting.

Ans 1.4 For *k 1-10* the model is over-fitting(especially for k = 1) as seen that the accuracy on training data is around is higher than the accuracy on testing data.

For *k 11-17* the model is neither over-fitting nor underfitting as it performs well on the training as well as testing data.

For the remaining values of *k 18-51* the model is underfitting as it performs poorly in both testing and training data

Q1.5 (6 Points) Based on the analyses made in the previous question, which value of k would you select if you were trying to fine-tune this algorithm so that it worked as well as possible in real life? Justify your answer.

Ans 1.5 Based on the previous question, I would select the value of k in the range *11 - 17* so that it works as well as possible in the real life. This is because the model performs well(accuracy greater than *96%*) for both training and testing data in this range.

Q1.6 (10 Points) In the experiments conducted earlier, you normalized the features before running k-NN. This is the appropriate procedure to ensure that all features are considered equally important when computing distances. Now, you will study the impact of *omitting* feature normalization on the performance of the algorithm. To accomplish this, you will repeat Q1.2 and create a graph depicting the average accuracy (and corresponding standard deviation) of k-NN as a function of k, when evaluated on the *testing set*. However, this time you will run the algorithm *without* first normalizing the features. This means that you will run k-NN directly on the instances present in the original dataset without performing any pre-processing normalization steps to ensure that all features lie in the same range/interval. Now ***(a)*** present the graph you created; ***(b)*** based on this graph, identify the best value of k; that is, the value of k that results in k-NN performing the best on the testing set; and ***(c)*** describe how the performance of this version of k-NN (without feature normalization) compares with the performance of k-NN *with* feature normalization. Discuss intuitively the reasons why one may have performed better than the other.

Ans 1.6

(a) Figure 4 displays the graph that was created after not performing normalisation

(b) The model performs the best for the value of k = *11*

(c) The model which included normalisation performs slightly better than the one which didn't perform normalisation. This is because without normalisation, one attribute could have a higher bias on the decision of the model than any other attribute. for eg: the values of *'length_sepal'* ranges from 4.3 - 7.9 whereas the values of *'width_petal'* ranges from 0.1 - 2.5 and while calculating the Euclidean Distance between 2 points, the *'length_sepal'* attribute will have a higher weightage in the distance than the *'width_petal'* hence the model might become biased towards *'length_sepal'*

2. Evaluating the Decision Tree Algorithm
(50 Points Total)

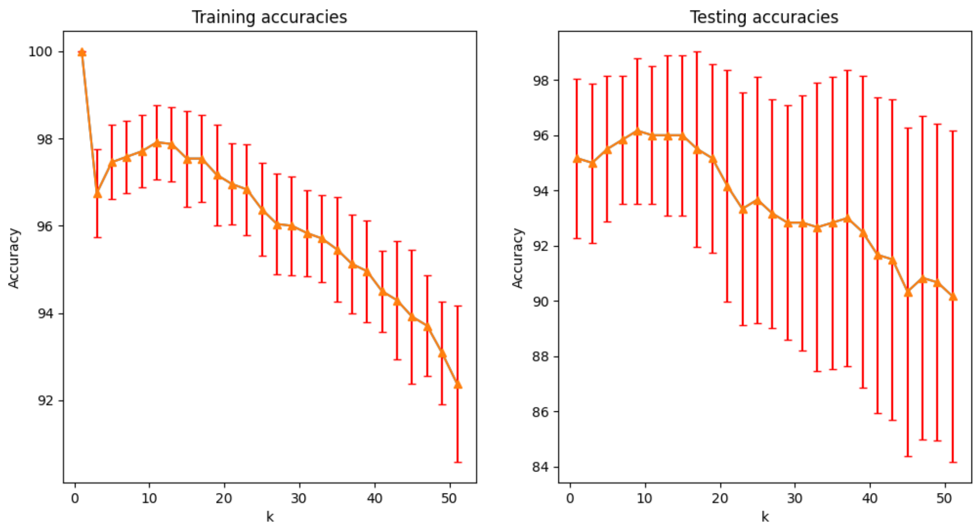


Figure 4: The graph for the average accuracy of the model trained over training set and testing set **without normalisation**

In this question, you will implement the Decision Tree algorithm, as presented in class, and evaluate it on the *1984 United States Congressional Voting* dataset. This dataset includes information about how each U.S. House of Representatives Congressperson voted on 16 key topics/laws. For each topic/law being considered, a congressperson may have voted yea, nay, or may not have voted. Each of the 16 attributes associated with a congressperson, thus, has 3 possible categorical values. The goal is to predict, based on the voting patterns of politicians (i.e., on how they voted in those 16 cases), whether they are Democrat (class/label 0) or Republican (class/label 1). **You can download the dataset here.**

Notice that this dataset contains 435 instances. Each instance is stored in a row of the CSV file. The first row of the file describes the name of each attribute. The attributes of each instance are stored in the first 16 columns of each row, and the corresponding class/label is stored in the last column of that row. For each experiment below, you should repeat the steps *(a)* through *(e)* described in the previous question—but this time, you will be using the Decision Tree algorithm rather than the k-NN algorithm. You should use the Information Gain criterion to decide whether an attribute should be used to split a node.

You will now construct two histograms. The first one will show the accuracy distribution of the Decision Tree algorithm when evaluated on the training set. The second one will show the accuracy distribution of the Decision Tree algorithm when evaluated on the testing set. You should train the algorithm 100 times using the methodology described above (i.e., shuffling the dataset, splitting the dataset into disjoint training and testing sets, computing its accuracy in each one, etc.). This process will result in 100 accuracy measurements for when the algorithm was evaluated over the training data, and 100 accuracy measurements for when the algorithm was evaluated over testing data.

Q2.1 (12 Points) In the first histogram, you should show the accuracy distribution when the algorithm is evaluated over *training data*. The horizontal axis should show different accuracy values, and the vertical axis should show the frequency with which that accuracy was observed while conducting these 100 experiments/training processes. The histogram should look like the one in Figure 5 (though the "shape" of the histogram you obtain may be different, of course). You should also report the mean accuracy and its standard deviation.

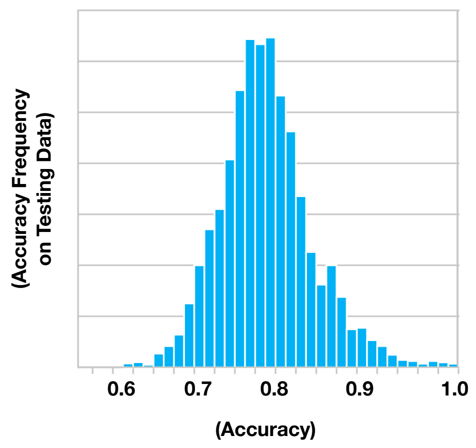


Figure 5: Example showing how your histograms should look like. The "shape" of the histograms you obtain may be different, of course.

Q2.2 (12 Points) In the second histogram, you should show the accuracy distribution when the algorithm is evaluated over *testing data*. The horizontal axis should show different accuracy values, and the vertical axis should show the frequency with which that accuracy was observed while conducting these 100 experiments/training processes. You should also report the mean accuracy and its standard deviation.

Ans 2.1/2 Figure 6 shows the histogram for the accuracy distribution evaluated over training and testing data along with their mean and standard deviation

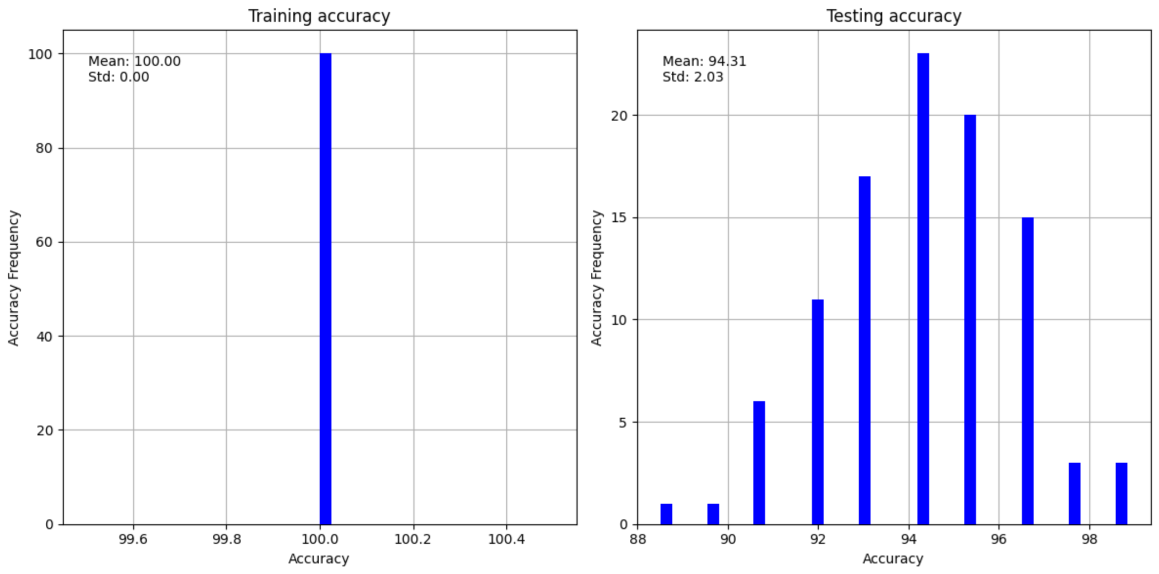


Figure 6: Accuracy distribution evaluated over training and testing data for decision tree algorithm

Q2.3 (12 Points) Explain intuitively why each of these histograms looks the way they do. Is there more variance in one of the histograms? If so, why do you think that is the case? Does one histogram show higher average accuracy than the other? If so, why do you think that is the case?

Ans 2.3 For the training data, the accuracy is 100% which means the model can classify all the training data correctly. This is because the decision tree trained is able to classify all the training data based on the attributes and the decision boundary hence made is very complex which fits all the training data without any misclassified data *0 & 1.*

For the testing accuracy, since the model is generalised on the training data, it misclassifies several testing data which is new to the model. This is an example of overfitting as it performs really well on the training data but not so much in the testing data.

There is more variance in the testing accuracy because the model has been trained over training data and correctly classifies all the training data but when it encounters an unseen data, it misclassifies for a lot of them. This results in the accuracy ranging *89% to 99%*

The training data has shown higher average accuracy than the testing data because the model has learned all the training data and is able to classify all of them. The accuracy drops when it encounters and unseen data as displayed in the testing accuracy histogram

Q2.4 (8 Points) By comparing the two histograms, would you say that the Decision Trees algorithm, when used in this dataset, is underfitting, overfitting, or performing reasonably well? Explain your reasoning.

Ans 2.4 The decision tree algorithm is overfitting in this case as it performs really well with *100%* accuracy on the training data but performs poorly on the testing data with *94%* average accuracy. This is because the model has learned all the dataset for the training data and is able to classify everyone of them correctly but that is not the case with the unseen(testing) data.

Q2.5 (6 Points) In class, we discussed how Decision Trees might be non-robust. Is it possible to experimentally confirm this property/tendency via these experiments, by analyzing the histograms you generated and their corresponding average accuracies and standard deviations? Explain your reasoning.

Ans2.5 Decision trees might be non robust as seen in the testing data accuracy. For every iteration we are creating a new tree and the accuracy for the testing data varies from 89% to 99%. This shows that a slight change in the tree results in a high variance for the testing data.

[QE.1] Extra points (15 Points) Repeat the experiments Q2.1 to Q2.4, but now use the Gini criterion for node splitting, instead of the Information Gain criterion.

Ans E.1

(a) Figure 7 shows the accuracy of the decision tree when using gini impurity when selecting the attributes

(b) As explained in Ans 2.3

(c) Same as answer 2.4

[QE.2] Extra points (15 Points) Repeat the experiments Q2.1 to Q2.4 but now use a simple heuristic to keep the tree from becoming too "deep"; i.e., to keep it from testing a (possibly) excessive number of attributes, which is known to often cause overfitting. To do this, use an *additional* stopping criterion: whenever more than

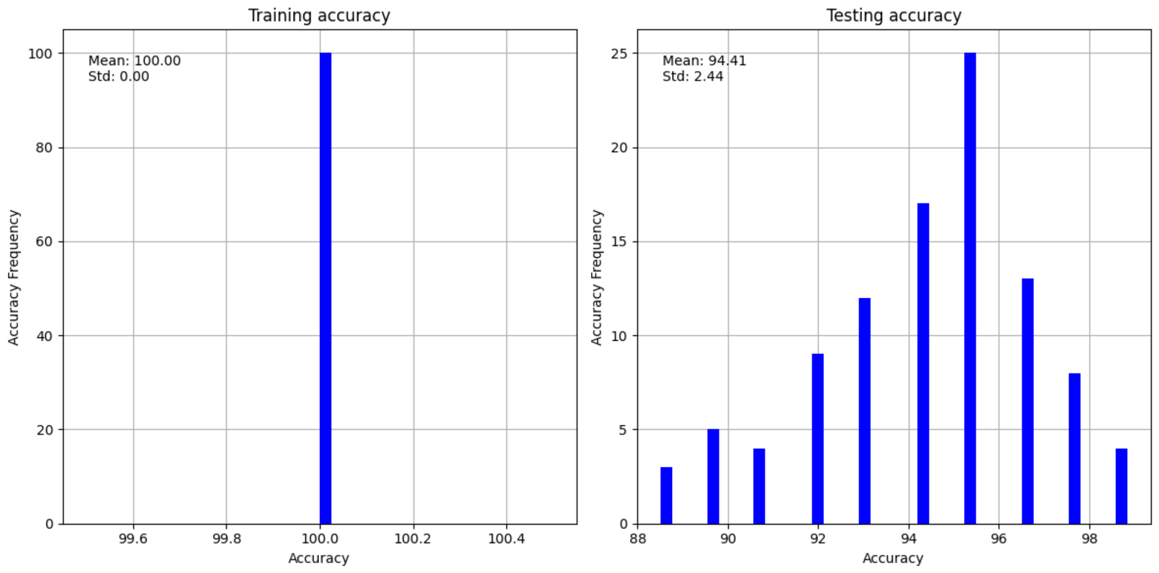


Figure 7: The graph for the average accuracy of the model trained over training set and testing set but using **gini impurity** for selecting attributes

85% of the instances associated with a decision node belong to the same class, do not further split this node. Instead, replace it with a leaf node whose class prediction is the majority class within the corresponding instances. E.g., if 85% of the instances associated with a given decision node have the label/class *Democrat*, do not further split this node, and instead directly return the prediction *Democrat*.

Ans E.2

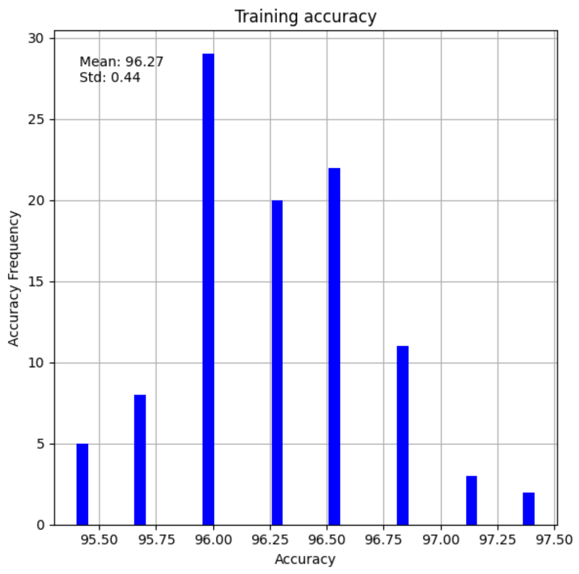
(a) Figure 8 shows the accuracy of the decision tree when using using a stopping criteria of not splitting the node further when 85% of the instances on the decision node belong to the same class

(b) With an additional stopping criteria of not splitting the node further when 85% of the instances on the decision node belong to the same class, we can see that the model is giving poorer performance for training data when compared to without having an stopping criteria. But at the same time, the testing accuracy has increased. This is because model can now generalise well for unseen data as well and is not over-fitting on the training data.

The testing data has higher variance than the training data because the tree is trained on the training data and it generalises on the training data with the exception of a few outliers. But for unseen data, the error rate might be higher or in some cases it might even be very low. Because of this the variance is higher in the testing data.

The training data shows higher accuracy than the testing data because the tree is being created based on the training data and is able to generalise all the data except few outliers but that's not the case for testing data.

(c) Based on the histogram, the model is performing reasonably well and is neither overfitting nor underfitting. the average accuracy across the training and testing data is almost equal which should be the case as we have stopped creating complex decision boundaries and added a stopping criteria of not splitting the node further when 85% of the instances on the decision node belong to the same class. With this criteria, the training data is not overfitted and is also able to perform reasonably well for unseen/testing data



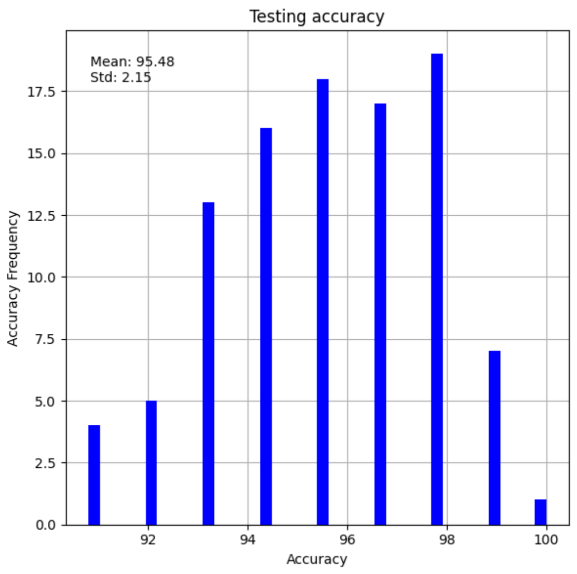


Figure 8: The graph for the average accuracy of the model trained over training set and testing set but when using using a stopping criteria of not splitting the node further when 85% of the instances on the decision node belong to the same class