COMPSCI 589 Homework 1 - Spring 2022

Due February 15, 2022, 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 you to use 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".

- 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 February 15**.

Programming Section (100 Points Total)

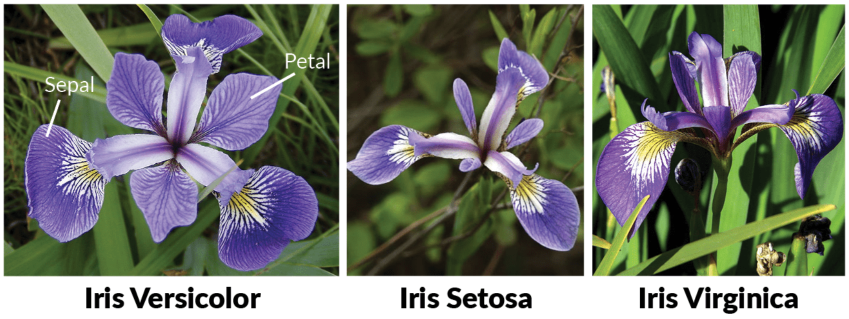
In this section of the homework, you will implement two classification algorithms: k-NN and Decision Trees. **Notice that you may not use existing machine learning code for this problem: you must implement the learning algorithms entirely on your own and from scratch.**

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.



Iris Versicolor **Iris Setosa** **Iris Virginica**

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;

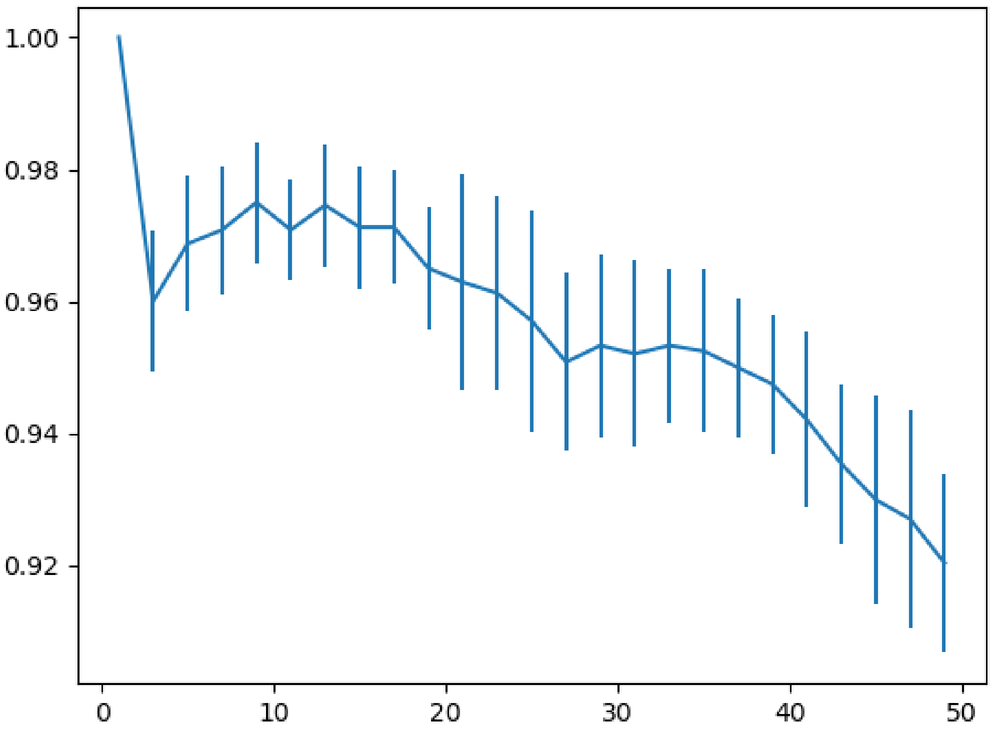
(e) compute the *accuracy* of the k-NN model when used to make predictions for instances in the *testing set*. To do this, you should compute the percentage of correct predictions made by the model when applied to the testing data; that is, the number of correct predictions divided by the number of instances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

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

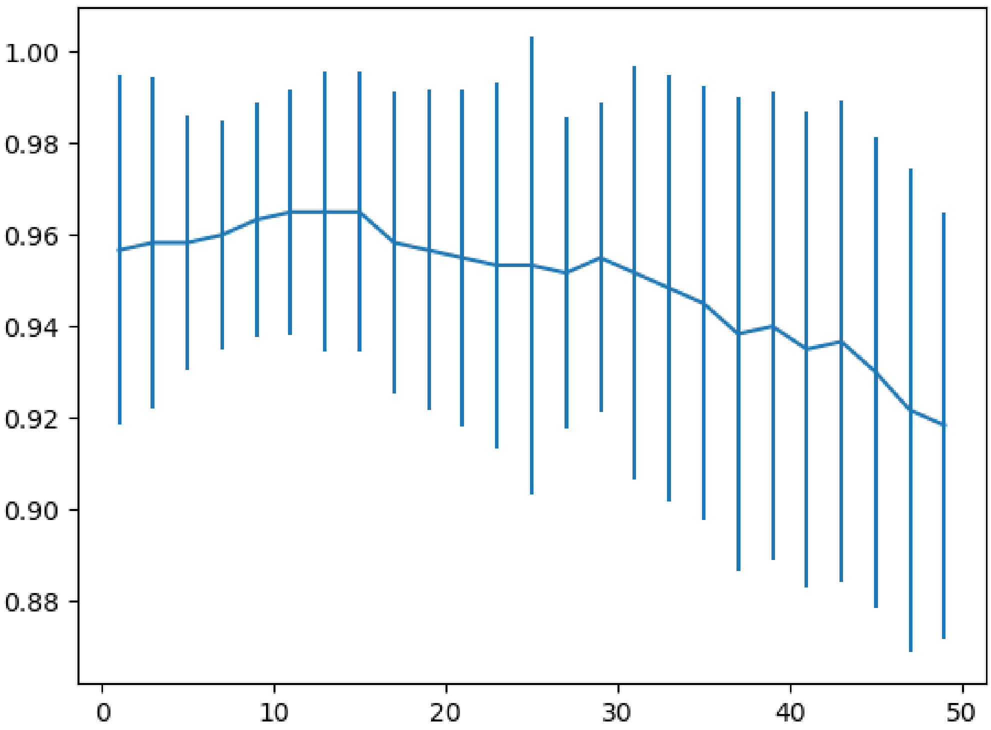
You will now construct two graphs. The first one will show the accuracy of the k-NN model (for various values of k) when evaluated on the training set. The second one will show the accuracy of the k-NN model (for various values of k) when evaluated on the testing set. You should vary k from 1 to 51, 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 (12 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 **??** (though the "shape" of the curve you obtain may be different, of course).

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



Q1.2 (12 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.



Q1.3 (10 Points) Explain intuitively why each of these curves look 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?

For training set, the accuracy starts off with 100 percent as we are training and testing on the same data. Its next best performance is at around k=10, after which there is a constant decline, and a steep decline as k nears 51 as we are considering based on too many points. For testing set, the accuracy marginally improves up to around k = 16 after which there is a constant but slow decline up till k = 51. This is because accuracy reduces when you take too many values into account for a small data set.

Q1.4 (8 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.

The model seems to be under fitting for high values [¿20] of k as the accuracy's are more or less the same in that range. And the model is over fitting for low values 1 and [6-10] of k as training accuracy is very high and testing accuracy is relatively low.

Q1.5 (8 Points) Based on the analyses made in the previous question, which value of k you would select if you were trying to fine-tune this algorithm so that it worked as well as possible in real life? Justify your answer.

I would pick k=17 as that value of k produces the highest accuracy for this in the testing set, with a relatively low standard deviation. It is better to go with the k value for which testing accuracy is high as it shows that the model generalises well on new data that it did not see while training.

2. Evaluating the Decision Tree Algorithm

(50 Points Total)

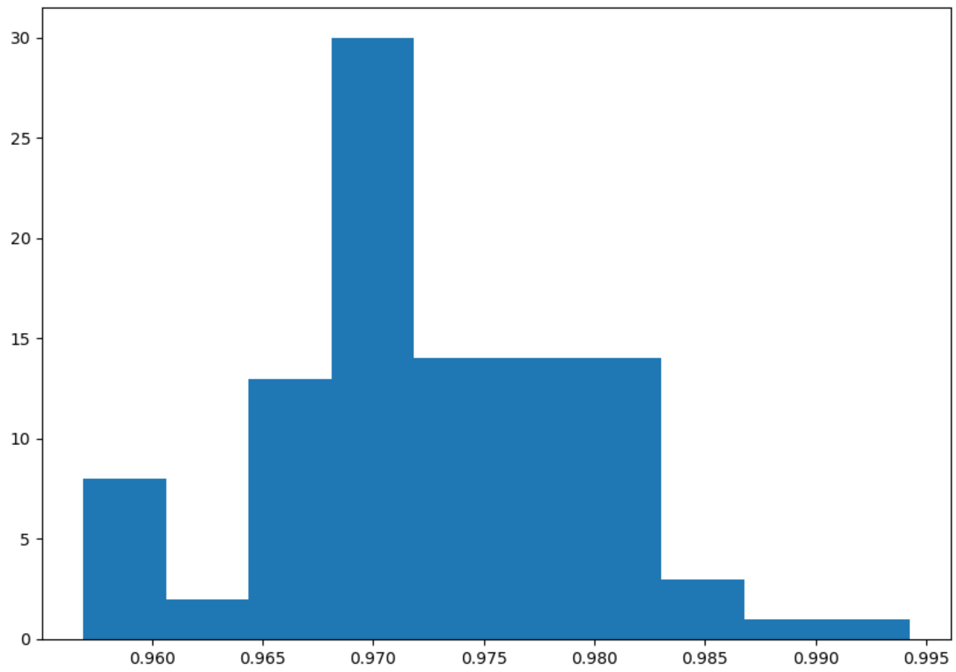
In this question, you will implement the Decision Tree algorithm, as presented in class, and evaluate it on the *1984 United States Congressional Voting* dataset. This dataset includes votes for each U.S. House of Representatives Congressperson on the 16 key votes. For each topic/law being considered, 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 on those 16 cases), whether they are a Democrat (class/label 0) or a 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. 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.

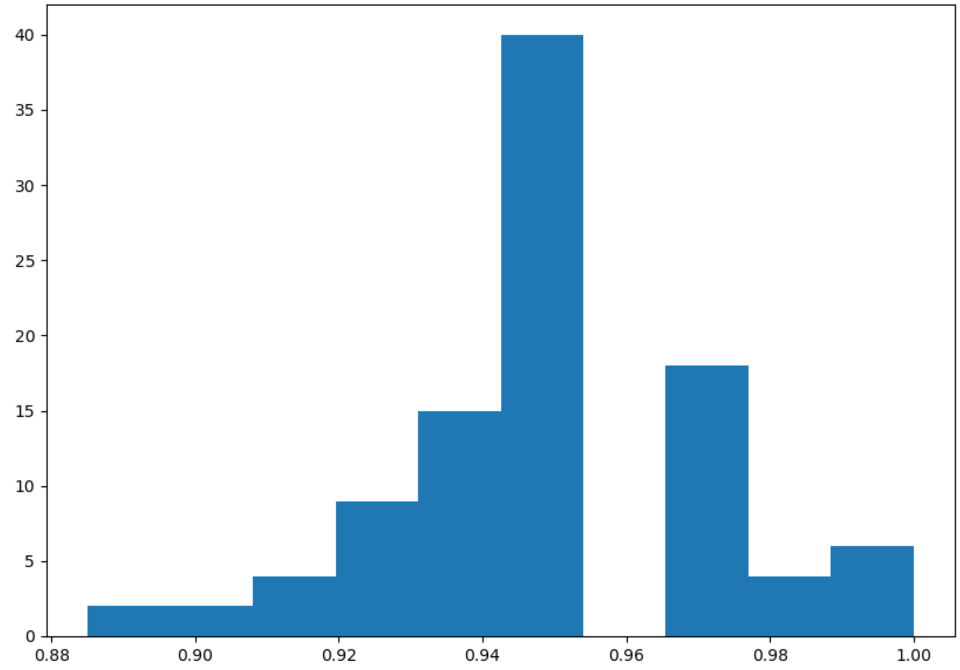
Q2.1 (12 Points) In the first histogram, you should show the accuracy distribution when the algorithm was evaluated over *training data*. The horizontal axis should show different accuracy values, and the vertical axis should show the frequency with which that accuracy was observed while conducting these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

The average training accuracy is: 0.9722413793103449. The standard deviation of the training accuracy is: 0.00697062708517885



Q2.2 (12 Points) In the second histogram, you should show the accuracy distribution when the algorithm was 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.

The average of the testing accuracy is: 0.9463218390804596.The standard deviation of the testing accuracy is: 0.022818625181469942



Q2.3 (12 Points) Explain intuitively why each of these histograms look 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?

Histograms are centred around the mean value and show the distribution of the accuracies.Training data has a higher accuracy and more values closer to the mean compared to testing data. This is a definite consequence as the the model is being tested on data it has never seen before that too in a shuffled manner. But a good question is, why is the training data accuracy not 1 ? This is because while creating the decision tree when we reached a situation where the decision is not splitting into three different decisions i.e [0, 1, 2] we made it the decision node a leaf node with its target pointing to the majority class. So even when we are training on the training data, the model is wrong in those leaf node cases. With regards to comparative accuracy and standard deviation: testing set has more accuracy, and training set has more standard deviation.

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.

The model is performing well as the training accuracy is between 96 and 99.5 while the testing is between 89 to 99 with almost the same mean, and low standard deviation.

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.

Yes, when I created the decision tree by letting the attributes split as much as they can, my accuracies for training data were 100 percent every time. But as soon as I handled the leaf nodes differently the accuracy dropped by 3 percent. This means that decision tree has significantly changed. Same way, though the testing and training accuracies are just 3 points away, the fact that there is a drop in accuracy and increase in standard deviation for newly seen data shows that decision trees are not robust. We can even test this by manually changing some training data and then seeing the accuracy drop.

Extra points (15 Points) Repeat the experiment above but now using the Gini criterion for node splitting, instead of the Information Gain criterion.