# COMPSCI 589 Homework 1 - Spring 2026

Due **February 24**, 11:55 pm Eastern Time

## Instructions

- This homework assignment consists of a programming part.
- While you may discuss problems with your peers (e.g., to discuss high-level approaches), 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.
- Submissions must be typed. You must use LaTeXto prepare your submission. Handwritten and scanned submissions will not be accepted.
- 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.
- We strongly encourage using Python 3 for your homework code. You may use other languages. In either case, you *must* provide clear instructions for running your code and reproducing your experiments.
- You *may not* use any machine learning-specific libraries in your code, e.g., TensorFlow, PyTorch, or 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 also use libraries such as 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. Your code will also be compared with submissions from your classmates and with publicly available implementations online.
- 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 the corresponding section of the syllabus.

- The automated system will not accept assignments after 11:55 pm on February 24.
- The TeX file for this homework (which you should use to write your solutions in LaTeX), as well as the datasets you will investigate, can be found on Canvas under "*Homework #1 - Supporting Files*".

# Programming Assignment [100 Points, total]

In this section of the homework, you will implement two classification algorithms: $k$-NN and Decision Trees. **Recall 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 Wisconsin Breast Cancer Diagnostic (WDBC) dataset. Each instance in this dataset is described by 30 features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. These features describe the characteristics of the cell nuclei present in the image. The goal is to train a classifier capable of predicting whether a breast mass is malignant or benign.

The WDBC dataset contains 569 instances. Each instance is stored in a row of the CSV file and consists of 30 attributes describing a breast mass, along with a corresponding label (the instance's class) indicating whether it is benign or malignant. The goal is to predict whether a mass is benign or malignant based on its 30 attributes. More concretely, each training instance contains information about a specific breast mass, such as its radius, texture, area, perimeter, and concave points. In the CSV file, the attributes of each instance are stored in the first 30 columns of each row, and the corresponding class/label is stored in the last column of that row. As discussed in the Instructions, this dataset is available on Canvas under "*Homework #1 - Supporting Files*".

The goal of this experiment is to evaluate the impact of the parameter $k$ on the algorithm's performance when used to classify **(i)** *training instances* and **(ii)** *new (test) instances*. These two cases are evaluated independently. This will allow us to investigate, for example, whether the classifier is overfitting to the training set, and whether it can learn patterns from the training data and generalize them well enough to make accurate predictions for new instances/patients. For each experiment described below, you should use Euclidean distance as the distance metric and then follow these steps:

1. Shuffle the dataset to make sure that the order in which examples appear in the dataset file does not affect the learning process;[1]
2. Randomly partition the dataset into two disjoint 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, 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., the 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;
3. Run the $k$-NN algorithm using *only* the data in the training set;
4. 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;
5. 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.

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

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

**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 *(1)* through *(5)*) 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.

- (**Question 1.1 - 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 evaluated on 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 type of graph you construct should look like the one in Figure 1.

  **Important**: The "shape" of the curve you obtain will most likely be slightly different. The figure below is just an example of the type of graph you should construct.

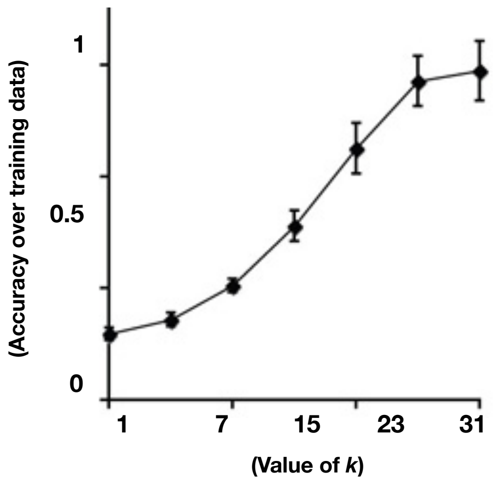

Figure 1: Example showing how the graph should be presented. Your curve will likely have a different shape.

- (**Question 1.2 - 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 evaluated on 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.
- (**Question 1.3 - 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?
- (**Question 1.4 - 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.

- **(Question 1.5 - 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.

- **(Question 1.6 - 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.

## 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 a Car Evaluation dataset. This dataset includes six categorical attributes about different cars: buying price, maintenance price, number of doors, capacity (in terms of the number of people it can carry), size of the luggage boot, and the estimated safety level of the car. Each of these attributes has a different set of possible values. The dataset also contains the class/label of each car, which describes how good a particular car is. There are four possible classes/labels: *unacceptable*, *acceptable*, *good*, and *very good*. The goal is to use the six features of a novel car to predict its quality level.

This dataset contains 1,728 instances and, as discussed in the Instructions, can be found on Canvas under "*Homework #1 - Supporting Files*". 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 6 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 *(1)* through *(5)* 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 which attribute should be used to split each 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.

- **(Question 2.1 - 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 type of histogram you should construct should (*approximately*) look like the one in Figure 2. You should also report the mean accuracy and its standard deviation.

  **Important**: The "shape" of the histogram you obtain will most likely be slightly different. The figure below is just an example of the type of histogram you should construct.

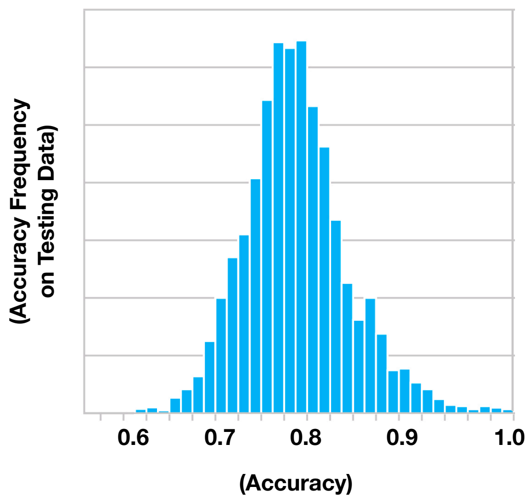

Figure 2: Example showing how the histograms should be presented. Your histograms will likely have a slightly different shape.

- (**Question 2.2 - 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.
- (**Question 2.3 - 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?
- (**Question 2.4 - 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.
- (**Question 2.5 - 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.

## Extra Credit Questions

**[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.

**[QE.2] Extra points (15 Points)**. Repeat the experiments Q2.1 to Q2.4, but now use a simple heuristic to prevent the tree from becoming too "deep"; that is, to keep it from testing a possibly unnecessary number of attributes, which is known to often cause overfitting. To do this, use an *additional* stopping criterion: whenever more than 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% (or more) of the instances associated with a given decision node have the label/class *Very Good*, do not split this node further, and instead directly return the prediction *Very Good*.