Homework 4 CS699 A1, Spring 2025

Due date: 2/24

**Note:** For problems 1, 2, and 3, you must do all calculations yourself and must show all calculation steps. You may use a spreadsheet software only for calculations.

**Problem 1 (10 points).** Suppose that you have the following training dataset for classification:

|  |  |  |  |
| --- | --- | --- | --- |
| A1 | A2 | A3 | Class |
| 6.6 | 65.2 | 4.1 | 0 |
| 6.4 | 78.9 | 5.0 | 0 |
| 7.2 | 61.1 | 5.0 | 1 |
| 7.0 | 45.8 | 6.1 | 1 |
| 7.1 | 54.2 | 6.1 | 1 |
| 6.4 | 58.7 | 6.1 | 0 |
| 6.0 | 66.6 | 5.6 | 0 |
| 6.2 | 96.1 | 6.0 | 1 |
| 5.6 | 100.0 | 6.1 | 0 |
| 6.0 | 85.9 | 6.6 | 0 |

Classify an object X: <A1= 5.8, A2 = 66.6, A3 = 4.9> using the KNN method that we discussed in the class. Use the Euclidean distance.

1. Assuming K = 3
2. Assuming K = 5

To classify X with A1 equals 5.8, A2 equals 66.6, and A3 equals 4.9 using the KNN method, we need to calculate the Euclidean distance between X and each point in the dataset. The Euclidean distance formula is the square root of the sum of the squared differences between corresponding attributes. First, calculating the distances, we get approximately 1.8 for the first data point, 12.3 for the second, 5.7 for the third, 20.9 for the fourth, 12.5 for the fifth, 8.0 for the sixth, 0.73 for the seventh, 29.5 for the eighth, 33.4 for the ninth, and 19.4 for the tenth. Sorting these from smallest to largest, the three closest points to X are from class 0, meaning for K equals 3, X is classified as class 0. Extending to K equals 5, four out of the five closest points are also class 0, so X is still classified as class 0. Therefore, for both K equals 3 and K equals 5, X is classified as class 0.

**Problem 2 (10 points).** Consider the following dataset, which is used for classification:

|  |  |  |
| --- | --- | --- |
| A1 | A2 | Class |
| Low | On | N |
| Middle | On | N |
| Middle | Off | P |
| High | On | N |
| High | Off | P |
| Low | Off | N |
| High | Off | N |
| Low | On | N |
| Low | Off | N |
| Middle | On | P |
| High | On | P |
| High | Off | P |
| Middle | On | N |

Classify a new object X = <A1 = Low, A2 = On> using the Naïve Bayes algorithm that we discussed in the class.

To classify the new object X with A1 = Low and A2 = On using the Naïve Bayes algorithm, we start by figuring out the prior probabilities for each class, N and P, based on the training dataset. We count how many times each class appears and find that class N shows up 6 times while class P shows up 5 times. This gives us prior probabilities of about 0.545 for N and 0.455 for P. Next, we look at the conditional probabilities for each feature given each class. For A1 = Low, we see that P(A1 = Low | N) is around 0.667, but P(A1 = Low | P) is 0 since there are no Low values in class P. For A2 = On, P(A2 = On | N) is 0.5, and P(A2 = On | P) is 0.6.

Now, we use Bayes' theorem to calculate the posterior probabilities for each class given the features. For class N, we get P(N | A1 = Low, A2 = On) by multiplying 0.667 by 0.5 by 0.545, which gives us about 0.182. For class P, the result is 0 because P(A1 = Low | P) is 0.

In the end, we compare the probabilities and see that class N has a higher probability than class P. So, we classify the new object X as belonging to class N. In short, using the Naïve Bayes algorithm, we classify the new object X = <A1 = Low, A2 = On> as class N.

**Problem 3 (10 points).** The following table shows actual values and predicted values of a test result:

|  |  |
| --- | --- |
| Actual | Predicted |
| 24 | 19.5 |
| 21.6 | 23.6 |
| 34.7 | 21.5 |
| 33.4 | 26.3 |
| 36.2 | 28.7 |
| 28.7 | 21.3 |
| 22.9 | 24.2 |
| 27.1 | 26.1 |
| 16.5 | 20.4 |
| 18.9 | 24.3 |

Calculate MAE, ME, MPE, MAPE, and RMSE.

To calculate the error metrics for the given actual and predicted values, we start with the actual values as [24, 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9] and the predicted values as [19.5, 23.6, 21.5, 26.3, 28.7, 21.3, 24.2, 26.1, 20.4, 24.3]. First, we compute the errors by subtracting each predicted value from the corresponding actual value, which gives us the error values as [4.5, -2.0, 13.2, 7.1, 7.5, 7.4, -1.3, 0.9, -3.9, -5.4]. Next, we calculate the Mean Absolute Error (MAE) by taking the average of the absolute values of these errors. So, we find MAE by summing the absolute errors, which are [4.5, 2.0, 13.2, 7.1, 7.5, 7.4, 1.3, 0.9, 3.9, 5.4], resulting in a total of 53.2, and then dividing by the number of observations, which gives us MAE = 53.2 / 10 = 5.32. Then, we calculate the Mean Error (ME) by taking the average of the errors themselves. This involves summing the errors, resulting in 2.8, and dividing by the number of observations, giving us ME = 2.8 / 10 = 0.28. For the Mean Percentage Error (MPE), we calculate this by dividing each error by the corresponding actual value and multiplying by 100 to get the percentage errors. After calculating these, we find the average percentage error, which results in MPE = (4.5 / 24 \* 100 + -2.0 / 21.6 \* 100 + ... + -5.4 / 18.9 \* 100) / 10, yielding approximately 6.11%. The Mean Absolute Percentage Error (MAPE) is computed similarly, but we take the absolute values of the percentage errors, resulting in MAPE = (|4.5 / 24| \* 100 + |-2.0 / 21.6| \* 100 + ... + |-5.4 / 18.9| \* 100) / 10, which gives us about 19.54%. Finally, we calculate the Root Mean Squared Error (RMSE) by squaring each error, averaging those squares, and then taking the square root of that average. This results in RMSE = sqrt((4.5² + -2.0² + 13.2² + 7.1² + 7.5² + 7.4² + -1.3² + 0.9² + -3.9² + -5.4²) / 10), yielding approximately 6.38.

In summary, the calculated error metrics are as follows: MAE is approximately 5.33, ME is about 2.81, MPE is about 6.11 percent, MAPE is approximately 19.54 percent, and RMSE is around 6.38. These metrics provide insights into the accuracy and reliability of the predictions compared to the actual values.

**Problem 4 (20 points).** The KNN algorithm, which we discussed in the class, gives the same weight to all neighbors. There is another type of KNN, which assigns different weights to neighbors. One such algorithm implemented in R is *kknn*.

1. Study the *kknn* algorithm and write a description of the algorithm, including how it works. You should not copy from a documentation, a web page, or a response from an AI tool (such as ChatGPT). You must write the description in your own words. Your description must be at least two-page long and must be written using font size 12pt and single spaced.

The kknn algorithm is a powerful and intuitive method used for classification tasks, and it is based on a simple idea similar things belong together. Imagine you have a group of friends, and you want to figure out which friend is most like you based on certain traits like interests or hobbies. The kknn algorithm works in a similar way where we look at the features or attributes of the items we want to classify, and we compare them to the items in our dataset.

To start we need to identify the neighbors of the item we want to classify. This is done by looking at the nearest neighbors which are the closest items in the dataset to our item. We use a method called Euclidean distance to measure how far apart these items are. The Euclidean distance is like measuring the straight-line distance between two points on a map. For example, if we have two points A and B we can calculate the distance between them by using the formula

d = sqrt(A1 - B1 squared + A2 - B2 squared + A3 - B3 squared)

In this formula A1 A2 and A3 represent the features of point A while B1 B2 and B3 represent the features of point B. This formula helps us understand how similar or different the two points are based on their attributes.

Once we have calculated the distances the next step is to sort the items from our dataset based on how close they are to the item we are trying to classify. We then select the top k nearest neighbors. The choice of k is important, it represents the number of neighbors we want to consider when making our classification. A common choice for k is 3 or 5 but it can be adjusted based on the dataset and the problem at hand.

Now the big difference with kknn compared to regular KNN is how it handles these neighbors. In standard KNN all neighbors are treated equally which means they all contribute the same amount to the final decision. However, in kknn we assign different weights to the neighbors based on their distance from the item being classified. This means that closer neighbors have a bigger influence on the classification than those that are farther away. For example, if one neighbor is very close to the item it might count double while a neighbor that is further away might only count half.

To calculate the weights, we can use a simple method like inverse distance weighting where the weight for each neighbor is calculated as

Weight = 1 divided by Distance

This way if a neighbor is closer, it has a higher weight and if it is further away it has a lower weight. This weighting system allows the kknn algorithm to focus more on the most relevant neighbors leading to potentially better classification results.

After we have the weights for each neighbor the next step is to make the final classification decision. For classification tasks we look at the class labels of the nearest neighbors. We count how many neighbors belong to each class and apply their weights. For example if we have three neighbors and their weights are as follows neighbor 1 class A weight 0.6 neighbor 2 class A weight 0.3 and neighbor 3 class B weight 0.1 we can determine the total weight for each class. In this case class A would have a total weight of 0.6 plus 0.3 equals 0.9 while class B would have a total weight of 0.1.

Finally we classify our item based on the class with the highest total weight. If class A has the highest weight we assign our item to class A. If class B has the highest weight we assign it to class B. This way kknn allows us to make more informed decisions based on the closest neighbors and their significance.

The kknn algorithm helps us classify items by finding similar items in the dataset and assigning different weights to those neighbors based on their distance from the item being classified. By focusing more on closer neighbors kknn provides a better classification result making it a powerful tool for various applications in machine learning and data analysis. Whether its classifying emails as spam or not or identifying the type of flower based on its features kknn is a useful algorithm that balances simplicity with effectiveness.

The kknn algorithm has many practical applications beyond basic classification tasks. It is widely used in various fields such as healthcare, finance, and marketing. For instance, in healthcare, kknn can help predict patient outcomes based on their medical history and other relevant factors. In finance, it can be used to classify loan applications as high or low risk by analyzing applicant features. Marketing teams can leverage kknn for customer segmentation, allowing them to target specific groups based on purchasing behavior and preferences. Additionally, kknn can be applied in image recognition, where it classifies images based on the features extracted from them. The versatility of kknn, combined with its ability to handle different types of data and its robustness in various scenarios, makes it a valuable tool for data-driven decision-making across multiple industries. Overall, kknn is not just limited to classification but can also enhance predictive modeling, helping organizations to gain insights and make informed choices based on data.

1. What are tunable parameters of the *kknn*, which is implemented in R? You must show the name and description of each tunable parameter.

The kknn algorithm in R has several tunable parameters that allow users to customize the model for better performance based on the specific dataset and task requirements. One of the primary parameters is k which specifies the number of nearest neighbors to consider during the prediction process. The choice of k is crucial as a smaller value may lead to a model that is overly sensitive to noise in the data while a larger value can smooth out predictions and overlook local patterns. Another important parameter is distance which determines the distance metric used to calculate how far apart the points are. Common distance metrics include Euclidean distance Manhattan distance and Minkowski distance. The selection of the distance metric can significantly impact the models effectiveness.

The kernel parameter defines the weighting function applied to the neighbors. Different options might include uniform weighting where all neighbors contribute equally or distance based weights where closer neighbors have a greater impact on the classification outcome. This choice can influence how much each neighbor affects the final decision. Additionally the scale parameter indicates whether to scale the data before applying the kknn algorithm. Scaling can be essential for distance based algorithms to ensure that all features contribute equally to the distance calculations especially if they are on different scales.

Lastly the fitted parameter specifies whether to fit the model with the training data or to use existing fitted values which can be beneficial when evaluating the model on new data without retraining it. By adjusting these tunable parameters users can optimize the kknn algorithm to achieve better classification results tailored to their specific datasets and applications.

1. In the *L4.R* code, there is a KNN example which uses *tuneGrid* for parameter tuning. Using this example as a model, write a R code that builds and tests a *kknn* model. You must use the same drug consumption dataset and must use *tuneGrid* for parameter tuning. You must specify your own search grid. Capture the screenshot of the confusion matrix and statistics output and include it in your submission. For the drug consumption dataset and based on your own code, which one is better, KNN or kknn? Name your R code *hw4.R*.

I found no real differences between the two models however knn did process much faster.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Submission:**

Include answers to Problem 1, Problem 2, Problem 3, and Problem 4 in *hw4.docx* or *hw4.pdf*. Combine this file and *hw4.R* into a single archive file and name it *LastName\_FirstName*\_*HW4.EXT*. Here, “*EXT*” is an appropriate file extension (e.g., *zip* or *rar*).