Stars of the KNN Universe

A Teknowledge Activity by Christopher George

1.0 Data Described

The following is a subset of the data in provided. The **first** number correlates to the age of the star at its death. The **second** number correlates to the temperature of the star at its death. The True/ False refers to whether or not the star hit Supernova. (True means it did)

[

[5.290224332009728, 105.01630249070735, False],

[7.39568745662327, 91.20614517529071, False],

[3.0747743514096837, 105.01883055135922, True]

]

The KNN functions we will write will be able to take all of our data, and determine how accurate our training set is at determining whether a star will Supernova.

We will take all of our data, and pop the first half of the data off, to use as our test set.

2.0 Write Up

The purpose of this lab is to gain an understanding of our implementation of the KNN algorithm. The majority of the algorithm is given, so please take time to understand it after you have completed the code you do have to write.

Additionally, look at how the data was collected in the file dataCollection.py. After you are done the challenge, please feel free to modify how the data collection works, run the program, copy the data printed out. Paste it into data.txt, then run your KNN again to see how it performs. Note, modifying the order of elements in data would mean you would also have to modify your algorithm as those locations are currently hard-coded.

3.0 Challenge

Complete the challenges.

|  |
| --- |
| # Challenge 0: Run the code and print out the testSet and trainingSet a few  # times. Notice how the data is set up:  # [[ageOfStarAtDeath, tempOfStarAtDeath, superNova?]....]  # Remember: The goal of this function is to test how accurate our algorithm is  # at prediciting a supernova or not.  # Challenge 1: The first step is to complete the function getNeighbors.  # We are going to calculate the distance between the current test instance  # and every point in the training instance, put them in a list, then sort them.  # Then we will pick the top k elements from the list.  # Challenge 2: The next step is to complete the getLabel function. In this  # function we are iterating through the returned neighbors, and getting the  # true or false label and adding that label to our dictionaries.  # Challenge 3: The last step is to iterate through the test set in the knnMain  # function. First gather your neighbors using the getNeighbors helper function.  # Then, get the label, using the getLabel helper function. Then check whether  # the label retrieved matches the actual label and increment correct  # accordingly. |

4.0 Bonus Challenges (if time allows)

* Modify how data is collected, and then subsequently modify any function that uses that data accordingly.
* Practice writing all of these functions from scratch using different methods to perform the same task.
* We will be writing a similar but not exactly the same functions next class, so think about ways that we could add onto this. For example, what if each datapoint not only had a label, but also a weight as to how strong it was at predicting.
* Modify the distance formula to follow the euclidean distance of any dimensional input so as to generalize the formula and thus allow more data to be used to more accurately predict your output.

5.0 Comprehension Questions

|  |
| --- |
| Think about them and/or discuss them with a friend. We will discuss them at the end of class.   * Thinking about this general algorithm, in what instances will it fail to deliver a reliable result? What type of data would be have? * Try to think about how many dimensions this algorithm could handle in calculating the distance two the k closest points. * How do you think KNN can be used in conjunction with another ML algorithm we already learned about? * Unlike most ML algorithms that use real-world data, this algorithm uses generated data. Therefore, we KNOW that there is a function to best predict whether a star hit supernova or not (namely, the same function we used to generate the label on the data, although it won’t be perfect because of randomness). Therefore, we are basically solving a function approximation problem -- can we approximate the function used to generate the data by using a KNN-based function? Given this, is it possible for KNN to perfectly approximate the generating function? Why or why not? Is it possible for KNN to be more accurate on the test data than the actual classifier (which is basically the same as the function we used to generate the data)? If so, what does that mean the KNN learned? * As you can see, we found the k nearest neighbors by storing the distance to every point in a list. This requires a lot of storage -- we are storing every single training point and every single distance, even though we only really have to store k of them. Can you think of another way to implement that part of the algorithm which takes less storage? If you have time, feel free to modify it to the less-storage version. |

