Allan Pedin

CPSC-595

Samuel Henry

2/15/2020

Assignment 1

Introduction

This paper will discuss K-Nearest Neighbors classification as well as its application to the Iris Data Set. K-Nearest Neighbors is a simpler machine learning algorithm in which a data point is classified based on what its “nearest neighbors”, out of a training set, are classified as. In order to determine which training points are nearest to our point we need some sort of distance function. Common distance functions include:

- Euclidean: $d(a,b) = \sqrt{\sum_{i=1}^{d}(a_i - b_i)^2}$
- Manhattan: $d(a,b) = \sum_{i=1}^{d}|a_i = b_i|$
- Cosine: $d(a,b) = \frac{\sum_{i=1}^{d}(a_i * b_I)}{\sqrt{\sum_{I=1}^{d} a_i{}^2} * \sqrt{\sum_{I=1}^{d} b_i{}^2}}$

To better understand this let's look at an example. For training data we will generate 2 classes of (x,y) points:

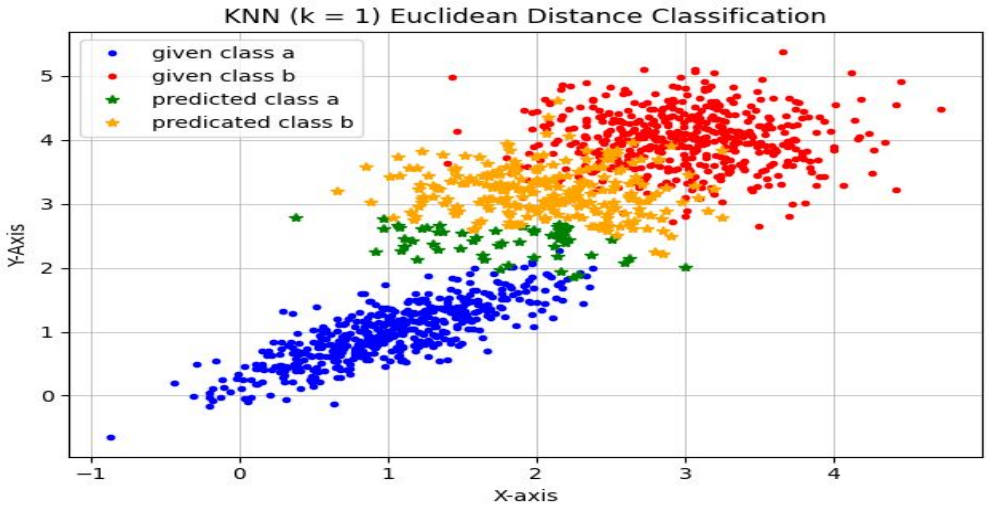
- Class A: 500 randomly generated normally distributed points
    - With a mean of (1,1)
    - And a covariance of $\begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.2 \end{bmatrix}$
- Class B: 500 randomly generated normally distributed points

- With a mean of (3,4)
- And covariance of $\begin{bmatrix} 0.3 & 0.0 \\ 0.0 & 0.2 \end{bmatrix}$

These 2 classes will serve as our training data. And lastly we will generate a third set of (x,y) points. Each point in this set will classify as either A or B, based on their “nearest neighbors”.

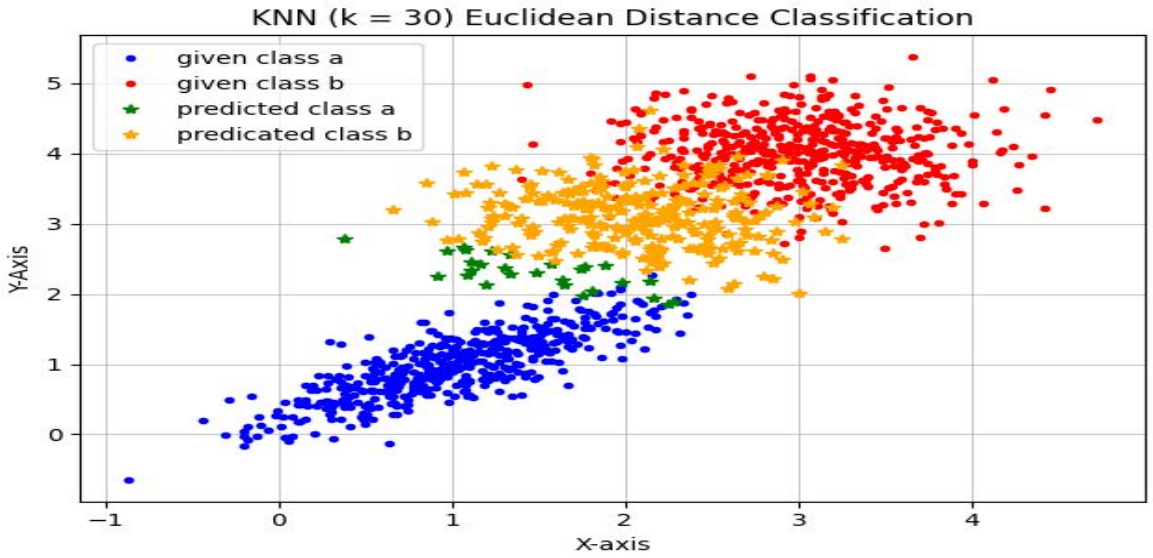
- Set to be classified: 300 randomly generated normally distributed points
  - With a mean of (3,4)
  - And covariance of $\begin{bmatrix} 0.3 & 0.0 \\ 0.0 & 0.2 \end{bmatrix}$

Now we are ready to classify our third set. Let's classify this third set using K-nearest Neighbors, with a K equal to 1 and our distance function being euclidean distance (Fancy math term for the length of a straight line between 2 points). What this is essentially doing is classifying each point by the class of the point that it is nearest to. Lets see what happens when we classify all of our third set using this method.



In this graphic we can see a pretty clear divide between the 2 classifications. With a k=1 our results are fairly good, although they are susceptible to noise, as points closest to just 1 noisy point will be classified as that point.

Now lets try with a K equal to 30. What this is doing is finding the 30 closest points to a point we want to classify and having a vote of sorts to determine the classification of our point.



As you can see with a K equal to 30 we get a similar divide between our points. Although this method is less susceptible to noise, if we increase K too much, our closest points may include points from the wrong class. A proper value K is one that is large enough to discern between what is noise and what is not, and small enough to only include relevant points.

Application of KNN to the Iris Data Set

The Iris data set is a famous dataset of Iris Plants and their features. Each plant in the set has 4 features (sepal length, sepal width, petal length, petal width) and a classification of either Iris Setosa, Iris Versicolour, or Iris Virginica. Interestingly 1 of

these 3 classes is linearly separable from the others, meaning that a clear line can be drawn through the data such that this class is entirely on one side of the line, and every other class is entirely on the other side (since this data is 4 dimensional it would technically be a solid, but a line is easier to visualize). Effectively this means that this set can be easily classified.

For this experiment I developed a K-nearest neighbor classifier for this dataset. In order to test my classifier, I reserved 20% of the data as a test set, which will only be used to get final results. I then used 5-Fold cross validation on the remaining 80% of the data to tune my model. My tuning parameters were K, the amount of nearest neighbors with which the data would be classified, and distance function. The distance functions I used included: euclidean distance, cosine distance, and mahalanobis distance. Euclidean distance is a good starting point but is susceptible to scale issues, cosine solves the scale issues and achieves good results, and mahalanobis distance is resistant to scaling while also accounting for correlations in the dataset.

I tried Several values of K, in combination with each distance function. In order to evaluate the performance of my model I calculated per-class and macro-averaged: Accuracy, Precision, Recall, and F1 Score for my model. After many iterations of different values of K and different distance functions I found that my best performance was achieved with K = 7 and the Cosine distance function. With this configuration on the test set my model achieved:

- Macro Averaged Accuracy: 0.9777777777777779
- Macro Averaged Precision: 0.9583333333333334
- Macro Averaged Recall: 0.9761904761904763

- Macro Averaged F1score: 0.9654320987654321

On my test set, as well as a perfect classification under all metrics of the Iris-Setosa class. My prediction as to why the cosine function performed the best is that it accounts for scale without overfitting the data.

This experiment found that a KNN classification of the Iris data set can achieve high results using a K equal to 7 and the cosine distance function, as well as find the linear separation between the Iris Setosa class and the other classes. While these hyper parameters proved great for this data set on other data sets with different properties other hyper parameters will likely perform better.