K-Nearest Neighbors Classifier

Machine Learning – Project Report 1:

Muhammad Saad Kaleem, Hamza Abugazia and Mohammad Hussein Shakerpour

Qatar University

***Abstract* —this report discusses the implementation of k-nearest neighbor and noise reduction technique, Tomek links algorithms on a set of synthetic generated random data on the 2D plane. The implementation of the algorithms was done in Python 3.0 and the libraries: NumPy and Matplotlib were utilized for array-processing and generating graphs respectively.**

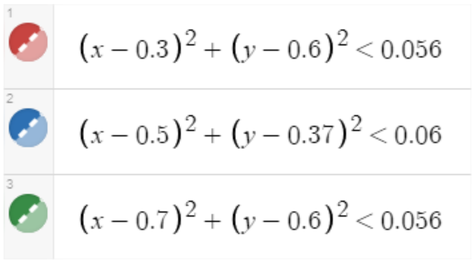
I. INTRODUCTION

This project simulates a supervised learning problem through artificially generated data. The data has been assumed to have two meaningful features consisting of positive and negative classes (Binary classification) with a total of 1000 data points. The data set was plotted on the two dimensional (2-D) plane. The K-Nearest-Neighbor (KNN) machine learning algorithm was used and its accuracy was verified experimentally for varying levels of noises. Moreover, observations on the test set accuracy were also made after applying a noise removal technique, known as Tomek links on the training set.

We then split the data randomly into a training set and a testing set with proportions (70%:30%) respectively.

II. DATA GENERATION

The data was assumed to be fully inside the 1st quadrant $(0,0)$ to $(1,1)$ and the shape was generated through limiting the radius of a circle, and three circles were overlapped



1 $(x-0.3)^2+(y-0.6)^2<0.056$

2 $(x-0.5)^2+(y-0.37)^2<0.06$

3 $(x-0.7)^2+(y-0.6)^2<0.056$

Figure 1: Equation of circle(s) with a limited radius

The graphical 2-D representation of the equations: The positive examples, would constitute the three shaded areas of the circles.

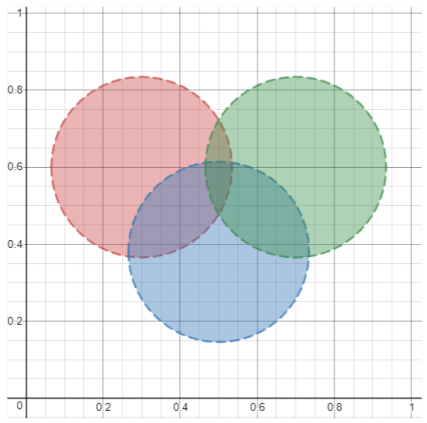


Figure 2: Shaded circles constituting about 50% of the total area.

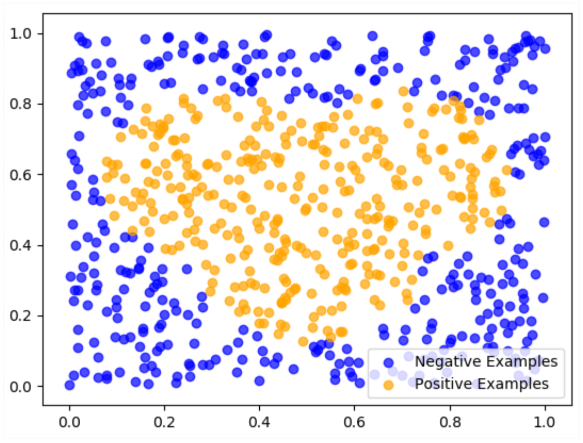


Figure 3: Example of a training set, corresponding to the above equations with 0% noise. Blue dots represent the negative examples, while the orange dots represent the positive examples. The "heart" shape corresponds to approximately half of the total window.

As for introducing the noise in the data set, random examples in order of (10%, 20%, 30%, 40%, and 50%) were chosen from the existing original training set, and their class labels were switched.

III. KNN & TOMEK LINKS ALGORITHMS

The distance metric used to implement the KNN algorithm is Euclidean Distance.

Firstly, these are the steps used to get a sorted array of distances, corresponding to each data point in the training set.

```
Algorithm 1: (Retrieving K-NN)
1. For all the training data (T), and a test example (x)
2. Calculate the distances between T and x
3. Get the index sorted (argsort) array of distances in descending order.
4. Return the K nearest neighbors (the first element in the sorted distances [Step 3] corresponds to the 1-NN)
```

Secondly, to get the predictions on the testing set for an arbitrary value of K neighbors, we incorporated the above Algorithm (1) in the following Algorithm (2):

```
Algorithm 2: (for Predictions)
1. For an arbitrary value of K, and for each data point in the testing set (z)
2. Find the K-nearest neighbors utilizing the Algorithm 1.
3. Using a majority-voting system, classify the testing point (z) as either positive or negative
4. Repeat Steps 1-3 for every data point in the testing set.
```

As for the accuracy, we used this algorithm:

```
Algorithm 3: (for Accuracy)
1. Retrieving the NumPy array of predictions, we compared the predicted labels with the actual, known labels of the testing set.
2. Calculate the accuracy by getting the number of correctly predicted data points and divide it by the total number of data points in the testing set.
```

For the Tomek links (noise removal), we used the following algorithm:

```
Algorithm 4: (for Tomek Links)
1. Let x be a data point in the training set, and let y be another data point that is it's 1-NN.
2. If the 1-NN of y, is also x AND x and y have different labels.
3. Classify them as a Tomek link.
4. Else if, x and y have same labels – they do not form a Tomek link.
5. Repeat Steps 1 to 4 for x = x+1 for x <= size of training set (N)
```

IV. RESULTS & ANALYSIS

We decided to visualize the decision boundaries to get a better idea of the overall performance of the KNN classifier. We did that by feeding the classifier a set of synthetic data with a size of 10,000 examples. We then classified it using our algorithm and plotted it to observe, how the classifier would perform (roughly) across all values of x and y features. The algorithm performed as expected (with and without noise).

Firstly, we experimented with different values of *K* (1, 3, 7, 15 and 31) before introducing any noise to see the effect on the "smoothness" of the decision boundaries

Then, we observed that higher values of *K* made the classifier less susceptible to "over-fit" anomalies in the data and resulted in a smoother decision boundary, as shown in this small animation linked **here**.

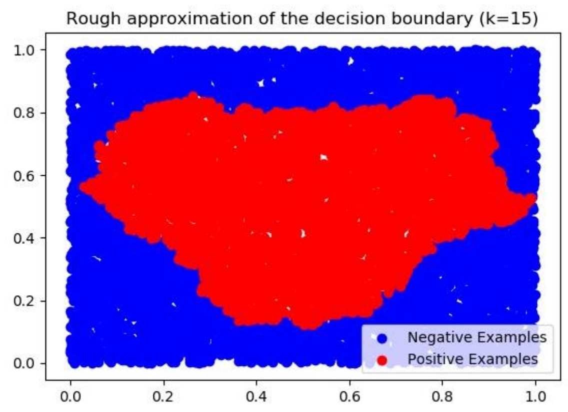


Figure 4: K=15, an example of the decision boundary (without noise) **Extended GIF is linked here (again).**

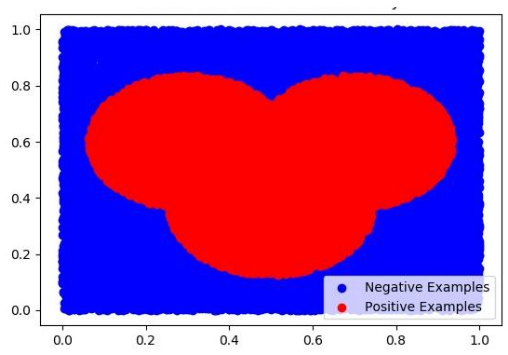


Figure 5: The actual desired decision boundary.

We also observed an issue with the side-edges with higher values of K but we figured that is due to the lack of data around these points.

We studied the accuracy, with different values of K and we noticed that the effect was insignificant (before introducing noise) and was heavily dependent on the randomness of the data.

After introducing noise (10 %, 20 %...) we noticed that the effect of changing K was much more significant as shown in the figure:

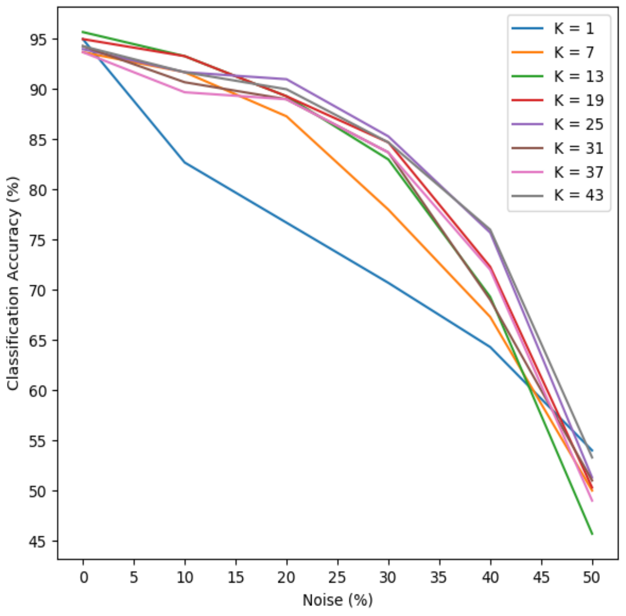


Figure 4: Accuracy vs. Noise across different values of K

We can easily observe that the classifier is affected by noise when K is very low (since it "over-fits" the noise), compared to higher values of K but it's only up to a certain point.

If we significantly increase the values of K past that point, we start to notice signs of under-fitting (as the accuracy becomes considerably lower). This is represented by the Figure 7 on the next page.

The data shows that the best optimal value for K is between 13 to 25.

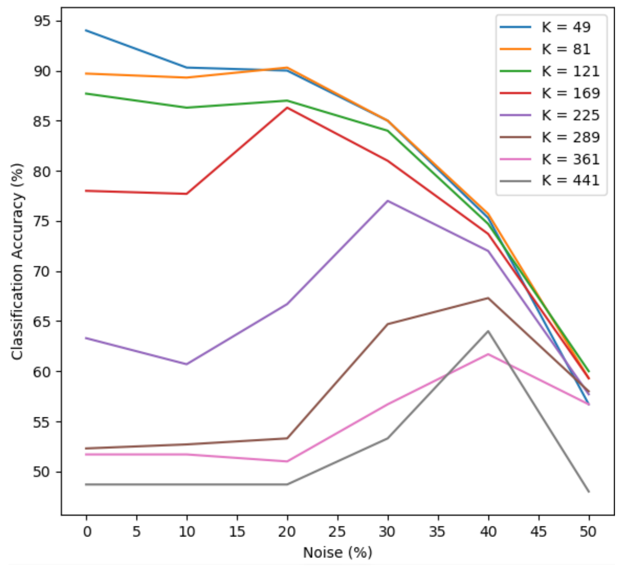


Figure 5: Accuracy vs. Noise (K= 49, 81, 121…)

Such high values of K, affected the decision boundaries along the transition areas of the class labels from positive to negative, especially along the side-edges and resulted in misclassifications across that region.

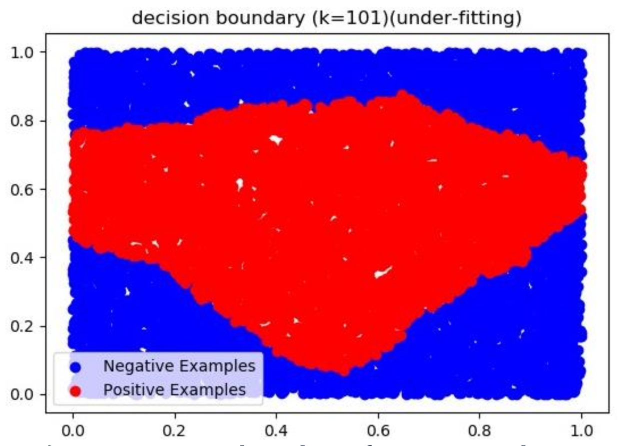


Figure 8: Decision boundaries for K=101, with 0% noise results in under-fitting.

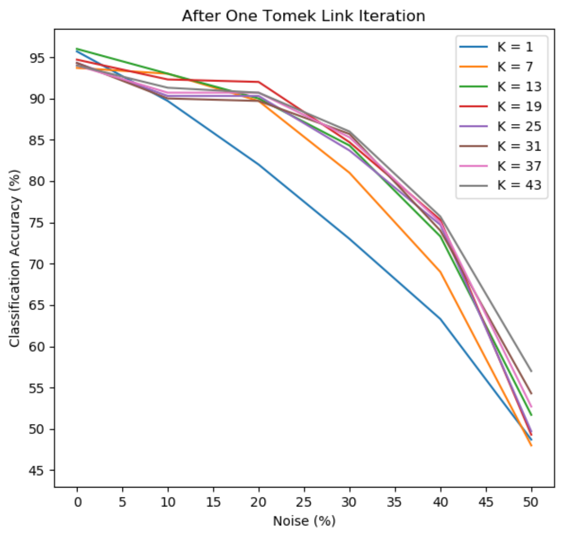


Figure 9: Accuracy vs. Noise (After 1 Tomek Link Iteration)

As for Tomek Links, the effects on the accuracies were significantly less than expected – However, we did observe a slight upward shift of the overall accuracies indicated on the graph.

Furthermore, after the 2nd iteration of Tomek links removal, we did not observe a significant change in the overall accuracies.

We also inspected the effect of Tomek link removal on the decision boundaries and noticed that it became less affected by noisy examples. The greatest effect of removal of the noisy examples was noticeable at K = 1, and lower values of K, in general.

The effect of Tomek links on decision boundaries is shown in this GIF animations.

- 10% noise.
- 20% noise.

V. SUGGESTIONS & CONCLUSION

To improve the performance of the classifier, a good choice would be to enhance the evaluation criteria of it. One way to do that is through cross-validation, as it eliminates the effect of “randomness” of the testing set and makes sure that the algorithm, performs well across all examples.

We expect this to show higher correlation between the hyper-parameters and the overall performance of the classifier. For example: if the testing set had a single training-testing split, the evaluation of the performance would heavily depend on the number of examples along the transition areas of class labels.

Therefore, cross-validation ensures that the evaluation criteria is reliable.