Nathaniel Weissinger

ECE 456 – Pattern Recognition and Machine Learning

Project #1 – Moments

February 5th, 2022

INTRODUCTION

The K-Nearest Neighbor (KNN) classifier is an algorithm that receives a dataset with several moment features (variable subsets), then calculates the distances between every point in the dataset. Depending on the type of classifier (1-NN, 3-NN, or 5-NN), the algorithm determines the lowest one, three, or five distances, then compares them to each other to determine a classification.

DATASET CALCULATIONS

The dataset used for the classifier was a set of ten letters: ‘a’, ‘c’, ‘e’, ‘m’, ‘n’, ‘o’, ‘r’, ‘s’, ‘x’, and ‘z’. Each of these classifications contained eight features, defined as moment calculations based on pixel weight. Depending on how the pixels were distributed around certain axes, the moment identification would classify certain letters as distinct, allowing for further distinguishment using preset training data.

Before information could be inferred from subsequent evaluations, a set of training data needed to be analyzed to determine the preset configuration of moments for each letter. All letters were given 10 sets of data points, totaling to one hundred data points, each point established with eight moment features. A class called Data was created in order to read in the text file containing the training data, then parse the data into a matrix. The Data class also performed simple calculations, such as the mean, the root-mean-squared (RMS), normalization (dividing each of the data points by the RMS), and returned variables such as letter arrays or index values.

Two sets of evaluation data were analyzed; Data objects were created for both the training and the evaluation data, and once the sets of data were normalized, the two sets of evaluation data were sent to the KNN classifiers.

DISTANCE ALGORITHM

The KNN class was created to perform both the simple distance calculation, and the three types of KNN algorithms: NN1, NN3, and NN5. One KNN object was created for each of the

two evaluation sets, and through those objects, the training data and the evaluation data were passed through to begin the nearest neighbor analysis.

The KNN algorithm chosen was based on the Euclidean distance algorithm, which subtracted each moment feature of an evaluation data point with the same moment feature of the training data. The resulting difference was then squared, summed to the other seven differences, then square-rooted to determine the overall distance between the two points in the separate sets. Through this method, one hundred distances are obtained per each evaluation data point, since there are one hundred points within the data set that need comparing. The first evaluation data set, *eval1dat*, had two hundred data points, and thus, twenty thousand distances were calculated. The second evaluation data set, *eval2dat*, contained one hundred data points, resulting in ten thousand distance calculations.

As each distance is calculated per evaluation data point, the smallest distance indices are saved for comparison. Within the NN1 classifier, if any distance is smaller than the previously saved distance, the originally saved index is overwritten, thus only keeping the smallest distance saved. The classification of the evaluation data point is then concurrently determined based on which training data point the evaluation data is closest to.

Using the NN3 classifier, the lowest three distances' indices are saved, and every new distance calculation is compared with the other three. If any saved instance is found to have a larger distance than the newest distance calculation, the new distance measurement index is saved, shifting the array so that the smallest distance is saved as *index[0]*, the second smallest is saved as *index[1]*, and the third smallest is saved as *index[2]*. The NN5 classifier works exactly the same, except that the five smallest indices are saved.

KNN CLASSIFICATION

After all distances were calculated and the indices of the smallest were saved, the classifier analyzed how many distances were classified as the same letter. As stated previously, the NN1 classifier only saved the single shortest distance, so after calculating al the distances, there would have been only one option for the data point to be classified as.

In the NN3 classifier, the evaluation data points were classified as the letters that matched two or three of the training data point letters out of the three saved distances. If, however, the three closest distances were three distances corresponding to three separate training data letters, the classifier picked the training data letter with the smallest distance. This is the only instance where the classifier would have had a voting tie.

In the NN5 classifier, if three, four, or five of the saved indices corresponded to the same training data letter, then the evaluation data point would be classified as that letter. However, if there were two pairs of two different letters out of the five saved (resulting in a voting tie between two pairs), the two smallest distances pair's letter was chosen as the data point classification. If only two of the points corresponded to the same letter and no other letters were the same, the evaluation data point would be classified as that pair's letter, regardless of the

distance index. Then, if all the distance indices corresponded to separate letters, the lowest index (smallest distance) would be chosen as the classification to break the voting tie.

RESULTS

Once each classifier returned an array containing the letter classifications, the data was sent through a formatting method that created a confusion matrix. This matrix visualized which letters were correctly classified and which letters the classifier 'confused' for another. Additionally, the confusion matrix output defines the number of errors, and outputs the total error percentage. Lastly, the output is both displayed on the system console, and is also converted into an output file labeled 'output.txt' (located in the *src* folder). Figures 1-6 show the output of both the console log and the output file.

KNN1: eval1

Decision

	a	c	e	m	n	o	r	s	x	z	error I
a	20	0	0	0	0	0	0	0	0	0	0
c	0	20	0	0	0	0	0	0	0	0	0
e	0	0	20	0	0	0	0	0	0	0	0
m	0	0	0	20	0	0	0	0	0	0	0
n	0	0	0	0	19	0	0	0	1	0	1
o	0	0	0	0	0	20	0	0	0	0	0
r	0	0	0	0	0	0	20	0	0	0	0
s	0	0	0	0	0	0	0	20	0	0	0
x	0	0	0	0	1	0	0	0	19	0	1
z	0	0	0	0	0	0	0	0	0	20	0
error II:	0	0	0	0	1	0	0	0	1	0	2

Classification Results: 99.0% correct, and 1.0% error

Figure 1: 1-NN Classification of Evaluation 1 Data

KNN1: eval2

Decision

	a	c	e	m	n	o	r	s	x	z	error I
a	8	0	0	0	0	0	0	0	0	2	2
c	0	2	8	0	0	0	0	0	0	0	8
e	0	0	10	0	0	0	0	0	0	0	0
m	0	0	0	10	0	0	0	0	0	0	0
n	0	0	0	0	7	0	0	0	3	0	3
o	0	0	0	0	0	4	0	6	0	0	6
r	0	0	0	0	0	0	10	0	0	0	0
s	0	0	0	0	0	0	0	10	0	0	0
x	0	0	1	0	0	4	0	1	0	4	10
z	0	0	0	0	0	1	0	7	0	2	8
error II:	0	0	9	0	0	5	0	14	3	6	37

Classification Results: 63.0% correct, and 37.0% error

Figure 2: 1-NN Classification of Evaluation 2 Data

KNN3: eval1

Decision

	a	c	e	m	n	o	r	s	x	z	error I
a	20	0	0	0	0	0	0	0	0	0	0
c	0	20	0	0	0	0	0	0	0	0	0
e	0	0	20	0	0	0	0	0	0	0	0
m	0	0	0	20	0	0	0	0	0	0	0
n	0	0	0	0	19	0	0	0	1	0	1
o	0	0	0	0	0	20	0	0	0	0	0
r	0	0	0	0	0	0	20	0	0	0	0
s	0	0	0	0	0	0	0	20	0	0	0
x	0	0	0	0	1	0	0	0	19	0	1
z	0	0	0	0	0	0	0	0	0	20	0
error II:	0	0	0	0	1	0	0	0	1	0	2

Classification Results: 99.0% correct, and 1.0% error

Figure 3: 3-NN Classification of Evaluation 1 Data

KNN3: eval2

Decision

	a	c	e	m	n	o	r	s	x	z	error I
a	8	0	0	0	0	0	0	0	0	2	2
c	0	2	8	0	0	0	0	0	0	0	8
e	0	0	10	0	0	0	0	0	0	0	0
m	0	0	0	10	0	0	0	0	0	0	0
n	0	0	0	0	7	0	0	0	3	0	3
o	0	0	0	0	0	4	0	6	0	0	6
r	0	0	0	0	0	0	10	0	0	0	0
s	0	0	0	0	0	0	0	10	0	0	0
x	0	0	1	0	0	3	0	1	0	5	10
z	0	0	0	0	0	1	0	7	0	2	8
error II:	0	0	9	0	0	4	0	14	3	7	37

Classification Results: 63.0% correct, and 37.0% error

Figure 4: 3-NN Classification of Evaluation 2 Data

KNN5: eval2

Decision

	a	c	e	m	n	o	r	s	x	z	error I
a	7	0	0	0	0	0	0	0	0	3	3
c	0	3	7	0	0	0	0	0	0	0	7
e	0	0	10	0	0	0	0	0	0	0	0
m	0	0	0	10	0	0	0	0	0	0	0
n	0	0	0	0	6	0	0	0	4	0	4
o	0	0	0	0	0	2	0	8	0	0	8
r	0	0	0	0	0	0	10	0	0	0	0
s	0	0	0	0	0	0	0	10	0	0	0
x	0	0	1	0	0	6	0	2	0	1	10
z	0	0	0	0	0	0	0	6	0	4	6
error II:	0	0	8	0	0	6	0	16	4	4	38

Classification Results: 62.0% correct, and 38.0% error

Figure 5: 5-NN Classification of Evaluation 2 Data

Table 1: Summarized Data

Classifier	Evaluation Data	Distance Method	Error (%)
1-NN	eval1dat	Euclidean	1%
1-NN	eval2dat	Euclidean	37%
3-NN	eval1dat	Euclidean	1%
3-NN	eval2dat	Euclidean	37%
5-NN	eval2dat	Euclidean	38%

DISCUSSION

In Figures 1 and 3, the data clearly showed that the set of evaluation data closely resembled the training data, and was ideal for the implementation of the Euclidean distance method. However, in Figures 2, 4, and 5, the second set of evaluation data proved suboptimal, as the percent of correct classification only reached 63% at its highest (shown in Figures 2 and 4). This heavily implies that the Euclidean distance method was not an ideal classification method for the given data. Additionally, the number of nearest neighbors the classifier identified did not improve the performance, even though some errors should have been corrected through the method's further speculation. Figure 5, having used the NN5 classifier, showed a higher total error percentage than either the NN1 or NN3 classifiers.

Additionally, there were surprisingly few voting ties within the NN5. Most calculations showed either three or four out of the five distances as being correlated to the same training data letter (thus having no voting ties), and on much rarer circumstances, the algorithm would ties resulting from two pairs of data points. The letters that would be most often appearing within these decisions were between 'x' and 'n', but on occasion, the letters 'o' and 's' would appear. The letter 'c' only appeared once within these voting ties.

Although it's not certain, the issue with the subpar performance was likely due to the Euclidean distance method's feature averaging of the variances. Although there were several different moment features that could have been used for individual identification, the distance method took the differences between each, squared them, then simply added them together before square-rooting them. In other words, it turned an eight-dimensional vector into a scalar. This method produces no individuality amongst the features, and as with several moments, many weights overlap with other letters, giving nearly identical results when compared to other features. Additionally, a single outlier within the data can skew the results heavily, pushing the distance measurements more towards other letter classifications.

Other methods might have worked better for the second evaluation data set's classification, but the other distance methods have the same faulty underlying principle of turning the vectors into scalars. A more accurate way to classify the data would have been by utilizing the individual features, and developing data ranges so that any newly identified data

point would not be classified as a letter unless it appeared within a certain range. Certain features could even be compared with other features, and the relationships between multiple pairs would be enough to classify a letter properly. These feature comparisons could even be used in conjunction with the distance method to further the accuracy of the classification.

CONCLUSION

Although there is no perfect way to train a machine to identify letters, this method proved to be far from perfect. Even though the first set of evaluation data was identified almost perfectly correct, the second set's high error proved that the first data set was only an idealized selection. In order to further improve upon the accuracy of the classifier, either a different distance method would need to be used, or there would need to be more ways to compare and identify features of the letter data, aside from, or in addition to, the distance method. Additionally, voting ties between two pairs of data in the NN5 classifier could have been improved, yet that would only improve the performance of the classification by a single percent or two. The only real way to truly improve the classifier would be to change or improve the distance algorithm while simultaneously adding additional parameters to determine feature ranges.