Instance-Based Learning

Ian Kessler IAN.KESSLER@STUDENT.MONTANA.EDU
Department of Computer Science
Montana State University
Bozeman, MT 59717, USA

Ethan Skelton ETHAN.SKELTON@STUDENT.MONTANA.EDU
Department of Computer Science
Montana State University
Bozeman, MT 59717, USA

Editor: Ian Kessler and Ethan Skelton

Abstract

In this project, we implemented a few different algorithms that train a model and try and predict the classes for a given data point. We are using k nearest neighbor, edited k nearest neighbor, and k means in order to try and predict these values. First we pull out a tuning set from our original data set, that will be used to tune each of the algorithms later on. Then the training sets were assigned using 10-fold cross validations. Our data sets comprise of breast cancer data, glass data, hardware data, forestfire data, abalone data and soybeans data which each have there defined labels for their features.

Half of our data was classification based where we had classes to compare the data to. These data sets were breastcancer, glass, and soybeans, where the other classes are regression based: forestfire, abalone, and hardware. For the regression based models we had to deal with the data in a bit of a different way. We had to create algorithms that were able to deal with continuous values. It is also necessary to make sure that our data sets with categorical values were turned into quantitative values so we can find a distance later on. We dealt with these categorical values using one hot coding.

Keywords: k nearest neighbor, k means

1. Problem Statement

For any of the size given data sets, we could train a model to predict the class of any example from the respective data set. We are trying to predict breast cancer, type of glass, type of soybean, age of abalone, performance of hardware, and area of a fire.

One of our models is based off of k nearest neighbor in order to predict these values. K nearest neighbor tries to predict the class of a data point based on the kth nearest other data points and their classes or values. We also use an edited nearest neighbor which is similar to k-nearest neighbor, but we also remove data points in the data set until we get a worse performance. Finally, we also implemented k-means clustering which we used as a reduced data set for k-NN.

Our goal is to figure out which algorithm will be have the best performance. We predict that the k-means will do the best with the regression data sets since it ignores classes

while deciding on the clusters. We also think that edited NN will preform the best for the classification data sets.

1.1 K-Nearest Neighbor Classifier

k-nearest neighbor classifier predicts a value of a data point based on the k closest points. It looks at the k closest points to the data point and determines which class it belongs in based on how close it is to each of those points.

1.2 Edited K-Nearest Neighbor Classifier

Edited k-nearest neighbor classifier works similar to the k-nearest neighbor where it looks at the k closest points and determines the which class the data point belongs in; however, it also is removing data points until the performance of the algorithm start to falter.

1.3 K-Means Cluster Classifier

k-means classifier places each of the points into clusters of similar data points. Then it uses k-nearest neighbor in order to predict the value of the data point.

2. Experimental Approach and Program Design

2.1 Data Preprocessing

2.1.1 Data Retrieval and Data Description

We downloaded the data sets as csv files from UCI Machine Learning repository at `https://archive.ics.uci.edu/ml/`. Some of these data sets did not have the attribute names or the class names labeled to the columns, and others did. We had to create a way to deal with both of these situations. For the situation where we did not have labeled columns we have a text file to know what columns names to append to each csv file. For the case where we already had the column names, we left the csv file as is.

2.1.2 Replace Missing Values

Some of the data sets had missing or unknown values denoted by '?'. For each of these data sets, we replaced each '?' with the same value that appeared frequently within the data set. In particular, in the "Breast Cancer" data set, we replaced each '?' with '3'. Luckily there was only the 'Breast Cancer' data set that was missing data, but if we needed to we could have changed out and other datasets '¿ value with a value of our choice.

2.1.3 Categorical Values

There were some data sets that had categorical values that do not work well with the algorithms that we are trying to implement. We decided to deal with these values at the beginning when we read the data set in. If we found values that were categorical then we used one hot coding in order to make them into quantitative values.

2.1.4 Data Sets

Half of our data sets were classification based: breastcancer, soybeans, and glass. For these data sets we defined that they were classification based which meant they had a column that was dedicated to the class that each data point belongs to. This is very helpful for training data because we each data point has a class associated with it so we can easily determine which class the data point we are trying to figure out belongs in through k-NN or k-means. The other half of our data sets were regression based: abalone, forestfires, and hardware. Regression sets do not have a specific class for each data point. Instead each data point has a continuous value that it associated with it. We delt with these data sets differently. For example in k-nearest neighbors we have a different ways for determining what "class" the data point belongs to. For classification we just determine the most classes that are near the data point and assign it to that class. However, for the regression data set we have to implement a Gaussian kernel to make our prediction for the value the point is. We use the Gaussian kernel to apply weights to each of the values for the nearest data points which then allows us to determine the value that the data point we are trying to predict will be

2.2 Experimental Design

2.2.1 10-Fold Cross Validation

To perform 10-Fold Cross Validation on a data set, we first partitioned the data set into 10 subsets of almost equal size. Using that partition, we tested our classification algorithm on each part of the partition by a model trained on the rest of the data.

2.2.2 Distance Function

The distance function determined by the euclidean distance where we square the difference of corresponding values of two data points, then sum all those squared values and take the square root of that sum which gives us our distance. Which looks like

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

2.2.3 Tuning K-NN

Tuning k nearest neighbor requires us to tune for k which is the amount of closest data points that the algorithm will check in order to determine what class the data point we are looking at belongs in. For the regression data sets we have to tune sigma which is our bandwidth which allows us to make predictions on a given data point based on the k closest points. Then we have to tune for the epsilon in the edited k-NN and the k for the amount of clusters in k-means.

2.2.4 Putting It All Together

For each of the six data sets, we start by splitting the data set into 10% tuning, and 90% rest. With the rest of the data set we preform 10-fold partitions and then we will apply k nearest neighbor, edited nearest neighbor and k-means on each of the data sets. Each

of these data sets will create predictions based on the tuning set that will be used to tune certain values for the data algorithms. This will allow each of the algorithms to make a prediction of a data point based on the training that it has had.

3. Results

3.1 Explanation of Error Data Frames

For each of the three estimators (K-Nearest Neighbor, Edited K-Nearest Neighbor, and K-Means Cluster) and for each of the six data sets, we created an error data frame. For an error data frame, each row is uniquely identified by the fold of the crossvalidation and the set of hyperparameters used for the prediction. Furthermore, each row has a recorded error for the set of predictions on the tuning set by the estimator trained by the training set of the given fold and the given set of hyperparameters.

3.1.1 Error Function

Error was calculated by one of two ways. For a given data set, let n be the number of examples in our tuning set. For the classification sets, our error function was derived by subtracting the average of the zero one loss and the P Macro by 1. More precisely, for a given classification set with the set of classes denoted as C and the i^{th} predicted class in our tuning set denoted as $c_i \in C$, then our error function $E : C^n \to [0, 1]$ was computed by

$$E(c_1, c_2, ..., c_n) = 1 - \frac{1}{2}(\text{ZeroOneLoss}(c_1, ..., c_n) + \text{PMacro}(c_1, ..., c_n))$$

where the "ZeroOneLoss" function calculates the fraction of classes predicted correctly and "PMacro" calculates the p macro of the derived confusion matrix from a given set of predicted classes. For the regression sets, our error function used was the mean squared error function. More precisely, with the i^{th} predicted target value and the i^{th} actual target value in our tuning set denoted as $p_i \in \mathbb{R}$ and $a_i\mathbb{R}$, respectively, our error function $E : \mathbb{R}^n \to \mathbb{R}$ was computed by

$$E(p_1, p_2, ..., p_n) = \frac{\Sigma_{i=1}^{n}(p_i - a_i)^2}{n}$$

.

3.1.2 Hyperparameter Space

For our classification sets, we tuned over the number of neighbors or the number of clusters, denoted by k. The k values we tuned over for each data set and each estimator was $[4, 5, 6, 7, 8]$. For the regression sets, we also tuned over the bandwidth σ, which was in the radial basis function. The σ values we tuner over were the integers from 4 to 13 (inclusive). For the regression sets and the edited nearest neighbor estimator, we also tuned over ϵ, the maximum tolerated distance between the predicted and actual target value of a point in the training set to be considered correctly classified by the other training points. The ϵ values we tuned over were in $[2, 3, 4, 5, 6]$.

3.1.3 Limits on Results (Due to Time)

Because the edited nearest neighbor had a much longer runtime and computational complexity, due to it being recursive, we made two types of limits on our analysis due to time. First, we limited the number of σ values we tuned over to just two values. Looking at the basic nearest neighbor results, we saw that $\sigma = 4, 13$ were the best values, so we tuned σ with just these values on the edited nearest neighbor estimator. Another restricted we implemented was taking only a subset of some of the data sets for error and analysis. For the BreastCancer set, we only used the first 400 examples for analysis. For the Hardware, ForestFires, and Abalone data sets, we only used the first 100 examples from each.

3.2 K-Nearest Neighbor Error Dataframes

	Fold	k	sigma	epsilon	Error
0	0	4	NaN	NaN	0.0
1	0	5	NaN	NaN	0.0
2	0	6	NaN	NaN	0.0
3	0	7	NaN	NaN	0.0
4	0	8	NaN	NaN	0.0

Table 1: SoyBean

	Fold	k	sigma	epsilon	Error
0	0	4	NaN	NaN	0.128788
1	0	5	NaN	NaN	0.128788
2	0	6	NaN	NaN	0.128788
3	0	7	NaN	NaN	0.072727
4	0	8	NaN	NaN	0.145455

Table 2: Glass

	Fold	k	sigma	epsilon	Error
0	0	4	NaN	NaN	0.018012
1	0	5	NaN	NaN	0.018012
2	0	6	NaN	NaN	0.018012
3	0	7	NaN	NaN	0.018012
4	0	8	NaN	NaN	0.035119

Table 3: BreastCancer

	Fold	k	sigma	epsilon	Error
0	0	4	4	NaN	12803.860336
1	0	4	5	NaN	12887.144547
2	0	4	6	NaN	13018.754343
3	0	4	7	NaN	13204.663172
4	0	4	8	NaN	13438.168681

Table 4: Hardware

	Fold	k	sigma	epsilon	Error
0	0	4	4	NaN	1081.643774
1	0	4	5	NaN	1091.729321
2	0	4	6	NaN	1097.479727
3	0	4	7	NaN	1100.972476
4	0	4	8	NaN	1103.223865

Table 5: ForestFires

	Fold	k	sigma	epsilon	Error
0	0	4	4	NaN	6.488409
1	0	4	5	NaN	6.490217
2	0	4	6	NaN	6.491433
3	0	4	7	NaN	6.492305
4	0	4	8	NaN	6.492962

Table 6: Abalone

3.3 Edited Nearest Neighbor Error Dataframes

	Fold	k	sigma	epsilon	Error
0	0	4	NaN	NaN	0.0
1	0	5	NaN	NaN	0.0
2	0	6	NaN	NaN	0.0
3	0	7	NaN	NaN	0.0
4	0	8	NaN	NaN	0.0

Table 7: SoyBean

	Fold	k	sigma	epsilon	Error
0	0	4	NaN	NaN	0.200126
1	0	5	NaN	NaN	0.147306
2	0	6	NaN	NaN	0.200126
3	0	7	NaN	NaN	0.200126
4	0	8	NaN	NaN	0.271307

Table 8: Glass

	Fold	k	sigma	epsilon	Error
0	0	4	NaN	NaN	0.050063
1	0	5	NaN	NaN	0.074116
2	0	6	NaN	NaN	0.050063
3	0	7	NaN	NaN	0.050063
4	0	8	NaN	NaN	0.050063

Table 9: BreastCancer

	Fold	k	sigma	epsilon	Error
0	0	4	4	2	26845.57244
1	0	4	4	3	26845.57244
2	0	4	4	4	26845.57244
3	0	4	4	5	26845.57244
4	0	4	4	6	26845.57244

Table 10: Hardware

	Fold	k	sigma	epsilon	Error
0	0	4	4	2	0.0
1	0	4	4	3	0.0
2	0	4	4	4	0.0
3	0	4	4	5	0.0
4	0	4	4	6	0.0

Table 11: ForestFires

	Fold	k	sigma	epsilon	Error
0	0	4	4	2	21.987184
1	0	4	4	3	23.091927
2	0	4	4	4	23.091927
3	0	4	4	5	23.091927
4	0	4	4	6	23.091927

Table 12: Abalone

3.4 K-Means Cluster Error Dataframes

	Fold	k	sigma	Error
0	0	4	NaN	0.577778
1	0	5	NaN	0.633333
2	0	6	NaN	0.366667
3	0	7	NaN	0.183333
4	0	8	NaN	0.577778

Table 13: SoyBean

	Fold	k	sigma	Error
0	0	4	NaN	0.476705
1	0	5	NaN	0.491453
2	0	6	NaN	0.450000
3	0	7	NaN	0.463068
4	0	8	NaN	0.501894

Table 14: Glass

	Fold	k	sigma	Error
0	0	4	NaN	0.491492
1	0	5	NaN	0.195736
2	0	6	NaN	0.215321
3	0	7	NaN	0.348529
4	0	8	NaN	0.348529

Table 15: BreastCancer

	Fold	k	sigma	Error
0	0	4	4	13189.814565
1	0	4	5	13458.649348
2	0	4	6	13795.954409
3	0	4	7	13970.530813
4	0	4	8	14669.380756

Table 16: Hardware

	Fold	k	sigma	Error
0	0	4	4	1675.526814
1	0	4	5	1408.316522
2	0	4	6	1284.545818
3	0	4	7	1156.684400
4	0	4	8	315.583484

Table 17: ForestFires

	Fold	k	sigma	Error
0	0	4	4	5.761004
1	0	4	5	5.980234
2	0	4	6	6.166266
3	0	4	7	6.464557
4	0	4	8	6.346007

Table 18: Abalone

3.5 Explanation of Analysis Dataframes

For a given data set and an estimator, we used the respective error data frame to help find, for each fold, the best set of hyperparameters to use to classify the test set of the given fold. Then, we calculated the error of that classification using the appropriate error function.

3.6 K Nearest Neighbor Analysis Dataframes

	k	sigma	epsilon	Error
0	4	NaN	NaN	0.0
1	4	NaN	NaN	0.0
2	4	NaN	NaN	0.0
3	4	NaN	NaN	0.0
4	4	NaN	NaN	0.0
5	4	NaN	NaN	0.0
6	4	NaN	NaN	0.0
7	4	NaN	NaN	0.0
8	4	NaN	NaN	0.0
9	4	NaN	NaN	0.0

Table 19: SoyBean

	k	sigma	epsilon	Error
0	7	NaN	NaN	0.112500
1	4	NaN	NaN	0.076786
2	4	NaN	NaN	0.102632
3	7	NaN	NaN	0.302339
4	4	NaN	NaN	0.238596
5	4	NaN	NaN	0.038816
6	4	NaN	NaN	0.162281
7	4	NaN	NaN	0.178180
8	4	NaN	NaN	0.219549
9	4	NaN	NaN	0.000000

Table 20: Glass

	k	sigma	epsilon	Error
0	4	NaN	NaN	0.055060
1	5	NaN	NaN	0.066856
2	5	NaN	NaN	0.036706
3	4	NaN	NaN	0.053810
4	4	NaN	NaN	0.033334
5	5	NaN	NaN	0.033334
6	5	NaN	NaN	0.018806
7	5	NaN	NaN	0.027501
8	5	NaN	NaN	0.033334
9	5	NaN	NaN	0.014017

Table 21: BreastCancer

	k	sigma	epsilon	Error
0	4	4.0	NaN	1299.488076
1	4	4.0	NaN	1977.963032
2	4	4.0	NaN	3096.367238
3	4	4.0	NaN	2521.164149
4	4	4.0	NaN	489.446783
5	4	13.0	NaN	5631.087103
6	4	4.0	NaN	5530.943933
7	4	4.0	NaN	4652.864291
8	4	4.0	NaN	874.189836
9	4	4.0	NaN	3423.352814

Table 22: Hardware

	k	sigma	epsilon	Error
0	5	4.0	NaN	1284.001155
1	4	4.0	NaN	1211.443246
2	5	4.0	NaN	5496.024330
3	4	4.0	NaN	14419.136509
4	8	4.0	NaN	24784.125053
5	4	4.0	NaN	1941.760811
6	4	4.0	NaN	927.243123
7	4	4.0	NaN	5146.425664
8	4	13.0	NaN	2713.451797
9	5	4.0	NaN	1070.982657

Table 23: ForestFires

	k	sigma	epsilon	Error
0	8	4.0	NaN	4.279999
1	8	4.0	NaN	5.388882
2	8	4.0	NaN	4.921013
3	7	4.0	NaN	4.840472
4	8	4.0	NaN	5.638111
5	7	4.0	NaN	4.827837
6	8	4.0	NaN	5.194876
7	8	4.0	NaN	4.864881
8	8	4.0	NaN	5.181836
9	8	4.0	NaN	4.317635

Table 24: Abalone

3.7 Edited Nearest Neighbor Analysis Dataframes

	k	sigma	epsilon	Error
0	4	NaN	NaN	0.0
1	4	NaN	NaN	0.0
2	4	NaN	NaN	0.0
3	4	NaN	NaN	0.0
4	4	NaN	NaN	0.0
5	4	NaN	NaN	0.0
6	4	NaN	NaN	0.0
7	4	NaN	NaN	0.0
8	4	NaN	NaN	0.0
9	4	NaN	NaN	0.0

Table 25: SoyBean

	k	sigma	epsilon	Error
0	5	NaN	NaN	0.037500
1	4	NaN	NaN	0.036905
2	4	NaN	NaN	0.102632
3	4	NaN	NaN	0.371711
4	4	NaN	NaN	0.238596
5	4	NaN	NaN	0.038816
6	4	NaN	NaN	0.162281
7	4	NaN	NaN	0.155263
8	4	NaN	NaN	0.219549
9	5	NaN	NaN	0.000000

Table 26: Glass

	k	sigma	epsilon	Error
0	4	NaN	NaN	0.083333
1	4	NaN	NaN	0.000000
2	4	NaN	NaN	0.000000
3	4	NaN	NaN	0.000000
4	4	NaN	NaN	0.029514
5	4	NaN	NaN	0.056349
6	4	NaN	NaN	0.057190
7	4	NaN	NaN	0.056349
8	4	NaN	NaN	0.050505
9	4	NaN	NaN	0.000000

Table 27: BreastCancer

	k	sigma	epsilon	Error
0	5	4.0	2.0	3127.799725
1	5	4.0	2.0	687.614959
2	5	4.0	2.0	553.663764
3	5	4.0	2.0	1463.397914
4	5	4.0	2.0	1164.530725
5	5	4.0	2.0	4124.100793
6	5	4.0	2.0	2991.959947
7	5	4.0	2.0	3187.191451
8	8	4.0	2.0	6953.290571
9	5	4.0	2.0	5793.625846

Table 28: Hardware

	k	sigma	epsilon	Error
0	4	4.0	2.0	0.0
1	4	4.0	2.0	0.0
2	4	4.0	2.0	0.0
3	4	4.0	2.0	0.0
4	4	4.0	2.0	0.0
5	4	4.0	2.0	0.0
6	4	4.0	2.0	0.0
7	4	4.0	2.0	0.0
8	4	4.0	2.0	0.0
9	4	4.0	2.0	0.0

Table 29: ForestFires

	k	sigma	epsilon	Error
0	8	13.0	2.0	8.865848
1	8	13.0	2.0	3.890681
2	6	13.0	2.0	3.154134
3	8	4.0	2.0	3.343808
4	5	13.0	2.0	4.148677
5	4	13.0	2.0	5.492798
6	7	4.0	2.0	2.418472
7	8	4.0	2.0	4.008281
8	5	13.0	2.0	4.110023
9	5	13.0	2.0	10.552076

Table 30: Abalone

3.8 K-means

First we went through the data set and found the errors for all the different combinations of k and sigma for k-mean.

	Fold	k	sigma	Error
0	0	4	4	13189.814565
1	0	4	5	13458.649348
2	0	4	6	13795.954409
3	0	4	7	13970.530813
...				
417	8	5	11	14999.328771
418	8	5	12	15593.320980
419	8	5	13	16284.429109

3.8.1 Best error

Then we found the best error for the data sets based on those combinations:

	Fold	k	sigma	Error
289	5	7	13	2381.857022

Found that a k-value of 7, with a sigma value of 13 was best for this data set.

4. Conclusions

In conclusion we found that the edited nearest neighbor was actually better at predicting regression set of Hardware with a lowest error of 553.66 where the k-means lowest error was 2381.85. Then we saw that normal k-nearest neighbor was actually better at dealing with a regression set than either k-mean or edited k-nearest neighbor. We also found that edited k-nearest neighbors had a much better prediction rate of classification data sets, and this time it was better than k-nearest neighbor where on the breast cancer data set ENN had the lowest error of 0 where NN did not have that low.

5. Summary

In summary we found that edited k-nearest neighbor was superior to k-mean with both regression data sets and classification data sets, and that normal k-nearest neighbor was better at reading regression data sets than ENN.