Machine Learning Project 2: K-Nearest Neighbor

Nicholas Stone nicholas.stone2@student.montana.edu

Matteo Björnsson matteo.bjornsson@student.montana.edu

Abstract

In this paper we present an experiment analyzing the performance of five different algorithms that extend the Nearest Neighbor algorithm. 10 fold cross validation is used to evaluate each algorithm using 0/1 loss and F1 score for classification and Mean Absolute Error and Mean Squared Error for regression. We propose that, in general, k-Nearest Neighbor will perform well on data sets with few classes, data sets that are real valued, and data sets with an even class distribution. We suggest that both Condensed and Edited nearest neighbor will improve upon KNN performance by removing outliers. Finally, we propose that clustering techniques will overall degrade performance in some cases because it may conflate cluster membership with class value. When class distribution cluster well, these algorithms will perform. We did not find any clear support of these hypotheses: many of the "improved algorithms" performed worse. We discuss the possible factors affecting performance. All of these algorithms seem very dependent on the distribution of data. No real conclusions can be drawn here without further testing.

Keywords: K Nearest Neighbor

1. Introduction

In its most basic form, the Nearest Neighbor algorithm is one of the simplest supervised learning algorithms. To estimate a new sample, one simply finds the nearest neighbor and applies the same classification. This paper explores several algorithms that extend this concept: K-Nearest Neighbor, Edited Nearest Neighbor, Condensed Nearest Neighbor, K-Means Clustering, and K-Medoids Clustering. We compare these algorithms against three classification data sets and three regression data sets. Relevant background information is reviewed in Section 2, experimental hypotheses in Section 3, and experimental design in section 4. We conclude with our results and discussion in section 5.

2. Background Information

Nearest Neighbor is a lazy, non-parametric learning method. Here "lazy" means that the method does not derive a model from a data set prior to classifying new examples, but rather classifies examples on demand using the data set. Nearest Neighbor is non-parametric because it makes no assumptions about the underlying distribution of the data. The basic idea behind the Nearest Neighbor method is that, given a new feature vector we want to classify or estimate, we classify the new example to match the class of the "closest" example.

We determine the nearest neighbor using a distance function. This function depends on whether the attributes are real valued or categorical. For real valued data sets Euclidean Distance is used, defined by (1). For categorical valued data sets we will be using Hamming Distance, defined by (2). In the case that the data includes both real valued attributes and categorical attributes, we define a distance metric (3) which combines the two distance functions.

$$D_{\mathrm{E}}(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^2 \right)^{\frac{1}{2}} \tag{1}$$

$$D_H = \sum_{i=1}^{d} H(x_i, y_i), \text{ where } H(x, y) = \begin{cases} 1 & \text{if } x \neq y \\ 0 & \text{if } x = y \end{cases} \tag{2}$$

$$D_{\mathrm{mixed}}(\mathbf{x}, \mathbf{y}) = \alpha D_{\mathrm{E}}(\mathbf{x}, \mathbf{y}) + \beta D_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) \tag{3}$$

In this paper we implement several Nearest Neighbor algorithms that extend the naive implementation of Nearest Neighbor which only uses the closest example to classify a new example.

2.1 k-Nearest Neighbor

The first algorithm implemented is k-Nearest Neighbor, where, instead of simply identifying the closest neighbor, the k nearest neighbors are identified and used to classify the new example via a plurality vote. If there is a tie, the closest example in the tie is used as a tiebreaker. In the case of regression, a Kernel Smoother (4) is used to estimate the real value assigned to the new example, given the N nearest examples. The Kernel used is a Gaussian Kernel (5). Here q indicates the query point, N is the number of neighbors being used to generate a measurement, and d is the number of features in the data set.

$$\hat{g}(\mathbf{x}_q) = \frac{\sum_{t=1}^{N} K\left(\frac{\mathbf{x}_q - \mathbf{x}_t}{h}\right) r_t}{\sum_{t=1}^{N} K\left(\frac{\mathbf{x}_q - \mathbf{x}_t}{h}\right)} \tag{4}$$

$$K(u) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left[-\frac{1}{2}||u||^2\right] \tag{5}$$

2.2 Data Set Reduction

The rest of the algorithms implemented in this experiment all modify the data set that k-NN uses to classify new examples, either by reducing it to only the important examples or by creating a new representative data set using clustering techniques. There are several ways to create a reduced data set. The first modified algorithm is called Edited Nearest Neighbor. This algorithm uses k-NN to classify the training data set, removes all misclassified examples, then re-classifies the remaining examples. This process is repeated until classification performance degrades or no new points are deleted.

An alternative way of reducing the training data set is by constructing it one example at a time via the Condensed Nearest Neighbor algorithm. This works by choosing one random

Table 1: Data set summary

Data set	Character	#Sample	#Attr.	Estimate	Notes
Fire	Mixed	517	35	Regress.	Difficult task, output trends toward 0.
Abalone	Mixed	4178	4	Regress.	Relevant features.
Machine	Mixed	209	9	Regress.	Some irrelevant features?
Vote	Categ.	435	16	Class.	Class split 45/55. Some missing attr.
Segmentation	Real.	210	16	Class.	Equal distribution of classes
Glass	Real	214	9	Class.	Very uneven distribution of classes.

example from the training set to define a new set Z, then iterating randomly over the rest of the examples. For each example $\mathbf{x}_i$, the nearest neighbor in Z is selected. If the two examples do not share the same class, or whose response variables are not equal within some tolerance ϵ, then add the example $\mathbf{x}_i$ to Z. This new set Z is now the data set used to classify new examples using k-NN.

2.3 Clustering

An alternative reduced data set can be constructed by clustering the existing data into k clusters and using the cluster center-points as the data set by which new examples are classified or estimated. This algorithm, called k-Means Clustering, randomly generates k new points $N = \{\mu_1, \mu_2, ..., \mu_k\}$ within the data set space. Every point $\mathbf{x}_j$ is assigned to the nearest point $\mu_i \in N$. Next, the position of each element μ_i is updated to be the center of the points $\mathbf{x}_j$ assigned to it. Every point $\mathbf{x}_j$ is then reassigned to the closest point $\mu_i \in N$. This continues until no points are reassigned or some max iteration step. The resulting set M is then used by k-NN to classify new examples.

The final algorithm implemented here is Partitioning Around Medoids for k-medoids clustering. This is extremely similar to k-means clustering, except instead of cluster points being generated, existing examples $\mathbf{x}_j$ are used as representatives of each cluster, forming a set $M = \{m_1, m_2, ..., m_k\}$. Every $\mathbf{x}_j \neq m$ is assigned to the closest element in M, creating a cluster. Each element $m_i \in M$ is then evaluated. If there exists some point $\mathbf{x}_j \neq m$ that minimizes the distortion function (6) if it were exchanged with m_j, it is exchanged. The process starts over and points are again assigned to the nearest medoid $m_i \in M$, and, again, the point that represents the center of the cluster is updated. Like k-means, this continues until no points are reassigned or some max iteration step. The resulting set M is then used by k-NN to classify new examples. (6) defines the distortion of all medoids. Here k is the number of medoids, and c_j is the set of all points that have been assigned to cluster center m_j.

$$\text{distortion}_{\text{k-medoids}} = \sum_{j=1}^{k} \sum_{i \in c_j} (\mathbf{x} - m_j)^2 \tag{6}$$

Datasets. The six data sets used in this experiment are summarized in Table 1.

3. Hypotheses

In this section, we will discuss how we expect the Nearest Neighbor algorithms to perform on each data set. Additionally, we will review how we expect the different classifications to perform on each of the algorithms.

KNN For the K Nearest neighbor algorithm the group has hypothesized that the algorithm will work best with data sets with only a few possible classifications, such as the Vote data set, as well as classes with evenly distributed data points. This is due to the fact that a given data set with an uneven class distribution will result in overfitting. Additionally, the group hypothesizes that this algorithm will work better on real valued data sets, and will work well with predicting output variables.

Edited KNN Recall that the edited K nearest neighbor algorithm works very similarly to the K nearest neighbor algorithm above, however this algorithm will remove outliers within the data set. By removing outliers, the algorithm will then be able to better classify and predict output values for all of the data sets. We propose that this algorithm will perform at least as well and likely better than KNN on all data sets.

Condensed KNN Condensed nearest neighbor works by randomly generating a representative subset of data. Due to the randomness of this algorithm the hypothesis is that algorithm will improve on KNN, however the improvements will be highly variable. Additionally, we fear it may be susceptible to overfitting uneven distributions.

Clustering We hypothesizes that this both K-Means and K-Medoids algorithms will work well with data sets that have well defined clusters. Namely if a given class has a highly skewed set of data with no real uniform cluster values this algorithm will work poorly. The group also hypothesizes that in scenarios where KNN will perform poorly so will clustering. Thus this algorithm will operate well on data sets whose classes cluster well or have relatively low number of classes and a well distributed data set.

4. Experimental Design

The experiment is divided into several distinct sections: Preprocessing, Learning and Classification, and Evaluation. Figure 1 illustrates the overall workflow for the experiment. Data sets are preprocessed and piped into the nearest neighbor algorithms. The results of the classification test and regression test are evaluated and saved to file.

4.1 Preprocessing

The data sets are transformed by first removing 10 percent of the data set to be used for tuning data. Once the tuning data has been removed the remaining data is then split into 10 roughly equally sized folds to be used for 10 fold cross validation. The categorical data sets are further transformed with stratification, equally distributing all classification examples such that there are roughly an equal number of each example in each of the 10 fold cross validation sets.
Vote is the only data set that has missing attributes in this project. Here each missing

attribute is replaced with a randomly generated value. The final step for all data sets is to normalize all real values features to take on values only between 0 and 1. This is important for classification algorithms that rely on a distance metric, such as K-NN. This normalization assigns equal weight to every feature.

The last preprocessing step is to remove or convert all categorical values when running the k-Means Clustering algorithm. Here we convert the binary Vote data to 0 and 1, the days of the week and months of the year to their numerical equivalent, Abalone sex to numerical values 1, 2, and 3, and remove model and manufacturer names from the Machine data set. This is one way of addressing the nature of the k-Means clustering algorithm which requires a way of finding the "center" of a set of data points, and is best calculated in Euclidean distance over real numbers.

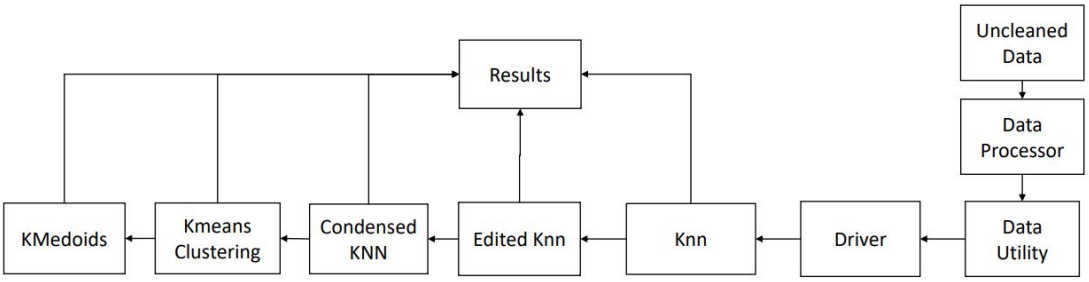
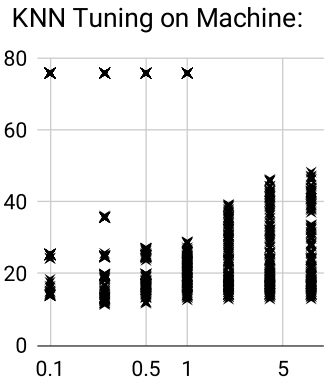


Figure 1: *Flowchart of the experimental design.*

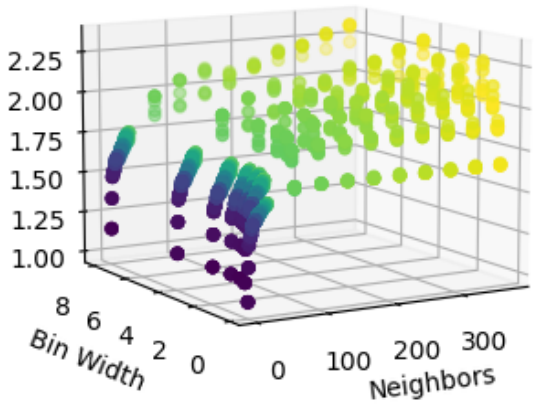
Figure 1 describes the overall data pipeline in the experiment. The project pipeline starts by manipulating each of the given data sets as described above, namely the data sets are split into ten fold cross validation with stratification implemented on categorical data sets. If any of the data sets have missing data then the missing attributes are replaced and are passed to the driver. The driver class breaks each data set from the ten fold cross validation produced in the first 3 classes and runs each of these sets on the algorithms in the following order. The data is first sent into K-NN, then the data is classified and passed to the results class which will generate the overall error and performance. Once K-NN has been completed the data is then passed into the Edited K-NN class and the same process of passing the classifications onto the results object is followed. One can see how the data is then passed down the pipeline to each of the algorithms in the same manner as described above.

4.2 Tuning and 10 Fold Cross Validation

Hyperparameter Tuning Before running each algorithm on the six data sets, several hyperparameters must be tuned. These parameters are tuned by taking a static 10% of each data set to be used for tuning data. Each of the algorithms is run using the remaining



(a) *Smoother bin width (horizontal axis) vs Mean Absolute Error from a tuning run of k-Nearest Neighbor on the Machine data set.*



(b) *Bin Width vs Number of Neighbors, with MAE on the Z axis. Tuning run of KNN on the Abalone data set.*

Figure 2: *Hyperparameter tuning plots.*

full data set as the training set and the tuning data as the test set. The parameters are bracketed over some interval and the algorithm is run with each new value. The performance of the algorithm is then measured. The overall performance for each parameter value is then evaluated, or visualized as in Figure 2, and a parameter value that maximizes performance is chosen.

For k-Nearest Neighbor, the number of neighbors, k, and the bin width for the kernel smoother, h, both play a very important role. For small data sets we bracketed k values from 1 to 1/10 the size of the data set. For large data sets we still bracketed the initial values of k between 1 and 30, then bracketed in larger steps out to 1/10 the size of the data set. Bin width was bracketed across values $\{.1, .25, .5, 1, 2, 4, 8\}$. Bin width in the kernel smoother has a complex relationship to the data so we tried to bracket over an exponentially increasing set of values to capture a large and differing range of possible values. Additionally, the parameters α and β (determining the weights of the real and categorical distance measures) were tuned in the form of a ratio δ ranging from $\alpha/\beta = 0.1$ to $\beta/\alpha = 0.1$.

Both Condensed-NN and Edited-NN use an error threshold to determine if regression classifications are correct. We found the algorithms performed better with smaller error thresholds, resulting in little reduction of the data sets, which suggests we may have been dealing with some overfitting problems while tuning. For k-Means and k-Medoids the algorithms were run with a possible set of clusters ranging from 1 to 100. Many data sets seemed to benefit from a larger number of cluster points. Additionally, cluster count seemed to affect the performance independently from the number of neighbors. The tuning results are summarized in Table 2.

Experimental Setup Once the hyperparameters are tuned for each data set, the remaining experiment is very straightforward. The remaining 90% of each data set is divided into 10 folds. For the classification data sets the classes were stratified across all 10 folds. To

	Segmentation	Vote	Glass	Fire	Machine	Abalone
Neighbors	2	5	5	5	5	12
Bin Width	-	-	-	.1	.25	.1
Error Threshold	-	-	-	1	1	2
β/α	-	-	-	.5	.1	.5
Clusters	80	15	60	60	50	50

Table 2: *Final Hyperparameter values.*

do this we shuffled our data then iterated through it. Each unique class had a fold index associated with it. Each new example went in the bin index associated with its class, and that index was incremented. We randomized the initial values for which class goes in which bin and also made some adjustments to ensure there are roughly the same number of classes in each bin.

Each algorithm is run ten times on each data set. Each iteration a new fold is used as the test data while the remaining data is used as training data. Each trial produces a pair of loss functions which are averaged over the 10 folds.

4.3 Evaluation

Different loss functions are used to evaluate classification and regression data sets. The performance of the algorithm on categorical data sets is evaluated using 0/1 Loss and F1 score. Regression is evaluated using Mean Squared Error and Mean Absolute Error.

0/1 Loss is a measure of classification accuracy and only measures the percent of examples classified correctly. 0/1 score does not weight the consequence of misclassification, but such a measure is not required for this experiment. Furthermore, for each class the number of True Positive, True Negative, False Positive, and False Negative assignments are counted. From these counts the precision and recall are calculated on a per-class basis. The F1 score of each class is the harmonic mean of these two values. The total F1 score for the model is a weighted average of all of the per-class F1 scores.

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

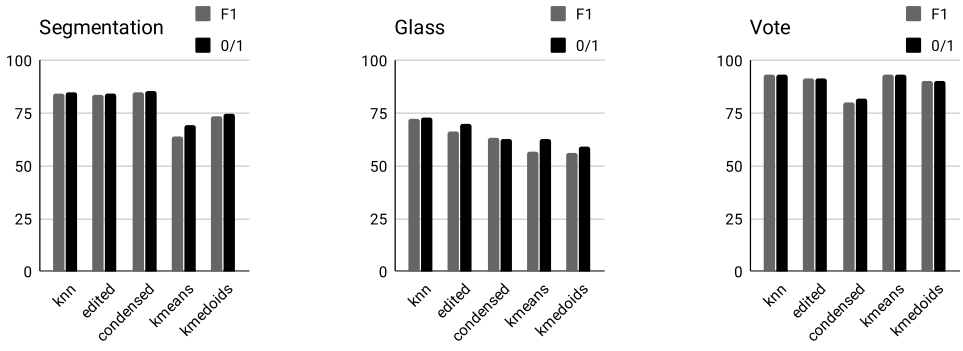
$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

The regression data sets were evaluated using Mean Absolute Error (MAE) and Mean Squared Error (MSE). These loss functions work similarly, with the Mean Absolute Error function taking the difference between the real regression value and the algorithm generated regression value and taking the absolute value of this result. The Mean Squared Error function works similarly by again taking the difference between the two values and squaring the result.

$$\text{Mean Absolute Error} = |\text{GroundTruth} - \text{HypothesizedValue}|$$

$$\text{Mean Squared Error} = (\text{GroundTruth} - \text{HypothesizedValue})^2$$

5. Results and Discussion



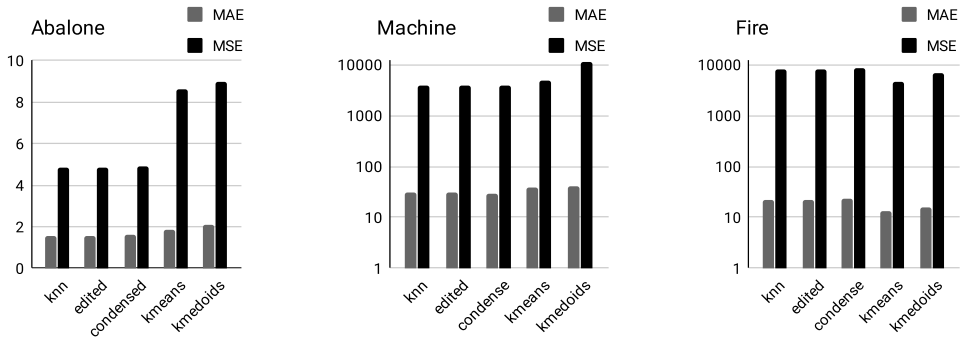
	Segmentation		Glass		Vote	
	F1	0/1	F1	0/1	F1	0/1
k-NN	84.28	85.11	72.46	73.04	93.41	93.37
Edited	83.59	84.52	66.66	69.83	91.67	91.59
Condensed	85.02	85.72	63.31	62.60	80.43	81.91
k-Means	63.87	69.15	56.87	62.71	93.12	93.10
k-Medoids	73.72	74.84	56.25	59.13	90.12	90.09

Figure 3: *Experimental results for the classification data sets. The performance (F1 and 0/1 score) is charted per algorithm per data set. These loss scores are a measure out of 100.*

The experimental results are summarized in Figures 3 and 4. Overall, most of the data sets performed somewhat poorly to very poorly, though Vote and Abalone seemed to perform fairly well. Across the board KNN and Edited NN seemed to perform very similarly, which could mean that none of the data was redundant. Due to the fact that Condensed NN behaved more naturally, it is possible there is an implementation error in Edited NN for the regression data sets, as it seems to be behaving exactly like KNN in these cases. As a point of reference for the performance, for regression the MAE is comparable to the size of the average response variable in the training data. In other words, the magnitude of the error is on average sometimes as much as 50 to 100% of the target value in these cases.

The overall hypothesis was simultaneously correct and incorrect, this was because the algorithm did perform well on data sets that we predicted it would such as the vote data set, however the overall performance of each algorithm was very different than initially anticipated. This is clear from the regression results: the algorithms did not perform as well on these data sets as anticipated. Additionally, on some data sets some algorithms performed worse than the K Nearest Neighbor algorithm when we expected them to perform at least as well or better, such as with Condensed Nearest Neighbor.

The collective hypotheses were not entirely correct due to a few reasons, one of which lies in the data sets themselves. Inspecting the names file for the data set fire, the file



	Abalone		Machine		Fire	
	MAE	MSE	MAE	MSE	MAE	MSE
k-NN	1.53	4.85	29.97	3884.27	21.38	8120.05
Edited	1.53	4.85	29.97	3884.27	21.38	8120.05
Condensed	1.61	4.93	29.82	3875.90	23.22	8384.36
k-Means	1.87	8.61	37.74	4743.50	13.43	4656.23
k-Medoids	2.09	9.00	40.27	11065.27	15.43	6954.64

Figure 4: *Experimental results for the regression data sets. These loss scores (MAE and MSE) are a measure of error of the regression estimate compared to the known value. This means the magnitude of the values are data set specific and are not directly comparable. For reference, the mean response variable value is 9.93 for Abalone, 105.6 for Machine, and 12.8 for Fire. Note the log scale for Machine and Fire.*

explicitly says to use a logarithmic function when conducting learning algorithms on the data set. Additionally the data set claims to be a data set that is difficult to learn from, this is due to the high variability in the data itself. This data set may have performed poorly due to skewed distribution of the data. The second error that occurred resulted from the assumption that each data set would perform better since the algorithm conducted some small task to change the data set such as removing outliers. It is possible that these small "improvements" could have increased overfitting in our data set thus resulting in an overall decrease in performance.

The experimental design implemented in this paper was one that aimed to simplify the experiment by running each algorithm on each of the given data sets in a well structured pipeline. If the experiment was to be re-conducted the experimental design would change such that each data set would be run against each algorithm individually. This would allow very fine grained hyper parameter tuning as well as allow the programmers the ability to interpret the results directly. This of course was highlighted due to the fact that in the current pipeline, the programmer must wait for the overall program to finish before analyzing data, which was not optimal for tuning each individual algorithms hyper parameters.

A few unique and interesting questions were identified when running the experiments on the given data sets. The biggest point of discussion is at what number of neighbors will the nearest neighbor algorithm perform optimally. While our tuning lead to an assumption

of a most optimal k, it is important to state that each data set will have its own optimal k value that may require much more careful inspection to optimize truly. It is clear that while it may seem that "more is better," K-NN as an algorithm is clearly greatly affected by the data set. The proximity of each data point to its neighbors and the uniformity or distortion of density across the data set must be tuned for particularly. Clustering fits this description as well, though it introduces the additional issue which is the tendency to conflate clusters with class membership. Data with a skewed distribution will result in over-fitting the data since a given point may be looking at a k value where it will almost always produce the most common occurring classification. Of course the same underlying principle stated above applies to all of the algorithms discussed in this paper. It is clear that more data would have to be collected to empirically understand these algorithms and their relationship to the data set.

6. Conclusion

This paper presented a study of several non-parametric classification algorithms that revolve around the Nearest Neighbor algorithm. The algorithms were all tested on six data sets, the performance of which was evaluated using 10-fold cross validation. In comparing our hypotheses to experimental results we found no clear conclusions. We did predict that the Vote data set would perform well due to having few classes and attribute values, but we also predicted that these algorithms would perform well generally on regression, which was no generally the case here. It is abundantly clear that the performance of these distance based algorithms are highly dependent on the character, shape, and distribution of the data set. It seems that an detailed analysis of the data sets themselves might provide the best insight into K-Nearest Neighbor algorithms. Extensive testing with a better hyperparameter search method could prove insightful.