KNN Classification Optimization through 10-Fold Cross Validation and Logic Gate Implementation Through Neural Networks

Timothy Ryan Lovvorn

April 7, 2017
ECE 471 Pattern Recognition
Undergraduate
tlovvorn@vols.utk.edu

Abstract

Data sets often cluster in grouping indicative of their classifications. Meaning one may determine the class of a given datum based upon the classifications of the nearby datums. This method of classification, which is flawed owing the minimization of error method used to create said classifier, may be improved through cross fold validation of the classifier. This method ensures multiple creations of the classifier as well as testing sets still result in the desired accuracy. In addition, multilayer perceptrons may be utilized to classify non-linearly separable data sets. Through a three layer perceptrons, a neural network, complex classification problems such as XOR logic gates may be implemented. This report seeks to create several incarnation of a tri-layer neural network to implement this logic gate effectively. Overall, with accuracy rates an average of 70% and as high as 100%, there was a good deal of variance, but great deal of accuracy and scientific usage gained through this project.

1.Introduction

Often with our data sets we are presented with a fundamental issue: the accuracy and reliability of our classifier when used in an experimental setting, versus usage in more practical, applied setting out in the world. Training sets are limited in scope and size purely by the limitations of data gathering and processing power. As we simply cannot ever include the entirety of all possible data points for a set, we cannot know if absolute certainty how well our classification model works in the real world. We may circumvent this issue through the usage of n-Fold Cross validation. Through splitting our data set into multiple different sets, we may achieve greater overall accuracy of our results, with the trade off being the larger the validation set, the smaller the training set and vice versa [4]. Through experimentation with varying sized validation and training sets, we may better determine the overall representative power of our data set to the actual real world entire data set as it allows us to test multiple configurations. We may apply this to our K-th Nearest Neighbor, KNN, classification method.

KNN classification relies upon the assumption data points of similar classes generally congregate in the same spatial area. Based upon this, for unknown data points we may determine the classification of said point by gathering the classes of a k-th number of neighbors, and giving a majority rules assertion of which class our unknown data point belongs to [1]. Based upon our Bayesian Formula:

$$P(w_i|x) = \frac{P(x|w_i)P(w_i)}{P(x)}$$

We may conclude, by the methods outlined in section 2.2 below, which class a given point may belong to from existing class information [2]. We may perform this several times on various k-th sized volumes of neighbors to determine what the most accuracy number of neighbors may be in order to classify our given data. The aforementioned cross validation is utilized as a further method to ensure data to ensure of classifier is representative of all possible data subsets of the larger real world data set. This is performed through our C++ coded program in the appended source code, labeled FGE.cpp, which separates the data into folds, and performs the k-th classifiers on various different folds and validation sets to determine effectiveness.

Extensive work with neural networks began back in the 1960s with a paper which first proposed the modern neural network algorithm. This paper proposed this would work in a manner similar to swarm optimization, allowing machine learning to determine through the network the best possible flight paths around extraterrestrial objects like Mars [3]. A paper by Rumelhart et. al. in the 1980 further expounded upon the way neural networks allow for parallelization of the optimization issues. They allow multiple testing configurations to attempt to optimize an issue, as well as their formation allowing for parallel creation and running [5]. Of particular interest to us is the 1970s paper by Voraeck, which outlined while perceptrons are unable to correctly solve non-linearly separable problems, multiplayer perceptrons, neural networks, could in fact be utilized to solve these more complex problems such as XOR logic. Voraeck proposed a simple 3 layer neural network to solve the XOR problem while postulating all multiplayer perceptrons are NP complete, that is they may be used to solve non-polynomial time, difficult problems [6]. Our paper explores this concept by using the 3 layer neural network design to solve just such a problem. These multiplayer perceptrons no longer constrain us to using simple binary separation of a classifier, and with sufficient layering, may be utilized for a number of possible classification problems. In our case, we will constrain ourselves to only solving the XOR issue. I also opted to, rather than use an existing neural network implementation, create my own 3 layer neural network in the C++ language to perform this. My network takes in the input, in this case only 4 possible values, gets a desired output, then adjusts the weighting of the neurons accordingly. After convergence to the ground truth is achieved, or a particular number of epochs are performed, the neural network stops training. At this point a final test set is run and the accuracy of this particular incarnation is evaluated. This entire sequence is performed multiple times to gain an average accuracy of the implementation as well as the best performance scenarios. The C++ source code is appended in the FGE.cpp file, and all relevant information on how to run the simulations and gather data is contained in the appended README.txt file appended to this report. All graphical representations were created through 3rd party software in Microsoft Excel spreadsheets.

2. Technical Approach

2.1 Data Normalization

To begin our work, data must be normalized, as data set features often have vastly different ranges. Normalization causes all data features to scaled to a 0 to 1 range, allowing all features to retain their relationships while now all having the same range. In our case, we achieve this by, for each feature, determining the mean and variance of a particular feature, using our Gaussian equations, as we assume normalized data for each jth feature:

$$\mu_j = \frac{1}{n}\sum_{i=0}^{n} x_j \qquad\qquad \sigma_j^2 = \frac{1}{n-1}\sum_{i=0}^{n}(\mu_j - x_j)$$

After these are determined for each feature, the individual elements of the feature may be normalized with the following equation:

$$x' = \frac{(x - \mu_j)}{\sigma_j}$$

For our model, we used 10-fold cross validation, meaning it was necessary to perform this task 10 separate times. On each individual training sets with the given m fold removed. For each m_{tr} fold, the same mean and variance values are used to train the m_{te} fold which is held out of the training set. After full normalization has occurred, classification may be performed.

2.2 KNN Classification

The initial classification may be performed using our Bayesian conventions. The probability of our current x data point occurring given, w_i is representable as a ratio of occurences of w_i within our entire sample data space over the overall volume of the space. Prior probability may be considered as the number of occurences of a particular class, over the entire sum of all occurences within the space. The evidence is thought of as the ratio of neighbors in a classification consideration space over the total number of neighbors, all over the volume of the space:

$$P(x|w_i) = \frac{\frac{k_m}{n_m}}{V} \qquad\qquad P(w_i) = \frac{n_m}{n} \qquad\qquad P(x) = \frac{\frac{k}{n}}{v}$$

We may substitute these values into our Bayesian formula to simplify to the following:

$$P(w_i|x) = \frac{k_m}{k}$$

Note that this system considers prior probability as already determined by the system. It is also possible to leave this value in flux during training to attempt to find a more optimal prior probability for each classifier's probable occurence. Recall our determined values for prior probability are only as representative as our data set is of the entire possible data space of all values in the system. With a sufficiently representative training set, this will be sufficient. If their is a higher degree of uncertainty as to the representative ability of our training set, such a guess-and-check method to determine prior probability may be prudent.

2.3 N-Fold Cross Validation

As alluded to in section 1, and section 2.2 in relation to prior probability, there is a degree of uncertainty as to how representative our training set is to the entire possible data set of all possible datums for our given study. To increase the assuredness of the validity of our classifier, we may use the N-fold cross validation, whereby we divide our training set into N number of fold and perform N number of classification training processes. For each iteration, an ith fold is held out of the overall set of N folds, and the N-1 folds are utilized to create our classifier. The ith fold, called the validation set, is then used to test the accuracy of the classifier. This is performed an N number of times for each fold, then an average of accuracy for all N cases may be computed:

$$TotalAvgAccuracy = \frac{\sum_{i=1}^{N} IterationAvgAccuracy_n}{N} * 100$$

In this manner, we may create N classifier's and test them using the same data set. There is a potential trade off, as a larger N will result in a larger classifier, but a smaller, less accurate validation set. A smaller N will result in a smaller classifier, but a larger, more accurate validation set. As a result, the best N number of folds to utilize to maximize the accuracy for the usage of N-fold cross validation is largely dependent on the size our your entire training set. For our experimentation, we chose 10-fold cross validation. Given the size of our overall data set, this was able to provide us with a large degree of accuracy in the testing phase while still having a sufficiently large testing set to create our classifier.

2.4 Neural Network

2.4.1 Forward Propagation

Our multilayer preceptron, neural network, must be able to adjust the weight and bias values based upon the given input with respect to the desired input for each epoch and individual iteration within an epoch. Values must first be taken into the input layer, whereby each input, in our case 2, is taken into the network and summed, with respect to the weighting given to each: I consider h to be the neuron, b to be the bias, x to be the input, and w to be the weight:

$$h_i = b_i * w_{i0} + x_{i1} * w_{i1} + x_{i2} * w_{i2}$$

The given values is then introduced into the sigmoid threshold function to determine the feed forward value to the next neuron in the hidden layer as input:

$$S(h_i) = \frac{1}{(1 + e^{-x})}$$

The resultant value is then plugged into the hidden layer to obtain the sum of h:

$$h_i = b_i * w_{i0} + S(h_{(i-1)1}) * w_{i1} + S(h_{(i-1)2}) * w_{i2}$$

And the final output o is determined by again applying the sigmoid function:

$$o_i = S(h_i)$$

4.2.2 Backpropogation

After the final output has been computed, the network's weights must be shifted by a given value based upon their relation to the ground truth,T. If we consider this change as a delta value we begin at the outermost layer:

$$\Delta_o = o_i * (1 - o_i) * (T_i - o_i)$$

And the inner layers are computed using the calculated sigmoid values:

$$\Delta_i = S(h_i) * (1 - S(h_i)) * w_i * \Delta_0$$

With our deltas, we can not change our weights, starting with our bias weights:

$$w_{ij} = w_{ij} + learnRate * b_{ij} * \Delta_i$$

Finally we update our weights for the inner layer:

$$w_{ij} = w_{ij} + learnRate * S(h_i) * \Delta_0$$

As well as the outer:

$$w_{ij} = w_{ij} + learnRate * wji * \Delta_i$$

In my model, this training, performing this for each iteration for all sets in a given epoch, is performed until either 500,000 epochs have been completed, or the following condition is met:

$$\begin{aligned} if\,|o_i - T_i| \geq .1\; then\; Not\; Converged \\ for\; all\; i\; in\; network \\ else\; Converged \end{aligned}$$

This checks to ensure the error between the ground truth and given output for each value in the training set is no larger than .1. If this is the case, the neural net is considered to be converged, and network breaks out of its iteration.

As with the KNN classifier, determining how accurate the network is versus its usage in the applied world outside of academic settings can be quite difficult. Somewhat uniquely in this case, our entire possible data values space is easily enumerable, as a space with only 2 binary input mechanisms may only take 2^2 possible values. This allows us to train the network with the entire possible data values space. In order to test the accuracy of using a neural network to solve the XOR classification, my method performed the creation of the network 500 times. Each time, after either 500,000 epochs or convergence is achieved, an average accuracy is calculated by determining if our value for each of the 4 possible data points give the desired ground truth output:

$$RoundAvg = \frac{number\; of\; correctly\; classified\; values}{total\; number\; of\; values} * 100$$

After 500 rounds, an accumulated average is computed:

$$AccumulatedAvg = \frac{\sum_{n=1}^{500} Round\ Avg_n}{500} * 100$$

3. Experiments and Results

3.1 The Data Sets

The data sets were obtained from the file fglass.dat. The data set consisted of 214 data points. Each data point had a total of 9 features based upon, as best can be determined from the file, elemental properties of each data value. The final 10th column of each row is the expected class of each data value. In order to perform our 10-fold cross validation, we also had the fglass.grp, which contained 10 sets of values corresponding to indices in the original data set of 214 values. The original data set was split into 20 different sets, or 10 pairings. For each pair, their was a validation set composed of all the values for a given set indicated by the indices in the fglass.grp file. The paired set contained all the remaining values for the entire original data set. All KNN operations were performed on all the given data sets to perform the cross validation appropriately.
For the Backpropogation neural network, as the possible input space was easily tractable given the 2^2 size, all possible inputs values of 00, 01, 10, 11 were utilized for every epoch of testing. The same set was used when testing the accuracy of the neural network, as creating any set larger would only have repeats and not give further useful data. As alluded to above, multiple iterations, 500 to be exact, were created of the network with randomized weights between 0 and 1 to observe the possible effects of different starting weights. A small amount of experimentation was also done with learning rates, but generally the rate of .1 was kept to ensure convergence.

3.2 KNN Classification 10-Fold Validation

3.2.1 K Optimal Determination and Euclidean Distance

With our current K classifier, the distance metric utilized was Euclidean. I attempted to classify my folds using K values between 1 and 70. For each fold, the number of correct results would be determined and the total correct sum would be tallied before dividing by the total number of samples in all folds combined. The resultant averages compared to their accuracies are shown below in Figure 1:

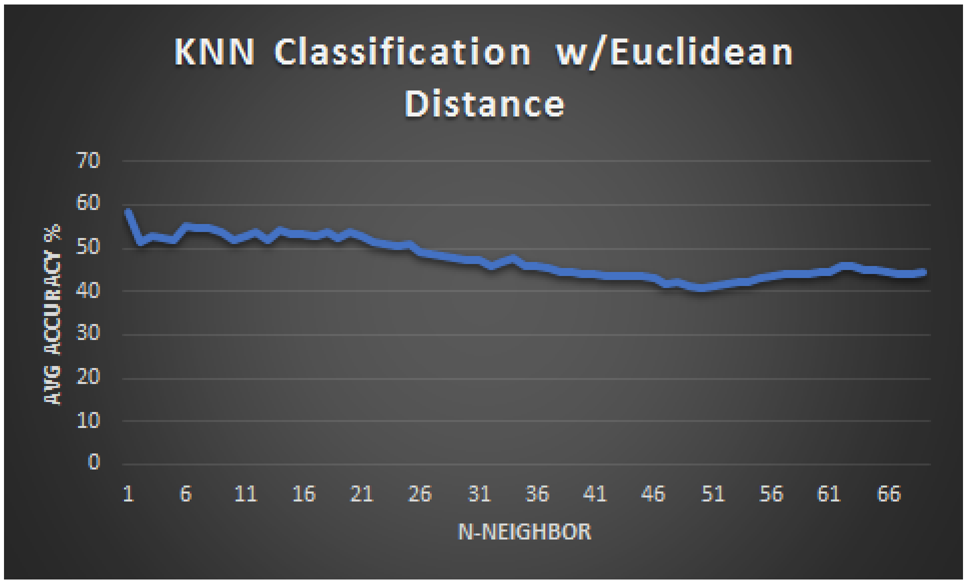

Figure 1: Illustrates the changing Accuracy vs. the K number of neighbors utilized.

Interestingly, the first round, with k of 1, had the highest accuracy range of all possible values. This is suggestive of two possible thoughts. Either all data points a clustered very closely together in a Euclidean space with class clusters all nearly overlapping, or their was an error in computation. Examination of my colleagues data leads me to believe the latter is the case, although the data was still somewhat accurate. In future iterations of this experimentation, I would recommend performing a form of dimensionality reduction on the data set, as this would allow graphing to a lower dimensional space, which would give better graphical representation. The conclusion of very tightly clustered data may be erroneous, but is the best conclusion which may be drawn from this data.

3.2.2 Higher Order Distance

After performing the initial classification and determining 1 to be the best K value, I next utilized minkowski distance to perform higher dimensional, and lower dimensional, distances to determine if the results were affected. The results are in the following table, with the second order distance being the original Euclidean distance:

Order of Distance	Calculated Distance
1	58.4112
2	58.4112
3	58.4112
4	58.4112

Figure 2: Illustrating the distances gathered using different orders of Minkowski Distance

Interestingly, there was no calculated change for any of the different orders of distance. Again this leads to two possible inferences. The distance is immutable over different planes, or a calculation error with the initial k values, for which k = 1 is utilized here. The immutability of distance is a consideration as the shortest distance between two points is constant; however, this relies on the distance being measured as a single straight line, which does not necessarily hold true in higher dimensional spaces. Comparison to my colleagues work would be the best method to make the most accurate determination.

3.3 XOR Neural Network

For XOR I performed a total of 500 rounds for creation of my Neural Network. In my system, there were at total of 3 layers. For the experiment, I did experiment with different values for the learning rate, but found the rate of .1 to be sufficient for the purposes of my network. Figure 3 illustrates the accuracy ranges I encountered throughout different rounds of my network creation.

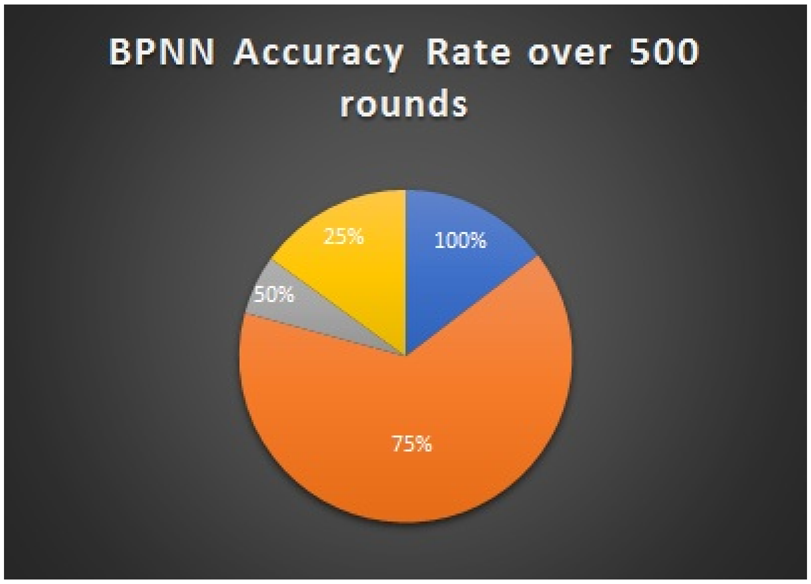

Figure 3: Accuracy rates of the BPNN over 500 runs. Learning rate was .1

As illustrated, an accuracy rate of 75% per round when convergence of the final epoch was reached was by far the most common. Total convergence or a single correct value were also common, with an inaccuracy rate of half being the least common. Averaged over all runs the accuracy is 69.3%. This value did change depending on the total number of epochs as well. With a rate as high as 77% being achieved when performing 1 million epochs per round. This is indicative of a direct relation to the overall number of epochs performed and overall convergence. While not all values did ultimately converge completely, there was a clear relation to more convergence with higher epochs, suggesting even very poorly initially configured networks may eventually converge given enough time. Figure 4 shows the average rate of change in output over various epochs until total convergence is achieved:

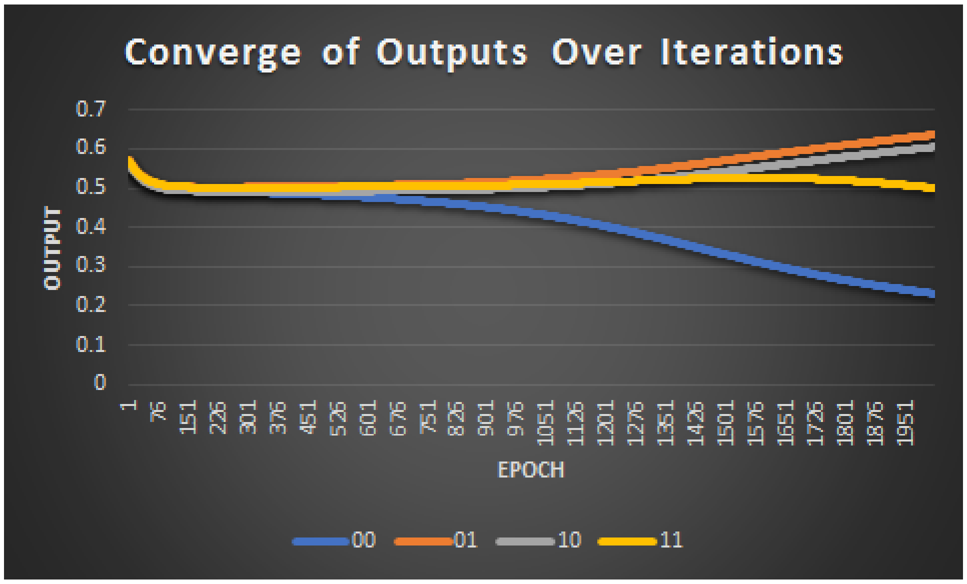

Figure 4 : The Typical Convergence of a Given round. Round 2 for my data is show here.

As shown, while the data points all begin at their randomly assigned weight, they gradually move toward their desired values of greater than or less than .5. The overall time it took for a particular network to converge did vary, with figure 5 showing the convergence rate of those with achieved complete convergence:

Rate of Convergence in Various Rounds	
range(epochs)	**Number Converged**
5,000 or Less	13
5,001 to 10,000	24
10,001 to 20,000	17
20,001 to 50,000	11
50,001+	5

Figure 5: The rates of various totally converged rounds

Note that I only considered rounds with 100% accuracy to be completely converged. There is a noticable bell curve to the rate of convergence, which does seem sensible given the weight initializations. Weights were randomly initialized with any value between -1 and 1 for each round. As a result of using true psuedo-randomness to initialize, we would expect this normalized sort of distribution for accuracy. It is also worth nothing while approximately 14% achieved total convergence, over 80% of the total rounds achieved an accuracy of 75% or higher. In particular for a known data set like the one used in the case of XOR, only a single case of high accuracy is needed, meaning this is a very practical model to utilize.

4. Summary

In summation, using our Bayesian Theory in conjunction with our understanding of N-Fold Cross Validation to improve upon the accuracy of our initial kNN classification method. Through its usage we are able to gain a clearer understanding of the possible limitations of our training sets. As mentioned at the introduction, one of the greatest issues with our classification methods is the uncertainty of how representative our training set is of the entire possible data set in the wider world. N-Fold Cross Validation allows us to compensate for this by using the same training set to create multiple classifiers, when we may then verify the accuracy of multiple times.When it is prohibitively expensive or difficult to gather large data sets, this is a very useful method of creating classifiers, as well as validating them.
Through the usage of neural network learning with the Backpropogation algorithm we were also able to construct a BPNN capable of solving the non-linearly separable problem of XOR, with an average accuracy of nearly 70% and ranging up as far as 100%. This method is extremely useful as it allows us to solve a myriad of complex problem even with a small training set, provided we have sufficient training of the network. The implications of a neural network are vast as they are virtually unconstrained in the level of possible relationships they may classify. While accuracy can be lost the more classes one has, this may be rectified by utilizing multiple neural networks as well.

The majority of the objective of this paper was met without great difficulty. My overall understanding of the classification mechanisms, in particular the cross validation method, we increased in the course of this project.I achieved a level of personal improvement as we as I have had experience with neural networks previously, but was unable to achieve any higher level of accuracy. With this system my understanding and implementation have greatly improved my understanding of the subject matter. Further work may also be completed. With the kNN classifier, dimensionality reduction would likely be prudent to allow for better data visualization to confirm the experimental results. Optimization of the learning rate for the neural network is also a possibility as while the .1 rate will converge, it may take a prohibitively long period of time to do so. Further work would also be completed on the training of weights, as weights were randomly allocated within a specific range for this project; however test results indicate their initial starting point has a dramatic effect on convergence rate. To this end testing the relationship between the range of weighted values, to the range of normalized data points may be prudent, as this could lead to faster, more consistent convergence. The activation function could also be improved, as sigmoid was utilized without optimizations.

This program may easily be expanded as well, as the utilities to create higher order distance kNN classifiers has be fixed to the modification of a single line of code. The overall neural network design, while small in this case, may be easily generalized, as the basic flow of operations does not change, only the number of weights, neurons, and layers upon which they must be performed. In addition we have expanded our overall arsenal of classifiers with a further, very dynamic method of classification. As with previous experimentation, the possible social or scientific applications are numerous and already existant. These forms of neural network classification are already common in image recognition and recovery alone. The far reaching future implications of Artificial Intelligence are quite exciting, and highly applicable to our exponentially evolving world.

5. References

[1] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, *46*(3), 175-185.

[2] The Editors of Encyclopedia Britannic.a. (2017). Thomas Bayes. Retrieved February 12, 2017, from https://www.britannica.com/biography/Thomas-Bayes

[3] Kelley, Henry J. "Gradient theory of optimal flight paths." *Ars Journal* 30.10 (1960): 947-954.

[4] Kohavi, R. (1995, August). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai* (Vol. 14, No. 2, pp. 1137-1145).

[5] McClelland, J. L., Rumelhart, D. E., & Hinton, G. E. (1986). The appeal of parallel distributed processing. *MIT Press, Cambridge MA*, 3-44.

[6] Vorácek, J. (1970). Design, properties and applications of a neural tree classifier. *WIT Transactions on Information and Communication Technologies*, *12*.