Comparing k-Nearest Neighbors and Neural Networks

Kevin Brachthuizen, Bas Broerse, Axel Ehrnrooth, Yari Koot, Daniil Paplauski

31 March 2023

1 Abstract

To what extent does a neural network outperform a k-nearest neighbors algorithm? This paper investigates the implementation and performance of two machine learning algorithms: k-nearest neighbors, and a neural network. The characteristics and advantages of each algorithm are explored, and their performance on the specific data set is compared. The kNN algorithm is a simple and powerful classification algorithm that is based on the distance between instances. A neural network, on the other hand, is a complex machine learning model capable of learning from patterns in data.

2 Introduction

In this report, two machine learning algorithms are compared: k-nearest neighbors (kNN) and neural networks. The main advantage of kNN is that it is one of the simplest supervised learning algorithms[10]. It is fairly straightforward to build a kNN algorithm without using pre-existing packages. A neural network is much more complex. Its complexity, however, might allow for increased performance. The issue at stake, therefore, is to evaluate to what extent a neural network outperforms a k-nearest neighbors algorithm.

The k-nearest neighbors algorithm is a well-known classification method. The kNN algorithm computes the distances to the other instances given a new instance and takes the k-amount of nearest neighbors to it, where k is a positive integer. Then, kNN assigns a class to the new instance by checking which class is most common among the nearest neighbors.

Neural networks are another popular group of classifiers. It uses the features in a given set of data to build one or more hidden layers, which combine linear and non-linear relations to find an output that corresponds to a class. This class will then be assigned to the new instance.

The data used for this experiment is a database for the quality of milk. The quality is categorized into three classes: low, medium, and high. These classes are based on seven parameters: pH, temperature, taste, odor, fat, turbidity, and color. The goal of both algorithms is to be able to accurately classify the quality of milk using these seven features. The performance of the algorithms is evaluated based simply on a comparison between the number of (in)correctly classified instances from each algorithm.

In this report, we will first discuss the data set a little bit further, as well as the process of preparing the data. We will then take a closer look at the kNN algorithm and neural networks respectively. Finally, we discuss the results of running these algorithms on our data set.

3 The Data

The data set used in this research was found on the internet platform Kaggle [16]. We will first give a broad description of the data set, before getting into the pre-processing and normalization of the data and the splitting of the data into a training set, a validation set, and a test set.

3.1 Data Set

The data consists of seven features and one class for each instance. The class, which is the target value, is a discrete variable. Each sample of milk can be of low, medium or high quality. For the kNN algorithm, we will keep these three categories intact. For the neural network, we need each class to correspond to a numerical value. We choose to give a value of 0 for the low class, 0.5 for medium, and 1 for high. More details on how the values from the neural network correspond to the class output can be found in the section on the neural network.

The dataset contains 1058 instances. Of these, 429 are of low quality, 374 are of medium quality, and 255 are of high quality. The figure below illustrates this distribution.

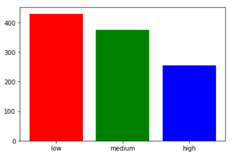

Figure 1: Class Distribution

The seven features of the data, are pH, temperature, taste, odor, fat, turbidity, and color. Four of these are categorical variables: taste and odor are either good (1) or bad (0) and fat and turbidity are either high (1) or low (0). The other three features are continuous variables. pH ranges from 3 to 9.5 and temperature from 34°C to 90°C. Color is defined by the RGB model and ranges from 240 to 255.

3.2 Pre-Processing

Before the data set can be used, a few problems need to be taken into account.

Each feature of the data has a different domain. For the neural network, this is not a problem. If there are two features with different domains that have the same effect on the class output, it will just give smaller weights to feature with the narrower domain. In a way, the neural networks re-scales the domain. kNN however, makes use of euclidean distances between data points in the feature space. This means that a feature with a wider domain will have larger distances between instances relative to a feature with a narrower domain. The result of this is that kNN weighs different features differently. To combat this, we normalize the data so that each value is scaled between 0 and 1.[13] For each feature, we find the difference between its maximum value and its minimum value. Then, for each instance, the difference between its value and the minimum value of the respective feature is divided by this domain width. The outcome of this manipulation is the normalized value for the instance. For instance i and feature j, with feature value $x_{i,j}$ this gives us the following protocol:

$$\frac{x_{i,j} - min_j}{max_j - min_j}$$

This protocol was followed for every instance of the non-binary variables, since the binary ones are already either 0 or 1.

The next task is to divide the data into a training set, a validation set, and a test set. It is important that the data we use to train the model, whether it's kNN or the neural network, is not recycled to test the model on, as this would skew the evaluation.[12] So we see the need for splitting up the data into a training set and a test set. This is not all, however. We also need to choose hyperparamaters for the models. For kNN, we have to choose which value to use for k, and the neural network, we have to decide on the number of hidden layers to use and the number of nodes per layer. We also choose the learning rate and the dropout rate. Here, we are again unable to use data from either the training set or the test set, as that too would mess up the evaluation process. Therefore, we need another split to give the validation set. For both models, we choose to use 60% of the data for model training, 20% for validation and 20% for the testing.

4 k-Nearest Neighbors

1 what is it 2 how does it work 3 what are the limitations

k-nearest neighbors is a popular supervised learning algorithm that can be used for both classification and regression.[9][8][10][6] In classification, it observes the target categories of a specified number of nodes nearest to a given unclassified node. These nearest nodes are called neighbors and the number of neighbors that are examined, is k. kNN takes the category that occurs most frequently among the k neighbors, and assigns this class to the given node. The algorithm is based on checking memorized data, rather than actual learning. For this reason, kNN is known as a "lazy algorithm".

Before implementing the kNN algorithm, we need to consider a number of topics. First of all, it is essential that we define how to find the nearest neighbors. We do so by the euclidean distance between points in the feature space. Since our data set contains seven features, we are dealing with a 7-dimensional space. Note that we use the normalized values for each feature.

Then, let x_i be a classified node and y_i an unclassified node. Then, for each feature $j \in \{1, 2, 3, 4, 5, 6, 7\}$, we compute the distance $d_{i,j} = y_{i,j} - x_{i,j}$. Then, to find the euclidean distance d_i on the 7-dimensional feature space, we square the $d_{i,j}$'s, add them together and take the root:

$$d_i = \sqrt{\sum_{j=1}^{7} d_{i,j}^2}$$

Note that the more dimensions the feature space has, the lower each dimensions distance weighs in the overall euclidean distance. Because of this, we have to be aware that the higher the dimension, the closer the nearest neighbors will be to the farthest neighbors. Research shows that this effect shows up with feature spaces of around 10 to 15 dimensions.[3] Since our data set only has seven features, we can assume that kNN is still meaningful.

Next, we need to consider how to choose the hyperparameter k. A smaller k ensures that we don't consider distant neighbors, but a larger k decreases the probability of outliers affecting the outcome.[1] Considering this paradox, we will try $k = 1, 2, ..., 25$ on the validation data to choose a k that is optimal for this specific data set.

4.1 Implementation

In implementing the kNN algorithm in Python, we make use of popular libraries pandas and NumPy.[2][11][17]. Pandas is a library commonly used for its DataFrame data structure, which, along with the vast number of methods, makes it easy to process and use data. NumPy is a commonly used library for linear algebra in python, in the case of kNN, the sum and square root methods are used, as they are more efficient in use than python's built-in methods for these operations. We pass in an integer k and two sets of data: the training set and either the test set or the validation set, which we will call unclassified set for clarity. Then, for each data point or node in the unclassified set, we perform two steps. The first step is to copy the training set. We compute the distance from the unclassified node to each training node and then order the copied set by this distance. Now, for the second step, we pick the k training points that are closest to the unclassified node and count the occurrences of each class label. The label with the highest occurrence is then assigned to the target node. We repeat these two steps for each instance in the unclassified data.

After assigning class labels to each point in the unclassified set, we can compare these labels to the actual data. Here, we simply count how many instances were predicted correctly.

For the validation data, we run the kNN algorithm one time for each $k = 1, 2, ..., 25$. We choose a k based on these results. This is the k we use for the final run of the algorithm: the run over the test data.

4.2 Analysis

The hyperparameter optimization for k was performed by running the algorithm on the validation data with a split of 60%/20% (leaving 20% as the test data) using all values for k between 1 and 25 and observing the total number of incorrectly classified instances. The results can be seen in figure 2, which shows that the optimal values for k are from 6 to 10, which all misclassified 11 instances. The values that were higher than 10 caused a consistently increasing error. Selecting the optimal value of k from the 5 best values was based on minimizing the chances of ties in the classification. Since there are three possible classes, selecting based on odd numbers does not yield the same benefits as it would for data sets with two classes. The values 6 and 9 were immediately removed to avoid any three-way ties, as they are both evenly divisible by 3. Out of the values 7, 8 and 10, there is no difference in the chance of having a tie, so the final selection was based on two factors. Since the error increased consistently with higher k values, 10 was removed as it was the highest of the three. Though there is no theoretical difference, in this case, in the chances of a tie between 7 and 8, the conclusive k-value was chosen to be 7 as it is not only odd but is also a prime number.

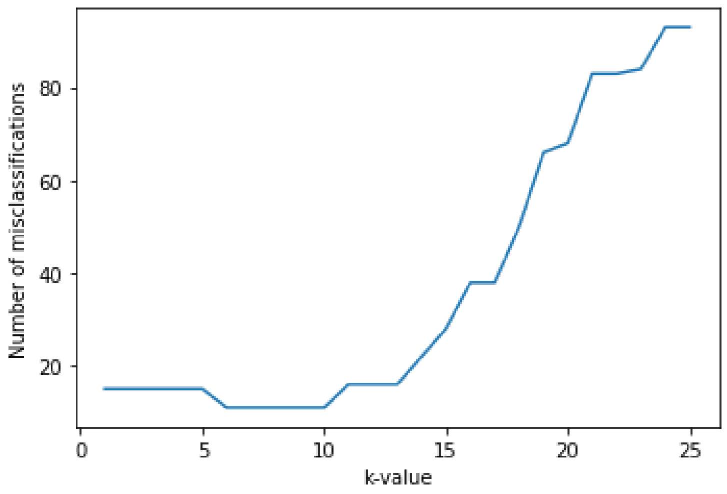

Figure 2: Optimization of k-value

When running the algorithm on the test data using the optimized value $k = 7$, it performed extraordinarily well, resulting in a correct classification of all instances in the test data.

5 Neural Network

5.1 Motivation

Though the kNN algorithm implemented in this project performed well on the test data, it is still a lazy algorithm and does not aim to better itself based on a loss function, for example. This means that re-shuffling the validation- and test data makes it highly unlikely that it will perform as well, and depending on the data used, this can have dire consequences.

A common issue faced when implementing machine learning algorithms is the existence of majority and minority classes. This is known as class imbalance and can have a detrimental impact on the effectiveness of the algorithm in question. For example, if the proportion of patients scanned for cases of breast cancer is 1% positive, there is a significant class imbalance. This is a problem when evaluating the performance of the algorithm. If it classifies every case as negative, it is still 99% accurate, which seems like a good performance, but the outcome defeats the purpose of the algorithm.[12] Another issue, known as cost imbalance, emerges in this scenario where the cost of a false negative is much higher than that of a false positive.[12]

There are numerous ways of tackling class imbalance, but it is normally done during the data pre-processing stage of a project. Popular methods for preparing the data include over- and under-sampling the minority or majority classes, respectively, or specific feature selection. [5] In other cases, machine learning algorithms have been developed specifically for the task of reducing or eliminating class imbalance.[4]

Though this project is not dedicated to solving the issue of class- and cost imbalances, it is important to explore more powerful machine learning algorithms that aim to optimize based on performance. For this reason, a neural network was implemented and run on the same data as the kNN algorithm.

A neural network is a machine learning algorithm that inputs data points into a network of nodes to compute a prediction as a single output. Each node receives data, multiplied by some weight and added bias using the equation

$$w_i \cdot x_i + b$$

To avoid a situation where each output value has a mere linear relation to the output values of the next layer, a non-linear "activation function" is embedded in each subsequent node. This changes the output in such a way that value is gained from adding multiple layers of nodes after each other. The final output is passed through a loss function which compares it to the desired target value, the result of which is used to adapt the weights in the network for the model to fit the data better.[15]

5.2 Implementation

To implement the neural network in Python, we use pandas and NumPy.[2][11][17] Pandas mainly make it easier to import and prepare the data. Numpy is used to perform more complex computations on our data in an efficient manner, like the calculation of the dot product or multiplying matrices. Since the neural network will need to perform many computations on a large amount of data points, even small improvements in speed will be greatly beneficial for the overall runtime of the algorithm.

The neural network is written as a separate class, which is initialized with a network size, a learning rate, a dropout probability, a set of training data and a set of unclassified data. Note that for the training data, the class labels have been translated from low, medium, and high into 0, 0.5, and 1. This allows us to compute a loss function and use this to optimize the weights in the network. The loss function we use is the Mean Squared Error or MSE:

$$MSE = \frac{\Sigma(y_{i,expected} - y_{i,predicted})^2}{n},$$

Here, n is the number of instances in the data.

For the activation function in the hidden layers of the network, we choose to use the sigmoid function:

$$\sigma = \frac{1}{1 - e^{-x}},$$

In order to perform the back-propagation, we need the derivatives of these functions. These are:

$$dMSE = \frac{\Sigma(2 * (expected - predicted))}{n}$$

And

$$\frac{d\sigma}{dx} = \sigma * (1 - \sigma)$$

First, we implement the auxiliary functions needed to compute the full network and back-propagate through it. Next, we generate the initial weights. This is done using the size of the network, generating weights for each layer separately, and then looping over all the layers. This policy gives a weight for each possible connection between two nodes in neighboring layers, so the network will be fully connected.

We propagate through the network by iterating over all the layers starting at the data input and computing the dot product between the weights and the input nodes. The values for the new nodes are plugged into the sigmoid function and will be fed as input into the next layer of neurons. In the end, all neurons, are written to the class variables, as well as the final output.

The back-propagation part of the code was based on the pseudo-code found in the course material.[14] For the back-propagation function, the deltas of the neurons are iteratively calculated and saved in a multidimensional array. These deltas are then used in a dot product with the weights, in order to calculate the difference in weights. This difference is multiplied by the learning rate and is then added to the weights variable present in the class.

Finally, we create the training and testing function of the network, along with a function to get classes from numbers (so we can actually output a class instead of a seemingly arbitrary number). The latter function is rather straightforward; take the numbers generated by the network, and translate them to the corresponding class in the data.

The other two functions are very similar. The training function iterates a pre-specified number of times, each iteration computing the loss function and then propagating backward, altering the weights. The testing function works similarly, but without back-propagating. The testing function finally outputs a list of predicted outputs and a list of expected outputs.

Due to suffering from a mode collapse, predicting almost the same value for every instance, as well as high bias, an attempt was made to resolve this. Mode collapse is commonly solved using a technique called dropout. A dropout function transforms the weight so that they have a random probability of being replaced with 0. This is supposed to prevent overfitting as well as mode collapse, as the neural network is forced to learn from the other weights to provide a proper model of the data. In addition to this, some model runs were done using the ReLU activation function instead of the sigmoid, in an attempt to solve the two aforementioned problems. In the end, the sigmoid function was still used as it is preferr

5.2.1 Important Note

It is important to note that, both due to time constraints and a lack of a complete understanding, the neural network implemented by the group does not function at full capacity. As of writing this, it only guesses one value for all inputs, which seems to indicate that the network is suffering from mode collapse, as well as high bias, getting a very significant portion of the classifications wrong. Thus, in order to still get results for the question, the popular machine learning library TensorFlow was used.

5.3 Analysis

When running the neural network implementation on the data, the results are significantly worse than anticipated. After training the model, it is only 35% accurate. Because of this, the decision was made to use the TensorFlow library to build a new neural network in order to produce useful data.[7] The new model consists of 4 hidden layers of sigmoid activation functions and a final layer that uses a softmax activation for the classification of the data. The number of nodes in each layer is 3, 5, 7, 5, and 3, respectively. Nevertheless, this model is only 41% accurate on the training data and validation data and 39% accurate on the test data. Upon further inspection of the raw data, it can be seen that the majority of classes in each subset is ”low.” Furthermore, the accuracy of the model coincides perfectly with the distribution of the low class in each subset. This raises suspicions that the model classifies every instance as low because this doing so is more accurate than classifying them as medium or high.

6 Results

The intent of this research was to compare the performance of a k-nearest neighbors algorithm and a neural network. The data set we used to investigate this was a database of milk quality based on seven features. Due to its higher level of complexity and the possibility of learning non-linear relations, the neural network was expected to outperform the kNN algorithm. However, due to unforeseen complications, the performance of the neural network was subpar, and the kNN algorithm performed far beyond any expectations. One of the more probable reasons for the lack of performance by the neural network is that the data would be very uniformly distributed over the feature space. The data consisting of small, intermingled clusters can be a reason for the rapid rise in error for the kNN algorithm as k increases. Another possible reason is the relatively low number of instances in the dataset. It could be the case that the neural network requires either a more sophisticated model or more instances to

With the results yielded from this experiment, it is difficult to make conclusive comparisons on the performance of the algorithms. That being said, what can be concluded is that this implementation of both algorithms shows that the optimized kNN model outperforms the neural network, likely because of its ”lazy” strategy. This experiment has shed light on the abilities of a simple algorithm such as kNN. Though it can be insufficient in many cases, it can become useful in others, largely due to its simplicity.

Bibliography

[1] T. M. Cover and P. E. Hart. “Nearest Neighbor Pattern Classification”. In: *IEEE Transaction on Information Theory* 13.1 (1967). URL: `https://ieeexplore.ieee.org/abstract/document/1053964`.

[2] Guido Van Rossum and Fred L Drake Jr. *Python reference manual.* Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[3] Kevin Beyer et al. “When Is Nearest Neighbor Meaningful?” In: *Database Theory - ICDT’99.* Berlin ; Heidelberg ; New York ; Barcelona ; Honh Kong ; London ; Milan ; Paris ; Singapore ; Tokyo: Springer, 1998.

[4] Roberto Alejo et al. “A hybrid method to face class overlap and class imbalance on neural networks and multi-class scenarios”. In: *Pattern Recognition Letters* 34.4 (2013). DOI: `10.1016/j.patrec.2012.09.003`. URL: `https://www.researchgate.net/publication/230856737_A_hybrid_method_to_face_class_overlap_and_class_imbalance_on_neural_networks_and_multi-class_scenarios`.

[5] Aida Ali, Siti Mariyam Shamsuddin, and Anca L. Ralescu. "Classification with class imbalance problem: a review". In: *Int. J. Advance Soft Compu. Appl* 5.3 (2013). URL: `https://www.researchgate.net/profile/Aida-Ali-4/publication/288228469_Classification_with_class_imbalance_problem_A_review/links/57b556d008ae19a365faff16/Classification-with-class-imbalance-problem-A-review.pdf`.

[6] Ahmad Basheer Hassanat et al. "Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach". In: *CoRR* abs/1409.0919 (2014). arXiv: `1409.0919`. URL: `http://arxiv.org/abs/1409.0919`.

[7] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[8] Nicolás García-Pedrajas, Juan A. Romero del Costillo, and Gonzalo Cerruela-García. "A Proposal for Local k Values for k-Nearest Neighbor Rule". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.2 (2017). URL: `https://ieeexplore.ieee.org/abstract/document/7368188`.

[9] Amit Pandey and Achin Jain. "Comparative analysis of KNN algorithm using various normalization techniques". In: *International Journal of Computer Network and Information Security* 11.11 (2017), p. 36.

[10] Batta Mahesh. "Machine learning algorithms-a review". In: *International Journal of Science and Research (IJSR).[Internet]* 9 (2020), pp. 381–386.

[11] The pandas development team. *pandas-dev/pandas: Pandas*. 2020. DOI: `10.5281/zenodo.3509134`. URL: `https://pandas.pydata.org/`.

[12] Peter Bloem. *MLVU 3.2: Model Evaluation*. Tech. rep. 2021.

[13] Peter Bloem. *MLVU 4.3: Normalization*. Tech. rep. 2021.

[14] Peter Bloem. *MLVU 4.6: Beyond Linear Models*. Tech. rep. 2021.

[15] Peter Bloem. *MLVU 6.1: Beyond Linear Models*. Tech. rep. 2021.

[16] Shrijayan Rajendran. *Milk Quality Prediction*. kaggle.com, 2022. URL: `https://www.kaggle.com/datasets/cpluzshrijayan/milkquality`.

[17] NumPy Developers. *What is NumPy?* URL: `https://numpy.org/doc/stable/user/whatisnumpy.html`. (accessed: 29.03.2023).