Comparison of Artificial Neural Network and k-Nearest Neighbour for Classification

Dennis M.C. Ideler
Computer Science Department
Brock University
di07ty@brocku.ca

***Abstract*—The goal of this comparative study was to analyze the differences and similarities in both supervised learning rules and determine if one classifier is better suited for certain classification problems. The classifiers used are (1) a feed-forward neural network with the error back-propagation learning rule, and (2) the k-nearest neighbour algorithm. The classifiers are tested on two datasets: (1) steel plates faults, and (2) handwritten digits. Preprocessing the datasets is also examined, specifically Feature Selection (FS) and normalization, and how they affect the accuracy of the diagnosis. Observed results concerning learning parameters and generalization ability as well as the role of the network architecture are explained in the analysis. The conclusion discusses several methods which can be used to enhance the classification accuracy.**

I. Introduction

Fault Diagnosis (FD) is an essential pattern recognition problem. It plays an important role in the operation and maintenance of mechanical equipment. FD has a major importance to enhance the quality of manufacturing and to lessen the cost of product testing. Artificial Intelligence (AI) techniques are often used for enhancing the accuracy of faults identification or for automating the process.

Handwritten digit recognition is a classic Machine Learning classification problem. It is an underlying problem of Optical Character Recognition (OCR), where you predict the label of each image using the classification function learned from training. Algorithms like support vector machine (SVM) and multi-layer neural networks and typically used for this type of problem. Handwritten digit recognition has many applications, such as recognizing (U.S. based) zipcodes, or recognizing input on computer tablets or touch screen devices.

II. Problem Definition

This study evaluates the performance of a vanilla back-propagation learning model in a multi-layer feed-forward artificial neural network, and of the *k*-nearest neighbour learning model, on two practical applications.

The first application is diagnosing seven commonly occurring faults of steel plates, namely: *Pastry, Z Scratch, K Scratch, Stains, Dirtiness, Bumps,* and *Other Faults*. This is done using information from 27 independent attributes of steel plates: *X Minimum, X Maximum, Y Minimum, Y Maximum, Pixels Areas, X Perimeter, Y Perimeter, Sum of Luminosity, Minimum of Luminosity, Maximum of Luminosity, Length of Conveyer, Type of Steel A300, Type of Steel A400, Steel Plate Thickness, Edges Index, Empty Index, Square Index, Outside X Index, Edges X Index, Edges Y Index, Outside Global Index, Log of Areas, Log X Index, Log Y Index, Orientation Index, Luminosity Index,* and *Sigmoid of Areas*.

The second application is recognizing handwritten digits from 0 to 9, using 256 attributes per image, which is simply the binary pixel data of a 16×16 image. These images have gone through various phases of preprocessing.

The datasets used in this study are hosted at the University of California at Irvine (UCI) machine learning repository [1] [2].

III. Feature Selection

Pudil et al. describe the feature selection (FS) problem as the process of selecting which features from the original set should be included when designing the classifier. The FS problem can be defined as follows: Given a set of D features, select a subset of size d ($d \leq D$), where d represents the desired number of features. The criterion function for subset evaluation is usually based on some measure of distance or dissimilarity between distributions. Without any loss of generality, consider a higher value of criterion function to indicate a better feature subset. Then seek a combination of features for which the criterion function is maximized [4].

Almost all real world applications of classification learning come with many features. Irrelevant features that can cause noise or extra dimensionality are present in the set of features. By removing the most irrelevant and redundant features from the dataset with the aid of FS techniques, a subset of predictive features remain, which can improve the performance of learning models. Selecting a set of features which is optimal or sub-optimal for a given classification task is one of the central problems in machine learning.

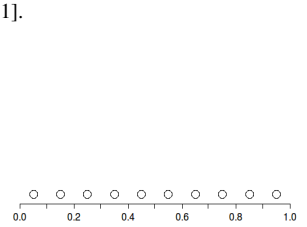
Feature Selection for Performance Improvement

FS can improve performance in the following ways:

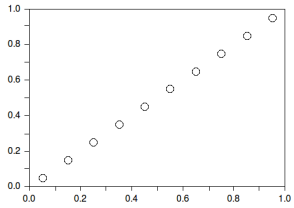
1) Reducing the effect of the "curse of dimensionality".
2) Enhancing generalization capabilities.
3) Speeding up the learning process.
4) Improving model and data interpretability.

Curse of Dimensionality: The curse of dimensionality (also known as the Hughes effect) is the problem caused by the exponential increase in volume associated with adding extra dimensions to a (mathematical) space [5]. Each feature adds

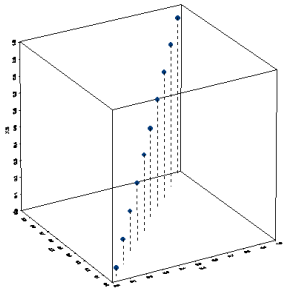
Fig. 1. Search space coverage of 10 sample points in (a) 1-dimensional, (b) 2-dimensional, and (c) 3-dimensional search spaces along the unit interval [0,1].



(a) One feature: 10% coverage by single sample point.



(b) Two features: 1% coverage by single sample point.



(c) Three features: 0.1% coverage by single sample point.

a new dimension to the search space. Figure III shows an example of the exponential increase in search space volume when features are added. FS can reduce the effect of this "curse" by reducing the number of features used which in turn reduces the dimensionality of the search space.

Generalization Capability: The main purpose of using learning models is to generalize. The ability to generalize allows for classification of unseen data from the same class as the learning data. This requires prior knowledge [6], [8]: the learning data or training examples. For any practical application, one should know which features are relevant inputs; including all imaginable inputs or noisy inputs is impractical. The cases to be generalized need to bear some resemblance to the training cases. That leads to the following necessary condition to make generalization possible: the inputs should contain sufficient information pertaining to the target, so that there exists a mathematical function relating correct outputs to inputs with the desired degree of accuracy. FS can enhance generalization capability by selecting the relevant inputs.

Quicker Learning: In optimal FS, an exhaustive search of all possible subsets of features would need to be done. This is impractical if there are many features available and a small timeframe. For practical FS, satisficing is done, that is, a search for a satisfactory subset of features rather than an optimal subset via an expensive procedure. Reducing the possible subsets of features reduces the search space. Removing irrelevant and noisy features reduces the dimensionality and thus the search space. The result is that less time is spent learning. It should be noted however that finding a relevant subset of inputs and collecting enough training data often takes far more time and effort than training the classifier.

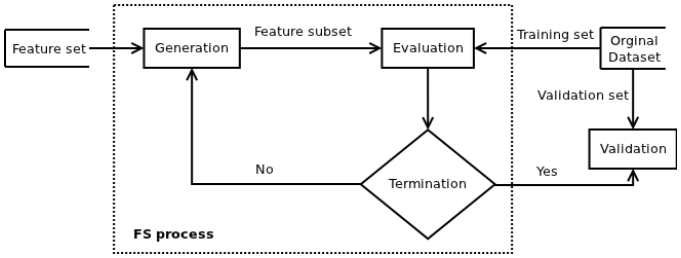


Fig. 2. High level description of feature selection methods.

Improve Interpretability: Interpretability reflects the ease to understand, properly uses, and analyze the data. By reducing noise and irrelevant parts of data, FS can give valuable insight to the data used. It helps indicates which features are important and how they relate with each other.

Common Steps in Feature Selection

There are typically four common steps in the FS process:

1) Generation
2) Evaluation
3) Termination
4) Validation

The flowchart in figure 2 gives a visualization of these key concepts.

Search Methods for Feature Selection

The Feature Selection Toolbox 3 (FST3) library was used in this project. The following sub-optimal searches were used:

- Wrapper-based (kNN) feature selection with Floating Search
- Retreating Sequential Search
- Generalized sequential feature subset search
- Monte Carlo - random feature subset search

The FST documentation [9] describes the search scenarios:

1) Wrapper-based (kNN) FS with Floating Search:

Floating Search (SFFS) can be considered one of the best compromises between versatility, speed, ability to find close-to-optimum results, and robustness against overfitting. If in doubt which feature subset search method to use, and the dimensionality of your problem is not more than roughly several hundred, try SFFS. In this example features are selected using the SFFS algorithm and 3-NN wrapper classification accuracy as FS criterion. Classification accuracy (i.e, FS wrapper criterion value) is estimated on the first 50% of data samples by means of 3-fold cross-validation. The final classification performance on the selected subspace is eventually validated on the second 50% of data.

2) Retreating sequential search:

Retreating Search (SFRS) is closely related to SFFS, but more thorough and slower. It evaluates more candidate subsets due to more excessive backtracking. This may be advantageous especially in the case of SFRS, which thus evaluates

more of the lower-dimensional potential subset solutions.Features are selected using the SFRS algorithm and 3-NN wrapper classification accuracy as FS criterion. Classification accuracy (i.e, FS wrapper criterion value) is estimated on the first 50% of data samples by means of Leave-One-Out estimation. The final classification performance on the selected subspace is eventually validated on the second 50% of data.

3) Generalized sequential feature subset search:

All sequential search algorithms can be extended to operate in a 'generalized' setting. In each step of a generalized sequential search algorithm, one best feature is not added to the current subset, nor is one worst feature removed from the current subset; instead, tuples of features are considered. Searching for such group of features that improves the current subset the most when added (or such that degrades the current subset the least when removed) is more computationally complex but increases the chance of finding the optimum or a result closer to optimum (nevertheless, improvement is not guaranteed and in some cases the result can actually degrade). The size of the tuple is set by user; the higher the value, the slower the search (time complexity increases exponentially). Note that setting the size equal to the number of all features would effectively emulate the operation of exhaustive search. Features are selected using the generalized (G)SFFS algorithm (G=2) and 3-NN wrapper classification accuracy as FS criterion. Classification accuracy (i.e, FS wrapper criterion value) is estimated on the first 50% of data samples by means of 3-fold cross-validation. The final classification performance on the selected subspace is eventually validated on the second 50% of data.

4) Monte Carlo:

Pure random search does not give any guarantees of the optimality of results. It is unlikely to yield better results than any of the sub-optimal or optimal FS methods, except in isolated cases. The advantage of random search is nevertheless its quick initial 'convergence'; often it is capable of quickly revealing a solution that is only marginally worse than solutions that would take considerably longer to find using more advanced methods. In this example features are selected using the Monte Carlo algorithm and 3-NN wrapper classification accuracy as FS criterion. The stopping condition is specified as a time limit in seconds. Classification accuracy (i.e, FS wrapper criterion value) is estimated on the first 50% of data samples by means of 3-fold cross-validation. The final classification performance on the selected subspace is eventually validated on the second 50% of data.

IV. Classification

Classification can be described as "the process of finding a model (or function) that describes and distinguishes data classes or concepts, for the purpose of being able to use the model to predict the class of the objects whose class label is unknown." [10] Classification has a predictive nature – for a given set of features, the goal is to attempt to predict the value of another feature. A classifier model is used to classify the actual data into defined classes. Ultimately, patterns need to exist in the data that can be exploited.

Many methods can be applied to classification. One option would be a direct algorithm. Direct algorithms systematically solve a problem, possibly with heuristics. For some problems, such as sorting, direct algorithms are almost trivial. For other problems they can take a lot of time and effort to design and/or run, or they are yet to be discovered (and may never be if they do not exist for the problem at hand). Direct algorithms require a lot of domain knowledge, and they usually try to find the optimal solution. Search and learning models such as ANN on the other hand require little domain knowledge. They are viable options when there is insufficient time to gather or apply domain knowledge. Using search as an alternative allows us to consider alternative solutions. It satisfices by quickly finding a solution that is sufficient for our needs, rather than an expensive but optimal solution. Search methods and learning models are also robust and reusable, direct algorithms are not.

V. Feed-Forward Neural Networks

Artificial neural networks (ANN), or simply neural networks (NN), are adaptive statistical models based on an analogy with the structure of the brain[1]. ANN are basically built from simple units called *(artificial) neurons*[2]. These units are interlinked by a set of weighted connections. Learning is usually accomplished by adjusting the weights. The units are organized into layers. A network will usually have several layers, where the first layer is called the *input* layer, and the last one is called the *output* layer. Any intermediate layers are called *hidden* layers (see Fig. 3). The information to be analyzed is fed to the first layer and then proceeds to the next layer until the last layer. The goal of the network is to learn some association between input and output patterns.

Feed-forward networks are a subclass of acyclic networks. They are the most basic form of ANN and are among the most common ANN in use. A feed-forward network only contains forward paths[3] to the layer directly succeeding the current layer. A feed-forward network can be either single-layer (no hidden layers), or multi-layer (there exists at least one hidden layer). This project consists of a multi-layer feed-forward network.

Error-correction learning is used with supervised learning systems. It is the technique of comparing the actual output to the desired or expected output, and using that error to train

[1]Biological Neural Nets (BNN) are the natural "equivalent" of the ANN.
[2]Sometimes called *nodes*, *neurodes*, *processing elements (PEs)*, or *units*.
[3]Data only flows in one direction; there are no cycles.

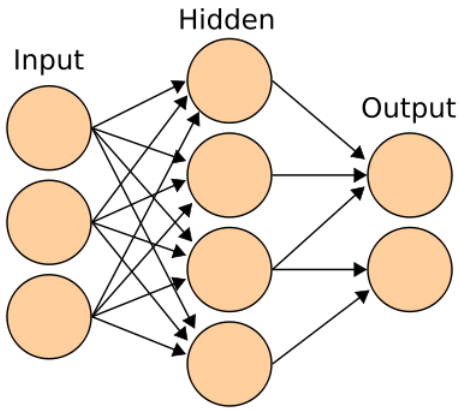


Fig. 3. Example of a 3-4-2 feed forward ANN [11]

TABLE I
ANN STRUCTURES

Steel dataset	Steel dataset w/ FS	Digits dataset
27-15-7	6-10-7	256-100-10

the learning model. In the case of the ANN, this happens by adjusting the weights with help of the error values. Error-correction learning attempts to minimize this error signal with each training iteration. A popular learning algorithm is the back-propagation algorithm.

VI. LEARNING SYSTEMS

A. Neural Network

1) Target Function: The target or objective function was to minimize the network error (or to maximize accuracy). The type of error used is simply the sum of squared output errors.

2) Representation: The input is represented in a vector of training examples and a vector of testing examples. Each example has its label as the last value. Each attribute value in an example is mapped to an input neuron.

3) System Structure: The ANNs used are all 3 layers (input layer, hidden layer, output layer). See table I for more information. The logistic function was used in all experiments.

4) Learning Algorithm: The main problem with early neural networks was that they could only deal with linear problems. Researchers knew however that they could overcome this limitation by adding one or more hidden layers. The problem was that there was no way to automatically adjust the weights in the hidden layer in case of errors; there was no learning algorithm. This caused the decline in interest in neural networks. The revival was due to error back-propagation[4].

Back-propagation, or backprop for short, refers to the "back-propagation of error". Strictly speaking, backprop refers to the method for computing the gradient of the error or cost function with respect to the weights for a feed-forward network. This is a straightforward but elegant and efficient application of the chain rule of elementary calculus [12] [13].

Extending its definition, *backprop* refers to a training method that uses back-propagation to compute the gradient. A *backprop network* is a feed-forward network trained by back-propagation.

In short, backprop networks consist of:

1) multiple layers of non-linear units
2) computation of an error signal[5] using the rate of change (derivative) of the non-linear function
3) back-propagation of an error signal
4) estimation of an error signal by the hidden units

Just like a linear unit, the non-linear unit computes its activation by summing all the weighted activations it receives. However, unlike the linear unit, it will create a response by putting this sum through a non-linear transfer function (also known as an activation function) which is mentioned in step 2 above.

If threshold is used, then the threshold value is added to the sum before going through the transfer function. If bias is used, then a "pseudo input" is created that has a constant output value of 1 and has weighted connections. There is one pseudo input for each hidden layer, and the output layer also has one. Bias is used as a safety measure in case all or most of the values of an input pattern are zero, then the weights in the network would never change and the network would not be able to learn.

Several transfer functions exist. For mapping an output value to a range of [0, 1], the *logistic function* is used. Its formula is

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

For mapping an output between the range of [-1, 1], the *hyperbolic tangent function* is used. Its formula is

$$f(x) = tanh(x) \tag{2}$$

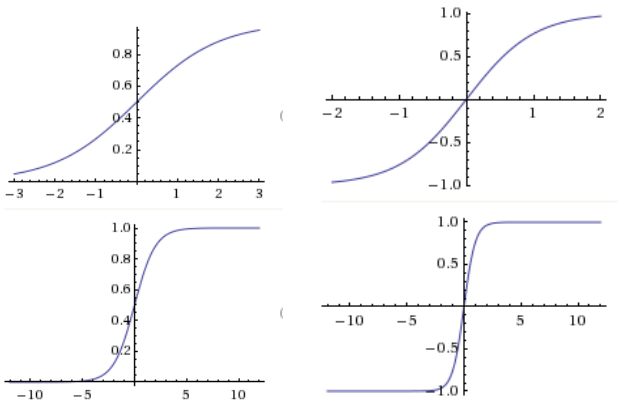
These two functions are also called "squashing functions" because they have bounded ranges, which is visible from their sigmoid curves (see Fig. 4). For hidden units, sigmoid activation functions are preferable to threshold functions [15] (such as the stepwise function which gives an output of either 1 or 0), because they are easier to train. With sigmoid units, a small change in the weights can produce a noticeable change in the outputs, which makes it possible to tell whether that change in weights was good or bad. With threshold units, a small change in the weights will often produce no change in the outputs.

A common approach is to use the tanh function for the hidden units and the logistic function for the output units. For binary targets (0/1), the logistic function is an excellent choice, which is why it is often used at the output layer.

[4]Back-propagation was around since the 1950s but only gained acceptance in the mid 1980s

[5]A backprop network uses supervised learning to calculate the difference between the output and expected output.

Fig. 4. Curvatures of the logistic and tanh functions on different x-ranges.



(a) Logistic function (b) Hyperbolic tangent function

The algorithm for back-propagation (using incremental training) can be broken down into the following high-level steps:

1) Initialize the network with small random weights [6]
2) Present an input pattern to the input layer
3) Feed the input pattern forward through the network to calculate its activation value (i.e. generate a forward flow of activation)
4) Take the difference between desired output and the activation value to calculate the network's activation error (i.e. compute the error term)
5) Adjust the weights feeding the output neuron to reduce its activation error for this input pattern
6) Propagate an error value back to each hidden neuron that is proportional to their contribution of the network's activation error
7) Adjust the weights feeding each hidden neuron to reduce their contribution of error for this input pattern
8) Repeat steps 2 to 7 for each input pattern in the input collection
9) Repeat step 8 until desired epochs reached or until the network is trained to satisfaction

At a very high level, the backpropagation algorithm (using incremental training) is as follows. Let F_A be the input layer, F_B the hidden layer, and F_C the output layer.

An advantage of backpropagation is that the algorithm is very popular for supervised learning in ANN because it has been around longer than other comparable algorithms and it is simpler than others. A disadvantage of backpropagation is that it learns at a very slow pace from examples (when not combined with other helping strategies). There are two major identified problems that contribute to this slow pace.

The first is the *step-size problem* [14] which identifies the infinitesimal steps that backpropagation makes during its learning, due to the algorithm computing only the partial

[6]Note that if *all* the weights are zero, then the error term is zero and that means no learning because the correction of the weights will then be equal to 0.

Algorithm 1 Back-propagation for a 2-weight-layer ANN

```
Initialize connection weights {usually random}
repeat
  for every instance in the training set do
    for node_i ∈ F_A do
      node_i ← att
    end for
    Forwardpropagate a training pattern's input from F_A to
    F_B, and F_B to F_C, to generate the output activations.
    Compute the error for the output layer and hidden
    layer.
    Backpropagate the output activations using the training
    pattern's target and the error in order to generate the
    deltas of all F_C and F_B neurons, and adjust their
    connections weights.
  end for
until termination criteria is met
```

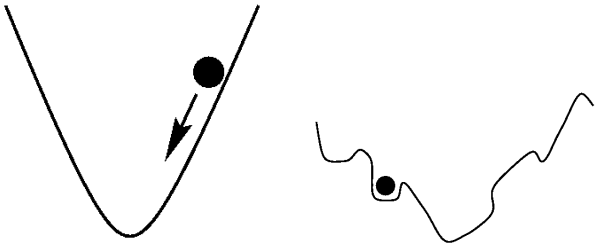
first derivative of the overall error function with respect to each weight in the network. To choose a reasonable step size, the algorithm needs to not only know the slope of the error function, but also its curvature in the vicinity of the current point in weight space[7]. To deal with the step-size problem, many solutions have been proposed, such as: momentum (which is good for summarizing the slope of the error surface during early stages in the computation); explicitly computing an approximation to the second derivative of the error function at each step and using that information to guide the speed of descent; and the increasingly popular quickprop (which computes the partial first derivative like in backprop, but instead of a simple gradient descent it uses a second-order method which is related to Newton's method to update the weights).

The second is the *moving target problem* [14]. Each hidden neurode of the network is trying to evolve into a feature detector that will play a role in the network's overall performance, but this is difficult because all other neurodes are changing simultaneously. The hidden neurodes cannot communicate with each other directly, each unit only sees its inputs and the error signal propagated back to it from the network's outputs. The error signal defines the problem that the unit is trying to solve but this problem is changing constantly. This makes it difficult for the units to take on a useful role quickly, instead there is a lot of blaming and changing among the units and it takes a long time to settle down.

5) Improvements & Modifications: Momentum is used by the ANN. In gradient descent we start the search on the fitness landscape at some point given by the error function defined over the weights. The goal is to reach the global optimum of the function. Ideally the landscape has no local optima, however, for real problems the error surfaces are typically complex and consist of many local optima.

There are several methods to escape local minima, one of

[7]Which would be knowledge of its higher-order derivatives



(c) Example of an ideal landscape (d) Example of a realistic landscape

those is momentum. Momentum simply adds a fraction, m, of the previous weight update to the current one. This is probably the most popular extension of the backprop algorithm. When the gradient keeps changing direction, momentum will smooth out the variations. For that reason, momentum is useful when the network is not well-conditioned.

With momentum, m, the weight update at a given time, t, becomes

$$\Delta w_{ij}(t) = \mu_i \ \delta_i \ y_j + m \ \Delta w_{ij}(t-1) \tag{3}$$

B. k-Nearest Neighbours

1) Target Function: The k-NN in this project uses a discrete target function, calculating the most common class of the nearest training examples (as opposed to the mean value with a continuous target function).

2) Representation: The same representation is used as for the ANN. The difference is in how the instances are mapped to the classifier. This k-NN classifier calculates the distance from each training example to the current testing example, then chooses the k nearest.

3) System Structure: The choice of k is very important. If k is too small, then the result can be sensitive to noise points. On the other hand, if k is too large, then the neighborhood may include too many points from other classes. For these experiments, $k = 3$ was used, which is a common value for k.

Another important consideration is the distance measure used. For these experiments, the squared Euclidean distance is used.

4) Learning Algorithm: K-NN finds a group of k objects in the training set that are closest to the test object, and bases the assignment of a label on the predominance of a particular class in this neighborhood. There are three key elements of this approach:

- a set of labeled objects (e.g. a set of stored records)
- a distance or similarity metric to compute distance between objects
- the value of k (the number of nearest neighbors)

To classify an unlabeled object, the distance of this object to the labeled objects is computed, its k-nearest neighbors are identified, and the class labels of these nearest neighbors are then used to determine the class label of the object. See algorithm 2.

Algorithm 2 K-NN

```
Determine k {number of nearest neighbours}
for query_i ∈ testing set do
  for record_i ∈ training set do
    distance ← d(query_i, record_i)
  end for
  Sort neighbours based on distance
  for k nearest neighbours do
    Determine majority class
  end for
  classification = majority class
end for
```

TABLE II
STEEL PLATES FAULTS DATASET INFO

Data Set Characteristics:	Multivariate
Number of Instances:	1941
Area:	Physical
Attribute Characteristics:	Integer, Real
Number of Attributes:	27
Date Donated:	2010-10-26
Associated Tasks:	Classification
Missing Values?	No

5) Improvements & Modifications: The squared Euclidean distance was used instead of the Euclidean distance. It is is similar to the Euclidean distance but does not take the square root of the result. This is possible because we are using distances for comparisons, we do not need the actual distance, just a relative measure. This also gives a speed improvement, as calculating the square root is computationally expensive.

Distance weighting was implemented but not used in the experiments.

VII. EXPERIMENT SETUP & DATA PREPARATION

Steel Plates Faults Dataset [1]

A brief description of both datasets was given in the Problem Definition section. This is a dataset of steel plates' faults, classified into 7 different types. Each instance of the dataset owns 27 independent variables and one fault type. The original purpose of the dataset was to train machine learning models for automatic pattern recognition. This dataset was generously donated to the UCI Machine Learning repository by the Semeion Research Center in Italy.

The dataset was split 75% for training and 25% for testing.

Table II shows more information about the dataset, and table III shows the class distribution.

Semeion Handwritten Digit Dataset [2]

This dataset consists of 1593 handwritten digits that were scanned from around 80 persons. Each pixel of each image was scaled into a boolean (1/0) value using a fixed threshold: setting to 0 every pixel whose value was equal or under the value 127 of the grey scale an setting to 1 each pixel whose

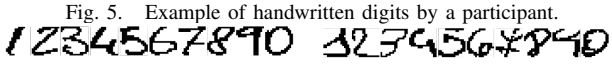
TABLE III
STEEL PLATES FAULTS DATASET CLASS DISTRIBUTION

Class #	Fault	# of cases
1	Pastry	158
2	Z Scratch	190
3	K Scratch	391
4	Stains	72
5	Dirtiness	55
6	Bumps	402
7	Other faults	673

TABLE IV
HANDWRITTEN DIGITS DATASET INFO

Data Set Characteristics:	Multivariate
Number of Instances:	1593
Area:	Computer
Attribute Characteristics:	Integer
Number of Attributes:	256
Date Donated:	2008-11-11
Associated Tasks:	Classification
Missing Values?	No

Fig. 5. Example of handwritten digits by a participant.



(a) Written with accuracy (normal). (b) Written with no accuracy (fast).

original value in the grey scale was over 127. Finally, each binary image was then scaled again into a rectangular 16×16 box in a gray scale of 256 values. It follows that each record represents a handwritten digit of 256 attributes.

Each person wrote on a paper all the digits from 0 to 9, twice. The commitment was to write the digit the first time in the normal way (trying to write each digit accurately) and the second time in a fast way (with no accuracy). See Fig. VII.

The dataset was created by Tactile Srl[8], a computer vision company in Italy who donated it in 1994 to the Semeion Research Center for machine learning research, who in turn donated it to the UCI Machine Learning repository.

Table IV shows more information about the dataset, and table V shows the class distribution. Note that even though each person wrote the same amount of digits, the class distribution is just slightly off from being consistent. This lost data is likely due to errors such as bad scans or corrupt data.

The dataset was split 75% for training and 25% for testing.

Preprocessing of Data

Simplification of Datasets: The first step of preprocessing was converting the data to be compatible with the system it will run on. This involved converting it from comma-separated values to space-separated values, and simplifying the class data. If there were n classes, each record in the dataset would label the class at the end of the record using n binary

[8]http://www.tattile.it

TABLE V
HANDWRITTEN DIGITS DATASET CLASS DISTRIBUTION

Class #	Digit	# of cases
1	0	161
2	1	162
3	2	159
4	3	159
5	4	161
6	5	159
7	6	161
8	7	158
9	8	155
10	9	158

values. For example: out of 7 classes, class 2 would look like $0\,1\,0\,0\,0\,0\,0$. The simplification would convert these n binary digits into a single digit (the above example would then simply become 2 after simplification).

Data Normalization: Rescaling, normalizing, or standardizing[9] the data before presenting it as input to the learning model is a common preprocessing step. It is not required that the inputs be in the interval [0,1], although there are often benefits, such as preventing irrelevant (or less relevant) attributes from being overpowering.

Normalizing either input or target variables tends to make the training process better behaved by improving the numerical condition of the optimization problem and ensuring that various default values involved in initialization and termination are appropriate [16]. Rescaling targets can also affect the objective function. However, normalization of cases should be approached with caution because it discards information. If that information is irrelevant, then standardizing cases can be helpful. If that information is important, then standardizing cases can be hurtful.

For the classifiers used in this study, preprocessing was done before presenting the inputs to the learning model by normalizing the variances of the input values to a range of [0,1] using equation 4.

$$x = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4}$$

This was necessary due to the range of certain features. Without rescaling the input data, certain inputs would contribute a lot more to the learning model than others. The contribution of an input depends heavily on its variability relative to other inputs. For example, if one input has a range of 0 to 1, while another input has a range of 0 to 1,000,000, then the contribution of the first input is comparatively so minuscule that it will be swamped by the second input. The range of an input also determines its weight. Determining optimal feature weights is an active area of study as it can improve classification accuracy. If a certain feature is more important than the others and thus needs to contribute more, it

[9]The definitions for all three are slightly different but they are more or less used interchangeably.

can be weighted through rescaling. So it is essential to rescale the inputs so that their variability reflects their importance, or at least is not in inverse relation to their importance. It is common to standardize each input to the same range or the same standard deviation. If some inputs are known to be more important than others, it may help to scale the inputs such that the more important ones have larger variances and/or ranges.

Feature Selection: Almost all realworld applications of classification learning come with many features. Irrelevant features that can cause noise or extra dimensionality are present in the set of features. By removing the most irrelevant and redundant features from the dataset, and keeping a subset of predictive features, the performance of learning models can be improved. Selecting a set of features which is optimal or near-optimal for a given classification task is one of the central problems in machine learning.

FS was only performed for the steel plates faults dataset. It was attempted for the handwritten digits dataset, but had to be canceled due to time constraints. Because the digit dataset is quite large (many records and attributes), it ran for several days without completing.

Several FS search methods were used (described in the previous FS section) to find feature subsets, but only the best feature subset found was used. In this case, the best feature subset found is $\{6, 7, 9, 10, 11, 14\}$ which are the attributes $\{X\_Perimeter, Y\_Perimeter, Minimum\_of\_Luminosity, Maximum\_of\_Luminosity, Length\_of\_Conveyer, Steel\_Plate\_Thickness\}$. It was found with the Retreating Sequential Search (yielding the highest criterion value of 0.749226).

Data Format

The data is split up into a training set and a testing set based on a user-defined splitting ratio. An attempt is made to use an equal amount of every label for the training set, to avoid over or under training a certain class. The remaining data is then used for the testing set.

The training set is randomly shuffled before it is loaded into the ANN, not for the *k*-NN. This is to prevent the classifiers from being biased towards a certain class since the datasets are originally ordered by class type.

Although the class information is known, the class labels are removed as a feature during training, otherwise prediction would be trivial.

Parameters

Selection of the parameter values was based on general rules of thumb, empirical evidence, and experimentation (i.e. trial and error). Parameters used can be seen in tables VI, VII, and VIII. Each run used a random number generator seed equivalent to its run number (e.g. run 1 used seed 1).

VIII. Results and Discussion

Table IX shows the averaged results of the classification accuracy from both classifiers on all datasets, gathered over 30 runs. Note that the ANN accuracy shown in the table is from the testing phase. The ANN training data for all datasets is plotted in figures 6 and 7.

It is clear that the simple, yet powerful k-NN outperforms the complex ANN on all datasets used. That leads to the conclusion that a simple distance measure as a similarity metric works better for these datasets than a complicated blackbox search for finding suboptimal weights to exploit any patterns in the data.

The digits experiment converges very early, at around 10 epochs, and does so almost instantly. Any extra epochs make a miniscule difference in the accuracy. The faults experiment starts to converge around 10 epochs as well, but takes much longer to fully converge, meaning that the extra epochs come

TABLE VI
Steel Faults Dataset

Input units	Hidden units	Output units
27	15	7
Learning rule	Momentum	Learning rate
backprop	0.9	0.2
Lower weight range	Upper weight range	Epochs
-0.05	0.05	200
Activation function	Training/testing ratio	Choice of k
logistic	75% : 25%	3

TABLE VII
Steel Faults Dataset with FS

Input units	Hidden units	Output units
6	10	7
Learning rule	Momentum	Learning rate
backprop	0.9	0.2
Lower weight range	Upper weight range	Epochs
-1.0	1.0	200
Activation function	Training/testing ratio	Choice of k
logistic	75% : 25%	3

TABLE VIII
Handwritten Digits Dataset

Input units	Hidden units	Output units
256	100	10
Learning rule	Momentum	Learning rate
backprop	0.9	0.2
Lower weight range	Upper weight range	Epochs
-1.0	1.0	200
Activation function	Training/testing ratio	Choice of k
logistic	75% : 25%	3

TABLE IX
Averaged accuracy results

	Steel Plates Faults	Steel Plates Faults Subset	Handwritten Digits
ANN	63.81	53.69	91.07
KNN	99.86	100.00	96.84

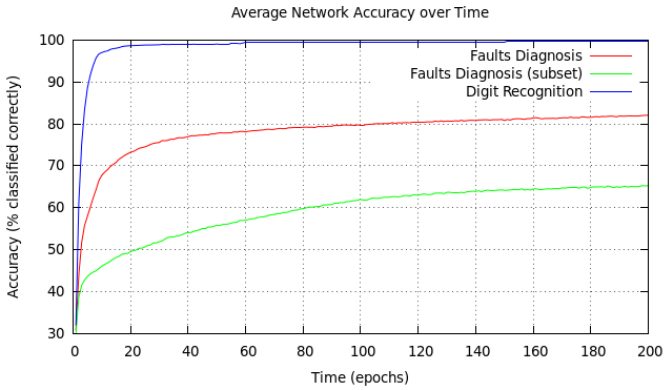


Fig. 6. Average Network Accuracy

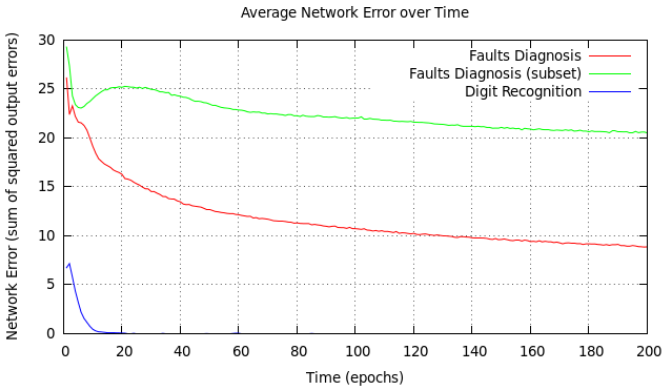


Fig. 7. Average Network Error

in handy. The faults subset is the slowest to converge. Only at around 180 epochs does it seem to have fully converged.

The training accuracies from the experiments on the faults dataset are much lower than that of the digits dataset. The digits training accuracy reaches approximately 99% accuracy, whereas the faults training accuracy reaches approximately 83% accuracy, and the faults subset training accuracy reaches approximately 65% accuracy. That leads to the most likely conclusion that the faults dataset is significantly more difficult to learn than the digits dataset. This could be due to various reasons, such as non-optimal or non-suboptimal parameters were used, not enough preprocessing of the data was done (data is still too complex), or patterns in this dataset are hard to detect. The same faults dataset have a high accuracy with k-NN, which means that there are patterns in the data, but perhaps they are just too difficult to accurately detect with a vanilla ANN. A large factor could be that the class distribution for the faults dataset is very unbalanced (see table III). In fact, just three classes (3, 6, and 7) make up approximately 75% of the entire dataset. This poor class distribution can lead to poor results during the learning phase due to lack of training examples for certain classes and an overpowering of other classes. The k-NN classifier avoids this issue because it has no real learning phase, that is, it does not build a learning model (which also means that new or modified examples do not affect its behaviour). This lack of a model serves as a benefit of eager learning in this case.

Another surprising observation is that the experiment that used FS did worse on the ANN than the one that did not use FS, however, the same experiment with k-NN shows that the FS did make a positive difference (although very slightly). The possibility exists that the feature subset used is lacking sufficient relevant attributes for the ANN to perform well. Note however that the ANN parameters were not the same when experimenting with FS and without, and this is also worthy of further investigation.

IX. Comparison of Learning Systems

The lazy learner (k-NN) had a very fast training phase, and could learn complex target functions. However, its bottle neck is the query time, and the possibility to be fooled by irrelevant attributes (especially if unweighted). KNN classification is an easy to understand and easy to implement classification technique. Despite its simplicity, it can perform well in many situations.

The eager learner (ANN) had a much slower training phase. An advantage is that the target function is approximated globally during training, thus requiring much less space than a lazy learning system. Eager learning systems typically also deal much better with noise in the training data, though unproven in these experiments. ANN is a much more complex classifier, but as shown, added complexity is not always useful.

Both classifiers benefit from reducing the curse of dimensionality.

X. Conclusion & Future Work

The k-NN classifier outperformed the ANN classifier on both datasets. This does not mean that k-NN is universally better, just that it is more suitable for the problems presented in this case.

The attempt at FS lead to an unexpected result. More investigation has to be done to determine why exactly the ANN performed worse with a feature subset, compared to the original feature set. An area explore is to try feature extraction (FE) instead of FS. This means that attribute values will be transformed, rather than removed. If some inputs are known to be more important than others, it may help to scale the inputs such that the more important ones have larger variances and/or ranges.

Other areas to explore include: (1) using a different target function (e.g. based off mean squared error), (2) continue the search for suboptimal parameters, (3) statistical analysis, and (4) comparison to results published in papers from others, which used the same datasets and/or classifiers.

More information, including the source code, can be found at http://dideler.github.com/ann-vs-knn.

References

[1] Steel Plates Faults dataset donated by Semeion, Research Center of Sciences of Communication, Via Sersale 117, 00128, Rome, Italy.

(www.semeion.it) Available via WWW: http://archive.ics.uci.edu/ml/datasets/Steel+Plates+Faults

[2] Handwritten Digits dataset donated by Semeion, Research Center of Sciences of Communication, Via Sersale 117, 00128, Rome, Italy. (www.semeion.it) Available via WWW: http://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit

[3] Ajith Abraham, *Artificial Neural Networks*. Oklahoma State University.
Available via WWW: http://www.softcomputing.net/ann_chapter.pdf

[4] P. Pudil, J. Novovicova, P. Somol, *Feature selection toolbox software package*, Pattern Recognition Letters, Volume 23, Issue 4, February 2002, Pages 487-492, ISSN 0167-8655, Available via WWW: http://www.sciencedirect.com/science/article/B6V15-44M2G3M-1/2/267fd2d7c3e453b79c3e56558ec3531f

[5] Miller, F.P. and Vandome, A.F. and McBrewster, J., *Curse of Dimensionality* (2010) VDM Publishing House Ltd.

[6] Wolpert, D.H. (1995), *The Mathematics of Generalization: The Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning*, Santa Fe Institute Studies in the Sciences of Complexity, Volume 1992, Reading, MA: Addison-Wesley.

[7] Wolpert, D.H. (1996), "*The lack of a priori distinctions between learning algorithms*," Neural Computation, 8, 1341-1390.

[8] Wolpert, D.H. (1996), "*The existence of a priori distinctions between learning algorithms*," Neural Computation, 8, 1391-1420.

[9] *Feature Selection Toolbox Usage*, http://fst.utia.cz/?fst3_usage

[10] J. Han, M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, August 2000.

[11] http://upload.wikimedia.org/wikipedia/commons/thumb/e/e4/Artificial_neural_network.svg/400px-Artificial_neural_network.svg.png

[12] Warren S. Sarle, Cary, NC, USA. *ai-faq/neural-nets/part2*
Available via WWW: ftp://ftp.sas.com/pub/neural/FAQ2.html#A_backprop

[13] Genevieve Orr. *Multilayer Networks and Backpropagation*
Available via WWW: http://www.willamette.edu/~gorr/classes/cs449/Backprop/backprop.html

[14] Scott E. Fahlman and Christian Lebiere. *The Cascade-Correlation Learning Architecture*, August 29, 1991.
Available via WWW: http://www.cs.iastate.edu/~honavar/fahlman.pdf

[15] Warren S. Sarle, Cary, NC, USA. *ai-faq/neural-nets/part2*
Available via WWW: ftp://ftp.sas.com/pub/neural/FAQ2.html#A_act

[16] Warren S. Sarle, SAS Institute Inc., Cary, NC, USA.
Ill-Conditioning in Neural Networks, Available via WWW: ftp://ftp.sas.com/pub/neural/illcond/illcond.html