A comparison of a multi-layered perceptron versus a support vector machine on letter recognition

Marc Kendal and Simon Suthers, 30th March 2017

Brief description and motivation of the problem

Character and hand-writing recognition is a common problem in today's digital age. Use cases can include signature recognition, number plate recognition and post code recognition.

Character recognition is a common use case for both neural networks and support vector machines. In this paper we take a dataset of 20,000 roman letters, first created by Frey and State (1991) and train both a multi-layered perceptron neural network (MLP) and support vector machine (SVM) on the data to assess the relative merits and performances of the networks compared to each other.

Neural networks have been used for character recognition since before 1990. It is a well-researched area. Using Support Vector machines as an alternative algorithm for character recognition has been gaining popularity since the early 2000s, popularized by benchmarks set by AT&T and Bell Labs in the late 1990s. Both techniques are now common tools for character and hand writing recognition. Decoste and Schölkopf (2002) conclude that neural networks appear to be much faster than SVMs at test time. However, they also state that SVMs can produce better results on MNIST dataset than MLPs. This paper aims to evaluate and compare the two techniques on a standard dataset using metrics such as time taken to build the model and accuracy returned of the model.

Description of the dataset

In the original paper by Frey and Slate, a set of 20,000 unique letter images was generated by randomly distorting pixel images of the 26 uppercase letters from 20 different commercial fonts. The fonts represent five different stroke styles and six different letter styles. Letters were created randomly and distorted by changing a variety of parameter values including letter of the alphabet, linear magnification, aspect ratio and horizontal and vertical "warp". However, the letters created were kept to an extent where they were still humanly readable. The paper provides a thorough explanation of how the letter set was created. An example of the letter set can be seen in figure 1.

The letters were put into a 45 by 45 pixel grid and these grids were then scanned to produce 16 numerical attributes. Each attribute was then scaled linearly to a range of 0 to 15. The attributes included the number of "on" pixels in the box, the mean horizontal and vertical positions of all "on" pixels (which can include negative numbers for letters with a skew to one side), and the mean number of edges when scanning from left to right (this could help distinguish between W and M or I and L). A full explanation of these attributes can be found in the paper by Frey and Slate (1991).

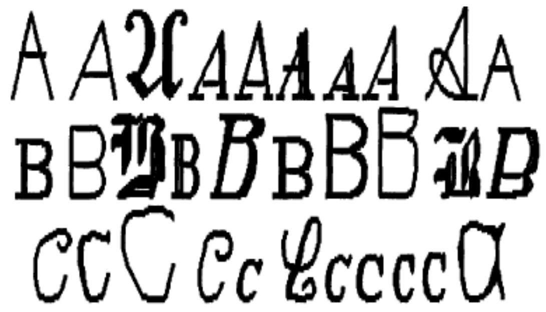


Figure 1 - Example of character images used

Frey and Slate used an adaptive rule-based classifier system first proposed by J.H. Holland in 1975 for letter classification. The paper used the dataset of 20,000 letters, representing all 26 letters of the alphabet. The dataset was split into 16,000 letters for training and 4,000 letters for testing. Using an exemplar-based rule creation procedure the paper obtained an 80.8% accuracy for predicting letters when classifying into an integer result set. The paper tried a number of different classification types, but saw accuracy reduced to below 60% when they used a binary and gray code result set. The paper produced a best accuracy of 82.7% using a hybrid accuracy-utility bidding system/exemplar-based rule creation procedure.

Due to time constraints, we restricted our paper to only classify the first 10 letters of the alphabet. When evaluating our results, we bore in mind that Faaborg found that a neural network had more accuracy when using a smaller output set.

Brief summary of the two neural network models

Multi-layer Perceptron with Levenberg–Marquardt

A multi-layer perceptron with backpropagation (MLP) is a traditional approach for letter classification. A MLP maps a number of inputs with a number of outputs. Between the inputs and outputs are usually one or more layers of hidden neurons. Each neuron applies a weight to its inputs along with a bias. The sum of these weights and biases are then applied to an activation function which returns the output of the neuron. Neurons are connected together through a forward only network. Once an output is obtained, an error rate is calculated using the difference between the actual output and the expected output. The weights and biases are then changed with a pre-set learning rate using a backpropagation technique (BPN) that uses stochastic gradient descent. An example of a MLP can be seen in figure 2.

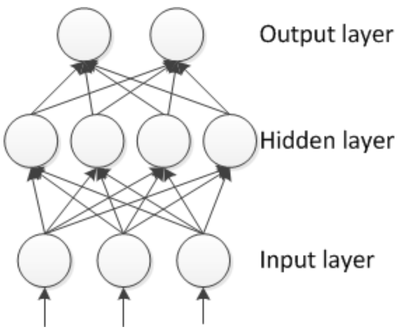


Figure 2 - Example of feedforward multi-layer perceptron with one hidden layer.

If a problem is linearly separable, the backpropagation technique guarantees that a result will be found. However, iteratively searching for a minimum gradient can be a very slow process. A heuristic approach, such as the Levenberg-Marquardt technique, using momentum and a variable learning rate, can speed up convergence rates for a traditional BPN.

Within Matlab, the *trainlm* network training function updates weight and bias values according to Levenberg-Marquardt optimization. Matlab publicises *trainlm* as "*often the fastest backpropagation algorithm in the toolbox, and is highly recommended as a first-choice supervised algorithm, although it does require more memory than other algorithms*". As speed was a primary concern for us, we chose to use Levenberg-Marquardt optimization.

The Levenberg-Marquardt (LM) algorithm is a Hebbian based algorithm that converges quicker than standard a backpropagation algorithm. The following is a brief description of the LM technique:

1. Present all inputs to the network and calculate all corresponding outputs
2. Compute the mean squared error over all inputs
   - The mean squared error is calculated by $e(x) = \frac{1}{2}\sum_{k=1}^{p}\sum_{l=1}^{m}(t_{kl} - y_{kl})^2$, where y_{kl} is the actual output of neuron *l* for input *k*, t_{kl} is the desired output at neuron *l* for input *k*, *P* is the total number of training patterns, *M* is the total number of neurons in the output layer, and *x* is the weights and biases for the network.
3. Compute the Jacobian matrix *J(x)* where *x* is the weights and biases for the network.
4. Weights are then updated using the formula: $w_{(t+1)} = w_t - (J^T(x)J(x) + \mu I)^{-1}J^T(x)E$, where μ is the training parameter, *I* is the identity unit matrix, $J^T(x)J(x)$ is the Hessian matrix, and *E* is a vector of size *kl* calculated as: $E = [(t_{l1} - y_{l1})(t_{l2} - y_{l2}) \dots (t_{lk} - y_{lk})]^T$
5. Re-compute the error. If the error is smaller than that computed in step 2 then reduce the training parameter μ.
6. Re-compute the error and go back to step 2
   - If the error is not reduced then increase the training parameter and go back to step 3.

The incremental rate and decremental rate of the training parameter default to 0.1 and 10 respectively. However, they can be adjusted within Matlab. For a very large μ, the LM algorithm becomes the error backpropagation algorithm. The algorithm is considered converged when one of the following criteria is met (the values we used are given in brackets):

- The maximum number of epochs has been reached (1000)
- The mean squared error is minimized to a predetermined value (0)
- The performance gradient falls below some predetermined value ($1e^{-7}$)
- The learning rate falls below a given value ($1e^{10}$)
- Validation performance has increased more than a maximum number of times since the last epoch it decreased (6)

Neural networks can be relatively slow to train, but are also relatively fast to produce results on unseen data. We chose the Levenberg-Marquart algorithm because of its speed, and our experiments were stopped using the validation performance criteria due to time constraints.

Support Vector Machine

Support vector Machines (SVM) are based on a learning theory developed by the Russian scientist Vladimir Naumovich Vapnik in 1962. The main difference between a neural network and a SVM is that a neural network seeks to minimize the error between the desired output and the output generated by the network, whereas a SVM seeks to maximize the margins between the borders of both classes.

A SVM separates classes by constructing a hyperplane (or a set of hyperplanes) in a high dimensional space. Classes are separated by any hyperplane that provides no misclassification on all data points. In SVMs the objective is to find the best separation hyperplane that provides the furthest distance between the nearest points of the two classes. This is referred to as the functional margin.

RUN/ CLASS	1	2	3	4	5	6
A	1	1	1	1	1	1
B	-1	0	0	0	0	0
C	0	-1	0	0	0	0
D	0	0	-1	0	0	0
E	0	0	0	-1	0	0
F	0	0	0	0	-1	0
G	0	0	0	0	0	-1

Table 1 – Example of one vs one classification system

A SVM is a binary classifier. It can only separate 2 categories. However, SVMs can be adapted to handle multiple classification cases by converting a multi-classification problem into multiple binary classifications. As we are trying to classify 10 letters, we try two common techniques for multi-classification: One-versus-all, where each label is classified against the rest of the labels combined, and one-versus-one, which classifies every pair of labels. An example of one-versus-one can be seen in table 1.

When testing a SVM after the model has been built, classification of new records using the one-versus-all case is done by assigning the class of the classifier with the highest output function. When using one-vs-one, classification is done by a “max-wins” voting strategy, in which every classifier assigns the instance to one of the two classes. The class that is assigned most often (has the most votes) is determined as the classification.

A SVM generally can train more quickly than a neural network. However, a SVM can classify new data more slowly than a neural network.

Hypothesis statement

Our hypothesis is that, on a testing set, a MLP will be quicker to return results and more accurate. However, a SVM will be quicker to train with a training set. Choosing between these compromises can be important when faced with a problem that is constrained either by time, or the need for accuracy.

Description of training and evaluation methodology, choice of parameters and experimental results

Multi-layer perceptron with backpropagation

When training the MLP, we first removed all data items where the letter was greater than J. The process that created the data should produce an even distribution of letters in the dataset (unlike actual written English). A count of the letters in the dataset was performed to make sure the letters were evenly distributed. The data was then randomly shuffled and 15% of the data was separated out to be used as a testing set in evaluating the performance of the model, both on its own and in comparison to the SVM. Both MLPs and SVMs are more efficient when working with normalised data, so the data was normalised so that each feature had a mean of 0 and variance of 1.

MLPs assume that the data is linearly separable in N dimensions. Given the number of dimensions (16), it would be impossible to assess if the data was linearly separable. However, using PCA, we visualized the data in 2 principal components to assess whether it was likely that the data would be linearly separable. Figure 3 shows the results, where we split each letter into a scatter plot of itself and every other letter. Whilst a number of letters have overlapping space, letters A and F appear to occupy their own space in 2 dimensions, so we assumed the data was linearly separable.

The LM algorithm in Matlab returns a softmax activation. To train the network, the dataset was split into inputs (the 16 features) and outputs. The outputs were then converted from letters to a 10 row matrix where the row corresponding to the letter has a 1 and all other rows have a 0. This can sometimes be referred to as an integer coding (Frey & Slate and Trebar).

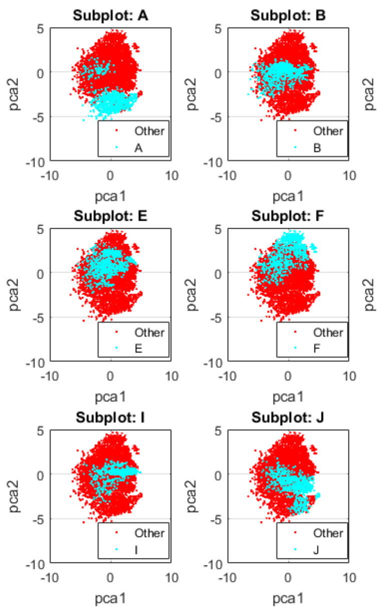


Figure 3 - Distribution of letters reducing 16 dimensions to 2 principal components. Letters A and F look like they occupy their own space. This led us to assume the data was linearly separable.

The performance of the network will depend on the structure of the network and training algorithm. We performed two experiments using grid search to assess the best structure. Both grid searches used 5-fold cross validation to confirm the results. We chose to restrict cross validation to 5-folds, rather than 10, for speed of processing. To assess the network, we used the mean value (across the 5 folds) of accuracy, mean squared error (MSE) and time taken to train the network. Due to the softmax nature of the LM algorithm, we placed more emphasis on the MSE, rather than accuracy to assess the networks.

The first grid search sought to optimize various hyper-parameters. Due to time constraints, we restricted the network to always have 2 hidden layers but tried various numbers of neurons in each hidden layer. As the LM algorithm changes the learning rate according to the MSE, we also varied the learning rate increase rate and decrease rate. We were interested in seeing what effect these changes had on the accuracy and time taken to train a network. The parameters used in the grid search can be seen in table 2.

Number of layers	2
Number of neurons in layer 1	10, 20, 30
Number of neurons in layer 2	10, 20
Learning rate increase factor	5, 10
Learning rate decrease factor	0.01, 0.1

Table 2 – options used to optimize hyper-parameters in the MLP

We found that the MLPs with 30 neurons in the first hidden layer and 20 neurons in the second hidden layer produced the best accuracy and lowest MSE. However, these MLPs also took the longest to train. Setting the learning increase rate to 5 produced both a lower MSE and a shorter training time, compared to 10. However, setting the learning rate decrease to 0.1 produced a lower MSE but a longer training time. The top 5 results of grid search ordered by MSE can be seen in table 3.

Layer 1 Neurons	Layer 2 Neurons	mu increase	mu decrease	accuracy (%)	mse	time elapsed (sec)
30	20	5	0.1	93%	0.007666	293
30	20	5	0.01	94%	0.009093	281
30	20	10	0.1	94%	0.009133	324
30	20	10	0.01	93%	0.009308	306
30	10	5	0.1	92%	0.009549	135

Table 3 - Results of hyper-parameter grid search. 30 neurons in hidden layer 1 and 20 neurons in hidden layer 2 performed the best.

The second grid search tried only increasing the number of neurons in the first hidden layer. Theory suggests that the more neurons, the better the accuracy of the network. There may be a point where increasing the number of neurons further over fits the model on the training data. However, time constraints did not allow us to explore that limit. Figure 4 shows the results. Performance was measured in mean square error. As the number of neurons increase, the mean square error decreases linearly. However, the time taken to train the model increases in what appears to be a quadratic manner. The network using 50 hidden neurons took 10 minutes to train, which may not be acceptable in a working environment. Despite a higher number of hidden neurons returning better accuracies, it was decided to keep the model at a maximum of 30 neurons because of the length of time taken to train the networks.

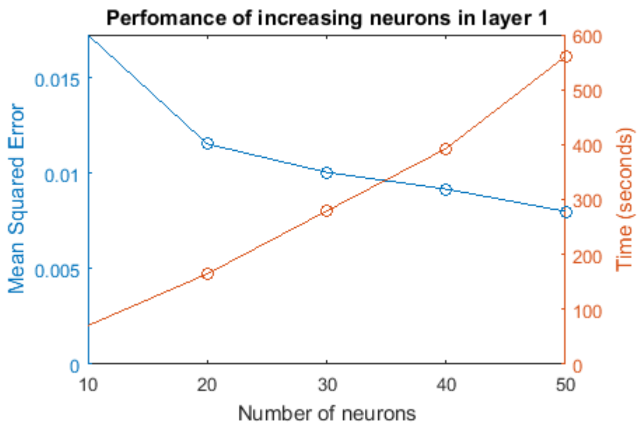
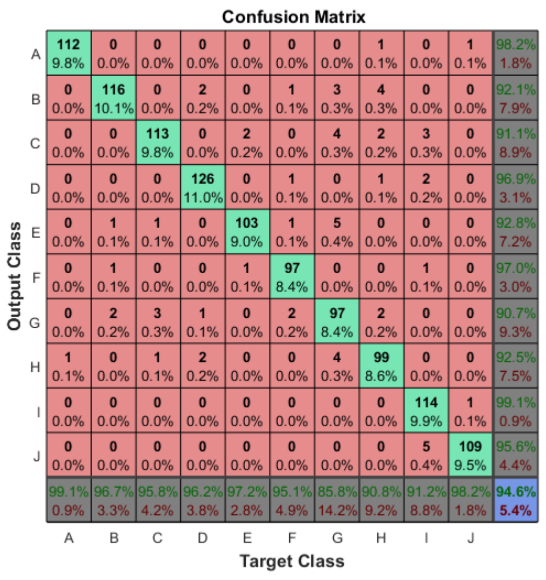


Figure 4 - Performance of the MLP network as the number of neurons in hidden layer 1 is increased. There is a trade-off between accuracy of the model and the time taken to train the model.

A network was then built using the hyper-parameters that returned the best MSE from the grid search. A validation set was used for the stopping criteria to save on time taken to build the model. The final model was then tested with the testing data that was split at the beginning of the experiment. The network returned an accuracy of 94.6% on the testing data in 0.3 seconds. This was a reasonably high figure and shows the viability of using a MLP for character recognition. A full confusion matrix of the results can be seen in figure 5. It is notable that, by letter, G returns the lowest sensitivity and precision, and that A returns the highest sensitivity and 2nd highest precision, which could be expected given the fact that A occupied its own space when reducing to 2 dimensions.



	A	B	C	D	E	F	G	H	I	J	
A	112 9.8%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.1%	0 0.0%	1 0.1%	98.2% 1.8%
B	0 0.0%	116 10.1%	0 0.0%	2 0.2%	0 0.0%	1 0.1%	3 0.3%	4 0.3%	0 0.0%	0 0.0%	92.1% 7.9%
C	0 0.0%	0 0.0%	113 9.8%	0 0.0%	2 0.2%	0 0.0%	4 0.3%	2 0.2%	3 0.3%	0 0.0%	91.1% 8.9%
D	0 0.0%	0 0.0%	0 0.0%	126 11.0%	0 0.0%	1 0.1%	0 0.0%	1 0.1%	2 0.2%	0 0.0%	96.9% 3.1%
E	0 0.0%	1 0.1%	1 0.1%	0 0.0%	103 9.0%	1 0.1%	5 0.4%	0 0.0%	0 0.0%	0 0.0%	92.8% 7.2%
F	0 0.0%	1 0.1%	0 0.0%	0 0.0%	1 0.1%	97 8.4%	0 0.0%	0 0.0%	1 0.1%	0 0.0%	97.0% 3.0%
G	0 0.0%	2 0.2%	3 0.3%	1 0.1%	0 0.0%	2 0.2%	97 8.4%	2 0.2%	0 0.0%	0 0.0%	90.7% 9.3%
H	1 0.1%	0 0.0%	1 0.1%	2 0.2%	0 0.0%	0 0.0%	4 0.3%	99 8.6%	0 0.0%	0 0.0%	92.5% 7.5%
I	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	114 9.9%	1 0.1%	99.1% 0.9%
J	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	5 0.4%	109 9.5%	95.6% 4.4%
	99.1% 0.9%	96.7% 3.3%	95.8% 4.2%	96.2% 3.8%	97.2% 2.8%	95.1% 4.9%	85.8% 14.2%	90.8% 9.2%	91.2% 8.8%	98.2% 1.8%	94.6% 5.4%

Figure 5 - Confusion matrix of MLP with testing set

Support Vector machine

As with the MLP, we only considered the first ten letters of the alphabet when training the SVM model. The same 15% of records were removed so that they could be used at a later stage to test the model both on its own and against the MLP. The training set was also randomly shuffled. As we were classifying more than 2 classes, we tried both one-vs-one and one-vs-all when optimizing hyper-parameters for the SVM. When using the multiple binary classifiers in a SVM**,** Thomé states "all binary classifiers are assumed to show equal competence distinguishing their respective class". This means that there is an assumption that all the binary classifiers are trustworthy in equal measure. We had already tested for an equal distribution of letters in the dataset so assumed this would not affect the model. To try to find the best model, we ran a grid search to optimize the hyper-parameters: coding, box constraint, kernel scale, and kernel function. The values we tried can be seen in table 4. We used 10-fold cross validation on the training set, which is the default in Matlab.

Hyper-parameter optimization for SVMs in Matlab attempts to find the parameters with minimum cross-validation loss. For multiple classes, loss is defined as the weighted fraction of misclassified observations. The best model found used a Gaussian function with a box constraint of 1 and a kernel scale of 1 using one-vs-all. This model returned a 92.1% accuracy on the training data.

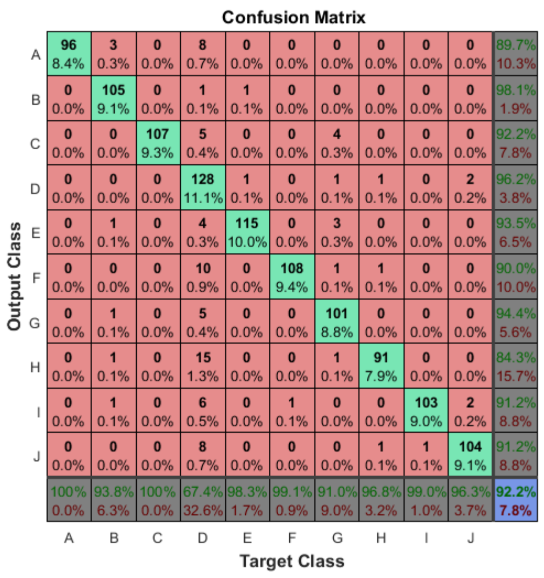
coding	One-vs-one, one-vs-all
Box constraint	0.01, 0.1, 1
Kernel scale	0.01, 0.1, 1
Kernel function	Gaussian, linear

Table 4 – Hyper-parameter values tested for the SVM

We then tested this SVM on the testing data that was extracted in the first step. Using the testing data, the SVM returned an accuracy of 92.2% in 5 seconds. The confusion matrix of SVM results on the testing set can be seen in figure 6. It is interesting to note that the SVM is just as accurate with the testing set as the training set. The SVM appears to be much better at predicting the letter G than the MLP. In the SVM G returned the third highest precision whereas G returned the worst precision in the MLP. D performs quite badly in sensitivity in the SVM. D is mistaken for all the letters in the SVM, with it most often being mistaken for an H. Both A and C returned no false negatives at all.

We plotted the results of the testing set of both the SVM and MLP back onto the original trellis grid plotting the distribution of letters reduced to two dimensions. The scatter plots for each letter plotted dots in four colours, two for correct true and false classifications and two for incorrect true and false classifications. We wanted to see if the incorrectly classified data points occupied any particular feature space. The charts did not show anything of note so have not been included in the paper.

Analysis and critical evaluation of results



Confusion Matrix

Output Class \ Target Class	A	B	C	D	E	F	G	H	I	J	
A	96 8.4%	3 0.3%	0 0.0%	8 0.7%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	89.7% 10.3%
B	0 0.0%	105 9.1%	0 0.0%	1 0.1%	1 0.1%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	98.1% 1.9%
C	0 0.0%	0 0.0%	107 9.3%	5 0.4%	0 0.0%	0 0.0%	4 0.3%	0 0.0%	0 0.0%	0 0.0%	92.2% 7.8%
D	0 0.0%	0 0.0%	0 0.0%	128 11.1%	1 0.1%	0 0.0%	1 0.1%	1 0.1%	0 0.0%	2 0.2%	96.2% 3.8%
E	0 0.0%	1 0.1%	0 0.0%	4 0.3%	115 10.0%	0 0.0%	3 0.3%	0 0.0%	0 0.0%	0 0.0%	93.5% 6.5%
F	0 0.0%	0 0.0%	0 0.0%	10 0.9%	0 0.0%	108 9.4%	1 0.1%	1 0.1%	0 0.0%	0 0.0%	90.0% 10.0%
G	0 0.0%	1 0.1%	0 0.0%	5 0.4%	0 0.0%	0 0.0%	101 8.8%	0 0.0%	0 0.0%	0 0.0%	94.4% 5.6%
H	0 0.0%	1 0.1%	0 0.0%	15 1.3%	0 0.0%	0 0.0%	1 0.1%	91 7.9%	0 0.0%	0 0.0%	84.3% 15.7%
I	0 0.0%	1 0.1%	0 0.0%	6 0.5%	0 0.0%	1 0.1%	0 0.0%	0 0.0%	103 9.0%	2 0.2%	91.2% 8.8%
J	0 0.0%	0 0.0%	0 0.0%	8 0.7%	0 0.0%	0 0.0%	0 0.0%	1 0.1%	1 0.1%	104 9.1%	91.2% 8.8%
	100% 0.0%	93.8% 6.3%	100% 0.0%	67.4% 32.6%	98.3% 1.7%	99.1% 0.9%	91.0% 9.0%	96.8% 3.2%	99.0% 1.0%	96.3% 3.7%	92.2% 7.8%

Figure 6 - Confusion matrix of SVM with testing set

Using the accuracy on the testing set as a measure of the two networks, the MLP using LM, with an accuracy of 94.6%, returned a better result than the SVM, with an accuracy of 92.4%, although the difference is arguably quite small. The 94.6% accuracy we obtained using the MLP seems quite good at first hand. However, Faaborg, admittedly using a different (and much smaller) dataset, obtained a 100% accuracy when predicting 8 letters using over 1,700 epochs. If we relaxed the stopping criteria for training the MLP maybe we could have obtained a better accuracy. However, time constraints prevented us from trying different stopping criteria. Our model was only trained on 10 letters. Faaborg saw a large drop in accuracy as more outputs were added and we would probably have seen something similar, though probably not to the same extent as our training set was much bigger than that used by Faaborg.

Using grid search, the MLP took over an hour to train, whereas the SVM could be trained in just over 15 minutes. The MLP took 0.3 seconds to return results on our testing set, whereas the SVM took 5 seconds. The difference in time on unseen data may not seem like much, but could be critical when dealing with much larger datasets or in situations where response time is important.

There is a very large range of hyper-parameters that can be adjusted for the MLP and we only barely scratched the surface with the parameters we tried. Increasing the number of hidden neurons in each layer is likely to produce better results as we didn't see anything to suggest we had reached a threshold where the number of neurons was overfitting the model, although as we were using an early stopping criteria, Caruana et al explain that we would be unlikely to see this.

With the SVM, however, our hyper-parameter optimization was fairly conclusive. Different values of the Box Constraint and kernel scale could have been searched but given our grid search returned the Matlab default for both values, it would suggest that no other values would return a higher accuracy.

Decoste and Schölkopf point out that SVMs training hand writing recognition can return better accuracies when the training set includes machine generated examples. Given that our dataset consists of entirely machine generated examples, it should be borne in mind that our results would differ if we used human generated characters.

Conclusions and future work

We conclude that, for letter classification, an MLP seems marginally more accurate than an SVM. However, when choosing a model, consideration needs to be taken that an MLP takes considerably longer to train than a SVM, but can return a classification of unseen data more quickly.

For future work, it would be interesting to see how much better a MLP performs when using a larger number of hidden neurons and how a SVM performs with a polynomial kernel function. Also, for the MLP, we used an integer coding system for our true result set. However, a number of papers (Frey & State and Trebar) used a binary and also gray coding system for a result set. It would be interesting to see how the MLP performs using a binary coding instead of an integer coding. We would hypothesize that it might not be as accurate, but it could be noticeably quicker to train.

References

Letter Recognition Using Holland-Style Adaptive Classifiers. P.W. Frey, D.J. *Slate Machine Learning, 6, 161-182 (1991)*

Use of MATLAB Neural Networks Toolbox in a Character Recognition Problem. Mira Trebar. *Comput Appl Eng Educ 13: 6671, 2005*

SVM Classifiers – Concepts and Applications to Character Recognition. Antonio Carlos Gay Thomé

Using Neural Networks to Create an Adaptive Character Recognition System. Alexander J. Faaborg. *Cornell University, Ithaca NY, May 2002*

Training invariant support vector machines. D. DeCoste and B. Schölkopf. *Machine Learning, 46(1/3):161, 2002*

Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and early stopping. R.Caruana, S.Lawrence, L.Giles