ML 574 Project 3

Daniel Amirtharaj

`damirtha@buffalo.edu`
UB Person Number 50291137

1 Projrect overview

This project aims to develop a predictive model for hand-writing digit recognition using machine learning. Features extracted from hand-written numeric digit samples in the MNIST dataset have been used to predict the hand-written numeric digit in an image sample. Four machine learning classifiers have been used to compare their respective advantages/disadvantages, and also to combine their predictions to create an ensemble classifier.

Objective

The objective here is to train a classifier using the MNIST dataset to predict the numeric digit from a given hand-written digit image sample, test them on both the MNIST and USPS datasets, compare results between the datasets as well as between different classifiers as well as to create an ensemble classifier that uses multiple models for prediction. The following are the tasks involved.

1. Pre-process image samples and apply anti-aliasing and normalization to aid enhanced recognition. Extract features as pixel values from these images, and simultaneously assign target labels for each image sample.
3. Split the features extracted from the MNIST dataset into 3 sets, the training, validation and test sets. The USPS dataset does not have to be split, since it is used only for testing.
4. Use Logistic regression (with one v all classification), Neural networks, SVM and Random forests to find weights to fit each model.
7. Change parameters such as learning rate, number of basis functions and regularization coefficient and other neural network, SVM and Random forest parameters and observe changes in performance using the validation set, and finally pick model setting with best performance.
6. Evaluate accuracy of the models using the predicted labels and the given labels for each test data sample in both MNIST and USPS datasets.
8. Using the predictions from the 4 classifiers, make a final prediction for an ensemble classifier using majority voting and evaluate its performance.

2 Analysis

Some of the key concepts used in the project have been described briefly here.

2.1 Handwriting analysis and ML

Handwriting analysis is a field vital to forensics and document examinations. Through advancements in handwriting recognition and document analysis with the development of ML, many problems have been solved. Discrepancies and challenges in traditional analysis has been greatly mitigated with the use of predictive models to ensure better accuracy as well as avoiding human error and judgment, in the recent years.

2.2 Logistic Regression

Logistic regression models predictions which are discrete and not continuous. The input data is still assumed to be continuous. It is implemented as an extension to linear regression by applying a sigmoid function to the prediction (linear function of parameters and input features). This compresses the range of the output to values between 0 and 1, which can interpreted as the probability that the model has predicted a certain class in a binary classification problem. This is ensured by using a loss function that maximizes the log-likelihood of label given the model's parameters.

One vs. All classification Logistic regression in its simplest form, is a binary classifier, which can classify data samples into 2 discrete classes. This can be further extended to classify data between multiple classes by training K binary models considering the data sample's label to be 1 for class k and all other classes labels to be 0, if the given label is k. This is the principle behind the working of this classifier. Softmax activation can be used in place of sigmoid activation to ensure that the total probabilities of each class prediction add up to one.

2.3 Support Vector Machines

The SVM is a classification algorithm that is a subset of a larger set of classifiers in ML, the large margin classifiers. Large margin classifiers differ from classifiers such as logistic regression in the way they separate different classes. Their optimization objective does not only minimize the error or loss function but also maximizes the width of separation between classes in the feature space, to ensure different classes are kept seperated as far away from each other as possible (the width can be tuned using hyper-parameters such as C, gamma and choice of kernel). They are thus powerful when classifying data from noisy samples which may introduce outliers. They are usually implemented with a kernel method, to introduce non-linearity in the input feature to label mapping.

Kernel methods Kernel Methods, add non-linearity to the dataset by transforming the input features into another set of transformed features that are linearly related to the target vector. This is extremely useful while modeling non-linear functions using SVMs. The RBF is a good example, it is computed, by taking the gaussian radial basis function of each data sample centered around a point (centroid of cluster) in the f dimensional feature space of the input dataset. They are the most popular choice of kernel, in an SVM classifier, but can be extended to other algorithms as well.

2.4 Random Forests

Random forests use multiple decision trees trained on different randomly chosen data samples and input features (with repetition, also known as bagging or bootstrap aggregating) to predict a label for a test sample. It aggregates results from different tree models to make a final prediction. Since results from differently modelled trees are combined, the prediction it makes has lower variance and error due to bias and is much more reliable when modelled on real world data.

Decision Trees Decision trees model data by branching data samples based on a feature chosen iteratively until branching is not required and each leaf has only one class of data. This ensures that each data sample is classified into any class, resulting is a 100% training accuracy. This might not be so good for unseen data and hence must be 'pruned', i.e nodes in the tree can be removed and then have it's performance evaluated on a validation set to ensure overfitting does not occur. The ID3 is a popular algorithm used to model decision trees, and works by picking a feature with the highest information gain or that which will result in the least entropy of it's resulting classification, and then branching until convergence. This gives a smaller tree size and is thus computationally efficient. Decision trees are also interprettable and their working can be visualized by a human observer, and can thus help analyze predictions and tune the model better based on analysis.

2.5 Neural Networks

The problem given is a supervised classification problem. Supervised, since we already know the output labels, and classification because the output labels are discrete. To solve this problem any classification algorithm can be used. A Neural network has been used here to classify the data. Neural networks is a highly efficient and robust algorithm used widely today. It emulates the human brain in terms of how we learn, and process data. Neural networks can represent complex non-linear functions and have applications in complex learning problems such as computer vision. They are flexible and can also be configured to give better results depending on the problem requirements.

Configuration of Neural Networks The following are parameters of the Neural network classifier that can be tuned to enhance performance,

Neural network model parameters and properties

1. Number of hidden layers
2. Number of neurons in each layer
3. Activation function of each layer
4. Optimization algorithm
5. Error function

Training parameters,

1. Batch size
2. Learning rate
3. Number of epochs

Pre-processing parameters Input data can be processed or transformed to another form in order to improve the effectiveness of the algorithm

2.6 Regularization

It is a tool used to avoid overfitting in machine learning algorithms. Since the training data can be tuned and adjusted to give the maximum performance on a dataset, its performance on the dataset is not reliable, and does not represent how the algorithm will behave with unseen data. In order to prevent the model from giving great training performances but poor ability to generalize, regularization is used. It ensures that the weights learned do not assume disproportionate or abnormally large values, preventing overfitting.

2.7 Training/Validation/Test split

The dataset is split into 3 sets, the training, validation and the test sets, usually in the ratio 8:1:1. The training dataset is used solely to train the model, while the validation dataset is used to tune hyper-parameters on the model, and finally the test set is used to measure the model's performance, and this gives a good idea on how well the model has learned, and its ability to generalize over unseen data samples.

3 Method

3.1 Feature extraction

The MNIST dataset has image samples that had to be sampled in order to construct the input feature vectors. This was readily available before the start of the project and was not repeated here. The sampled images were anti-aliased and normalized to form the 28x28 features that would later be fed to the classifiers for training. The USPS dataset was also sampled and resized to match the number of features extracted from the MNIST dataset.

3.2 Data Preprocessing

The training target labels were converted to the one-hot binary bit representation (For logistic regression and Neural networks) in order to train the algorithm for each class against all others. Since the MNIST dataset generated in the feature extraction step was used to train the models, the dataset was split into 3 sets, the training, validation and test sets in the ratio of 8:1:1. The entire USPS dataset was used solely for testing.

3.3 Logistic Regression with mini-batch Gradient descent algorithm

Training The equations that establish gradient descent for logistic regression were laid out and vectorized in python with the help of the 'numpy' library. The input feature vectors, data labels and the initial weight vectors (initialized randomly) were fed to this program. Softmax activation was used as the prediction output of the model, and the class with highest probability was chosen as the predicted label. Mini-batch Gradient descent was implemented by shuffling the training dataset for each epoch and picking batches of fixed size from the whole dataset. Once the model was trained, accuracy was evaluated by computing the number of correct predictions over the total number of test samples.

Validation and Testing Once the model was trained using a set of hyper-parameters lambda, learning rate, epochs and batch size, it's performance was evaluated using the validation set, and tuned so that the best accuracy was obtained on the validation set for a new set of hyper-parameters. The validated model was then tested using the MNIST and USPS testing datasets to evaluate it's overall performance and to see how well it performed on different datasets.

3.4 Support Vector Machines

Training SVM was implemeted using the SVC method provided in the scikit learn package. Since the algorithm took very long to converge on the whole training dataset, the dataset was reduced to 10,000 samples picked randomly from the training dataset. Once the model was trained, accuracy was evaluated by computing the number of correct predictions over the total number of test samples.

Validation and Testing Once the model was trained using a set of hyper-parameters C, kernel method, gamma value etc., it's performance was evaluated using the validation set, and tuned so that the best accuracy was obtained on the validation set for a new set of hyper-parameters. The validated model was then tested using the MNIST and USPS testing datasets to evaluate it's overall performance and to see how well it performed on different datasets.

3.5 Random Forests

Training Random forests was implemeted using the RandomForestClassifier method provided in the scikit learn package. The MNIST training dataset was used to train this model. Once the model was trained, accuracy was evaluated by computing the number of correct predictions over the total number of test samples.

Validation and Testing Once the model was trained for a number of trees, it's performance was evaluated using the validation set, and tuned so that the best accuracy was obtained on the validation set for a new forest size. The validated model was then tested using the MNIST and USPS testing datasets to evaluate it's overall performance and to see how well it performed on different datasets.

3.6 Neural networks

Training A Deep Neural networks and a Convolutional Neural Network were laid out using the sequential model from the keras package. Nodes and layers were added to the models, a softmax activation was used at the output layer and the cross entropy loss function with the Adamax optimizer were used to train the models. Additionally, Convolution (3,3) filters were added to the CNN model. The training dataset was fed to the model for a certain batch size and number of epochs.

Validation and Testing Once the model was trained using a set of hyper-parameters nodes, layers, learning rate, dropout etc., it's performance was evaluated using the validation set, and tuned so that the best accuracy was obtained on the validation set for a new set of hyper-parameters. The validated model was then tested using the MNIST and USPS testing datasets to evaluate it's overall performance and to see how well it performed on different datasets.

3.7 Ensemble classification

MNIST as well as the USPS Predictions of the test data inputs from the 5 classifiers (including CNN and DNN) were combined to form a single prediction. This aggregation was performed by taking the mode of the predictions of the 5 classifiers for each data sample. This ensemble prediction was then compared to the target labels to get the accuracy of the ensemble classifier.

4 Results

All 5 classifiers (including CNN and DNN) and the ensemble classifier were trained using the MNIST dataset, and tested using the MNIST and the USPS datasets. The models were evaluated based on their performances on different configurations. The following sections below highlight the findings.

4.1 Logistic regression

The following results were obtained by tuning the hyper-parameters on the Logistic regression model using grid search. The algorithm performed best on the validation set for a batch size of 20, η 0.01 and λ 0.01 and converged within an epoch. The following table gives the validation accuracies for different combinations of η and λ.

		η			
		0.1	0.03	0.01	0.005
λ	0.3	70.97	86.36	89.15	90.31
	0.1	79.52	87.17	91.08	90.8
	0.03	80.04	89.5	90.98	90.44
	0.01	88.92	90.72	91.18	90.33

Table 1. Measure of validation accuracy (%) for different learning rates η and regularization parameters λ values over 1 epoch for a mini-batch size of 20.

Testing accuracy On the MNIST dataset: 91.01%, On the USPS dataset: 33%.

The confusion matrix of this classifier is given in the following table.

	0	1	2	3	4	5	6	7	8	9
0	963	0	3	1	0	4	7	1	1	0
1	0	1112	3	2	0	3	4	1	10	0
2	8	10	930	9	8	1	14	9	35	8
3	5	0	30	910	0	19	3	7	19	17
4	2	4	9	0	878	0	12	2	6	69
5	13	3	8	42	9	753	14	4	33	13
6	16	3	8	1	9	11	905	2	3	0
7	5	18	27	4	7	2	0	894	0	71
8	12	14	14	20	10	29	17	5	825	28
9	11	7	1	10	25	6	1	11	6	931

Table 2. Confusion matrix for the logistic regression model trained and tested on the MNIST dataset. (Rows represent targets, columns represent predictions)

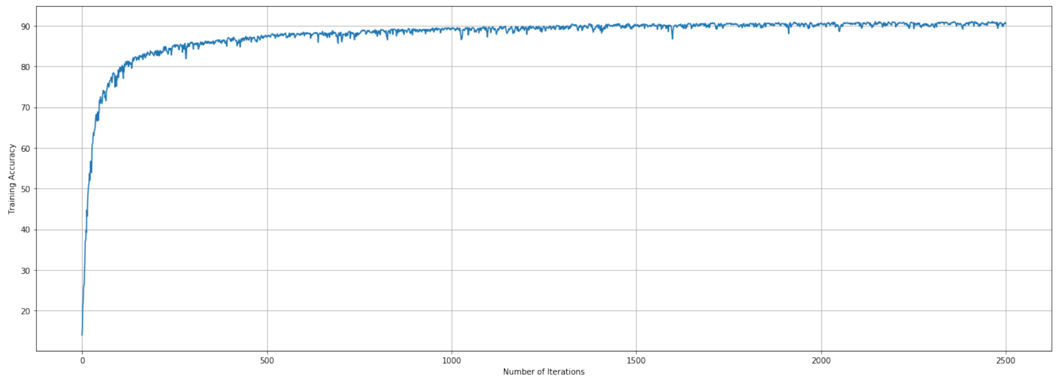

Figure 1: Training Accuracy of the logistic regression model for each iteration of the algorithm.

Observations The logistic regression algorithm performs reasonably well on the MNIST dataset giving an accuracy of over 90%. It also converged quickly when implemented using mini-batch gradient descent, this can be observed in figure 1. The confusion matrix gives an idea of where it goes wrong and is unable to make good predictions. It can be observed that the algorithm gets digits 4 and 7 wrong many times, classifying them as nine.

4.2 SVM

The following results were obtained by tuning the hyper-parameters (kernel method, C, gamma) on the SVM as suggested in the documentation. The algorithm performed best on the validation set for the RBF kernel with other parameters set to default values in the SVC function. Only 10,000 random samples from the MNIST dataset were used to train the algorithm, to reduce the time to train the SVM, since the algorithm converged quickly.

Validation accuracy for,

1. Kernel method: Linear, other parameters set to default. 91.79%.
2. Kernel method: RBF, other parameters set to default. 92.63%.
3. Kernel method: RBF, gamma = 1 and other parameters set to default. 18.08%.

Testing accuracy On the MNIST dataset: 92.4%, On the USPS dataset: 38.45%.

The confusion matrix of this classifier is given in the following table.

	0	1	2	3	4	5	6	7	8	9
0	962	0	2	0	0	7	5	1	3	0
1	0	1116	3	2	0	2	4	0	8	0
2	10	2	918	13	20	1	20	15	29	4
3	5	4	15	925	0	24	2	11	19	5
4	1	3	6	0	914	0	12	3	2	41
5	7	10	6	32	13	791	16	3	10	4
6	11	3	6	1	6	8	921	0	2	0
7	3	20	26	1	9	1	0	937	5	26
8	6	12	6	14	14	32	12	10	855	13
9	13	9	1	11	42	8	1	16	7	901

Table 3. Confusion matrix for the SVM model trained and tested on the MNIST dataset. (Rows represent targets, columns represent predictions)

Observations The SVM algorithm performs reasonably well on the MNIST dataset giving an accuracy of over 90%. It also converged quickly when trained over 10,000 random samples. The algorithm performs poorly for RBFs with gamma = 1, and since gamma represents the region of influence of a training sample over the entire dataset, choosing a value too small or too large will cause heavy changes to the RBFs and how they are able to model the data. This is very evident in this case, as SVM is unable to find a non-linear mapping between the inputs and labels.

The confusion matrix gives an idea of where the model goes wrong and is unable to make good predictions. It can be observed that the algorithm gets confused between 4 and 9 many times, classifying them as 9 and 4 respectively.

4.3 Random Forests

The following results were obtained by changing the number of decision trees used with bagging in the random forest algorithm. The model performed best on the validation set for 200 trees (kept increasing marginally with increasing number of trees, and hence was stopped at 200). The following table gives validation accuracies for different number of decision trees used.

Number of Trees	Validation Accuracy
1	75.47
5	87.48
10	91.99
20	93.95
50	96.65
100	97.03
200	97.24

Table 4. Measure of validation accuracy (%) for different number of trees in the random forest.

Testing accuracy On the MNIST dataset: 97.04%, On the USPS dataset: 37%.

The confusion matrix of this classifier is given in the following table.

	0	1	2	3	4	5	6	7	8	9
0	969	0	1	0	0	3	3	1	3	0
1	0	1121	3	3	0	2	2	0	3	1
2	6	0	998	7	2	0	4	9	6	0
3	0	0	13	968	0	9	0	8	9	3
4	1	0	1	0	952	0	6	0	2	20
5	3	0	0	15	3	856	7	2	4	2
6	6	3	0	1	3	5	937	0	3	0
7	1	3	19	1	1	0	0	991	2	10
8	3	0	6	9	3	5	3	3	931	11
9	5	5	1	10	9	4	1	4	6	964

Table 5. Confusion matrix for the Random forest model trained and tested on the MNIST dataset. (Rows represent targets, columns represent predictions)

Observations The Random forest algorithm performs reasonably well on the MNIST dataset giving an accuracy of over 95%. The confusion matrix gives an idea of where it goes wrong and is unable to make good predictions. It can be observed that the algorithm gets confused and classifies 6 as 2, 9 as 4 and 3 as 9 multiple times.

4.4 Neural networks

2 networks were trained, a DNN (deep neural network) and a CNN (convolutional neural network).

4.4.1 DNN

The following results were obtained by tuning the hyper-parameters on the Neural network model using grid search. The algorithm performed best on the validation set for 200 nodes in each layer, 2 layers, dropout of 0.2, batchsize 128 and epochs 100, with the Adamax optimizer, cross-entropy loss and relu activation. The following table gives the validation accuracies for different combinations of no. of nodes and layers.

		Number of nodes			
		20	50	100	200
Layers	1	95.81	97.35	97.88	98.15
	2	95.39	97.39	98.16	98.32
	3	95.27	97.34	97.87	98.26

Table 6. Measure of validation accuracies (%) for different values of nodes and layers with dropout 0.2, keeping all other hyper-parameters constant.

The network was first trained without dropout and performed a little worse and gave the following validation accuracies.

		Number of nodes			
		20	50	100	200
Layers	1	95.95	97.32	97.8	98.06
	2	96.23	97.39	97.81	98.18
	3	96.07	97.31	97.81	98.1

Table 7. Measure of validation accuracies (%) for different values of nodes and layers without dropout, keeping all other hyper-parameters constant.

Testing accuracy On the MNIST dataset: 98.4%, On the USPS dataset: 47.57%.

The confusion matrix of this classifier is given in the following table.

	0	1	2	3	4	5	6	7	8	9
0	972	1	0	1	0	1	2	1	2	0
1	0	1129	1	1	0	2	2	0	0	0
2	3	2	1014	3	1	0	3	4	2	0
3	0	0	3	994	0	5	0	6	2	0
4	0	0	4	1	965	0	4	1	0	7
5	3	0	0	10	1	873	3	0	2	0
6	2	2	1	1	2	3	946	0	1	0
7	1	1	5	1	0	0	0	1014	3	3
8	3	0	3	7	1	4	0	4	950	2
9	1	4	0	6	9	1	1	3	1	983

Table 8. Confusion matrix for the Neural network (DNN) model trained and tested on the MNIST dataset. (Rows represent targets, columns represent predictions)

4.4.2 Observations

The DNN algorithm performs very well on the MNIST dataset giving an accuracy of over 98% on the MNIST test dataset. It also performs relatively better on the USPS dataset when compared to the previous classifiers. It is able to learn with a training accuracy of nearly 100% making it one of the best performing classifiers seen so far (can be observed in figure 2). The confusion matrix gives an idea of where it goes wrong and is unable to make good predictions. It can be observed that the algorithm gets almost all digits perfectly with a few minor misses here and there.

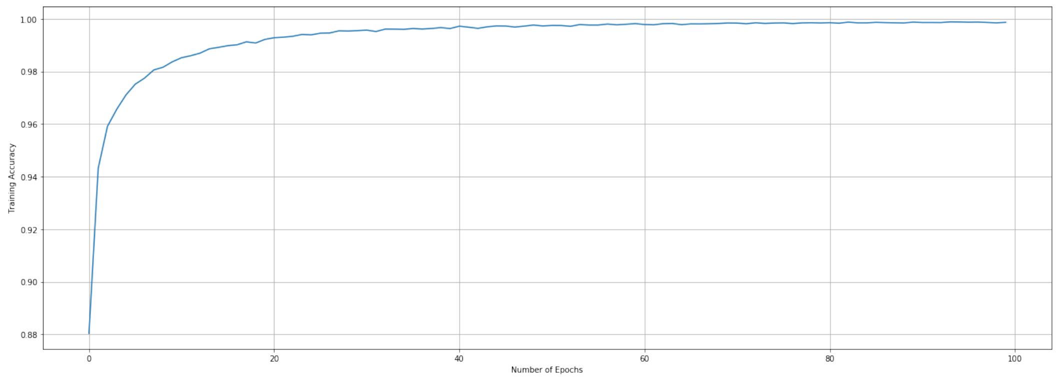

Figure 2: Training Accuracy of the DNN model over the number of epochs.

4.5 CNN

The DNN network in the previous section was augmented with 2 convolutional layers to perform better learning over the 2 dimensional image pixel features. The best validation accuracy was obtained when it was trained with 2 layers of 32 filters of size 3X3 with relu activation, with max pooling of pool size 2X2. Other settings such as one filter of size 20 and 2 filters of sizes 20,20 were also attempted. These gave similar accuracies that were marginally lower than the one mentioned earlier.

Convolutional Layer	Validation Accuracy
10	98.71
20	98.84
10,10	99
20,20	98.17

Table 9. Measure of validation accuracies (%) for different combinations of convolutional filters, keeping all other hyper-parameters constant.

Testing accuracy On the MNIST dataset: 99.08%, On the USPS dataset: 59.29%.

The confusion matrix of this classifier is given in the following table.

4.5.1 Observations

The CNN algorithm performs well on the MNIST dataset giving an accuracy of around 99% on the MNIST test dataset, and is also better than DNN when predicting USPS features. This can be attributed to its ability to pick and learn from features of 2D shaped inputs. It is also able to learn with a training accuracy of nearly 100%, similar to the DNN (can be observed in figure 3). The confusion matrix gives an idea of where it goes wrong and is unable to make good predictions. It can be observed that the algorithm gets almost all digits perfectly with a few minor misses here and there, also similar to DNN.

	0	1	2	3	4	5	6	7	8	9
0	976	0	1	0	0	0	0	2	1	0
1	0	1130	0	2	0	0	1	1	1	0
2	0	1	1024	2	1	0	0	3	1	0
3	0	0	0	1000	0	6	0	1	3	0
4	0	0	2	0	971	0	1	1	1	6
5	0	0	1	3	0	886	1	1	0	0
6	4	1	0	1	2	5	940	0	5	0
7	0	1	1	0	0	0	0	1019	1	0
8	1	0	2	0	0	1	0	2	965	3
9	3	1	0	0	3	3	0	1	1	997

Table 10. Confusion matrix for the Neural network (CNN) model trained and tested on the MNIST dataset. (Rows represent targets, columns represent predictions)

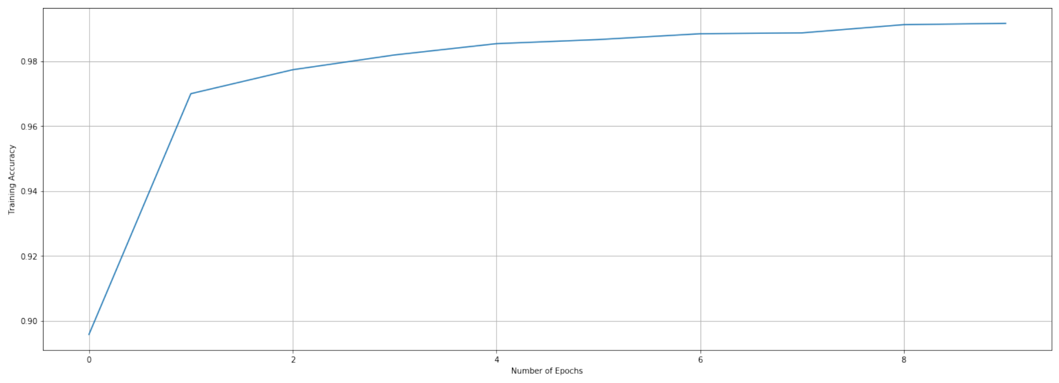

Figure 3: Training Accuracy of the CNN model over the number of epochs.

4.6 Ensemble classifier

The ensemble classifier, built by combining predictions of all 5 classifiers modelled so far using majority voting gave a test accuracy of 97.28% on the MNIST dataset and 46% on the USPS dataset.

The confusion matrix of this classifier is given in the following table.

	0	1	2	3	4	5	6	7	8	9
0	977	0	6	0	1	4	4	4	4	5
1	0	1124	0	2	0	0	2	11	3	5
2	1	2	1006	12	1	1	0	10	7	1
3	0	2	2	977	0	10	0	1	8	8
4	0	0	1	0	957	1	3	1	5	7
5	1	2	1	4	0	875	2	0	6	1
6	3	2	4	0	5	5	932	0	5	0
7	0	0	5	6	0	0	0	998	1	8
8	2	1	5	5	2	3	0	0	933	4
9	0	0	1	3	11	2	0	6	7	973

Table 11. Confusion matrix for the ensemble classifier trained and tested on the MNIST dataset. (Rows represent targets, columns represent predictions)

4.6.1 Observations

The model performs considerably better than the logistic regression, SVM and the random forest classifiers, but does not do as well as the neural networks. This is a consequence of the aggregating effect that is brought on the predictions of the 5 classifiers. Since the neural networks performed significantly better than other classifiers, their predictions would have been over-ridden and ignored if the other classifiers collectively predicted the wrong label.

5 Conclusion

Performance on USPS dataset All the classifiers trained gave good accuracies of atleast 90% on the MNIST dataset. The USPS dataset, used for testing however gave a very different result with accuracies between 30% and 60% for different classifiers. Although both the datasets were image samples of hand-written numeric digits, the classifiers performed differently on each of the datasets. This can be attributed to the differences in the data extraction and pre-processing steps of the 2 datasets. This essentially changes the way features are generated, resulting in 2 dissimilar datasets that had the same source. Since all the classifiers were trained specifically on the MNIST dataset, the models cannot be expected to do well on other datasets, since an algorithm which solves a specific problem cannot do well on a general set of problems, in accordance to the 'No free lunch theorem'.

Comparison of classifiers Of all the classifiers trained so far, the neural networks gave the best accuracies on both testing datasets, with the CNN performing a little better. On observing the classification matrices of each classifier it is evident that the neural networks get almost every digit right with very few mistakes. CNN performed better because of its ability to train features considering their 2D spatial correlations, which is great for 2D inputs such as images.

Other classifiers did not perform as well. Logistic regression was a simpler linear classifier that was easier to code from scratch, and had a reasonable performance but was not able learn non-linear features. SVM was computationally expensive and took a lot of time to train the data for an accuracy that was similar to logistic regression. Although it had the ability to train non-linear features better, it was not able to match up to that of neural networks. Random Forest performed better than SVMs and were able to converge quickly due to its ability to compute decisions in parallel. It also did not overfit too much due to its bagging strategy, but overall was not as accurate as the neural networks.

Performance of the ensemble classifier As described in the previous section, the model performs considerably better than the logistic regression, SVM and the random forest classifiers, but does not do as well as the neural networks. This is a consequence of the aggregating effect that is brought on the predictions of the 5 classifiers. Since the neural networks performed significantly better than other classifiers, their predictions would have been over-ridden and ignored if the other classifiers collectively predicted the wrong label.

References

[1] Numpy documentation
`https://docs.scipy.org/doc/numpy/reference/`

[2] Keras documentation
`https://keras.io/`

[3] SVM documentation
`https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`

[4] Random Forest documentation
`https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

[5] Code posted in UB learns, and used in previous projects.

[6] Gradient Descent Algorithms, by Sebastian Ruder.
`http://ruder.io/optimizing-gradient-descent/`