Digit recognizer

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this project we train and test a set of classifiers for pattern analysis in solving handwritten digit recognition problems, using MNIST database. The classifying methods implemented and tested include Ridge Regression, Support Vector Machine (with Principal Component Analysis), K-Nearest Neighbours, Neural Network and Random Forests. (Find our code here `https://github.com/PencilZQ/CS532-final-project.git`)

1 Introduction

Description of Dataset The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive.

Visualization of Dataset A bunch of examples are plotted as Figure 1 below to help in visualizing the data set. In the figure following that, a histogram in Figure 2 shows the break down of data points, based on class, to show that the data is uniformly distributed.

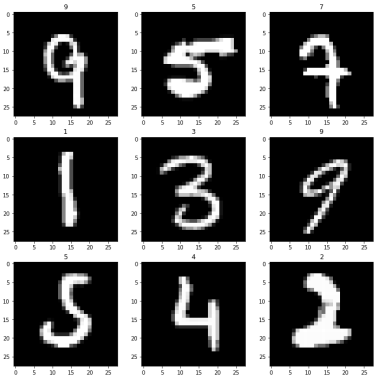

Figure 1: Examples

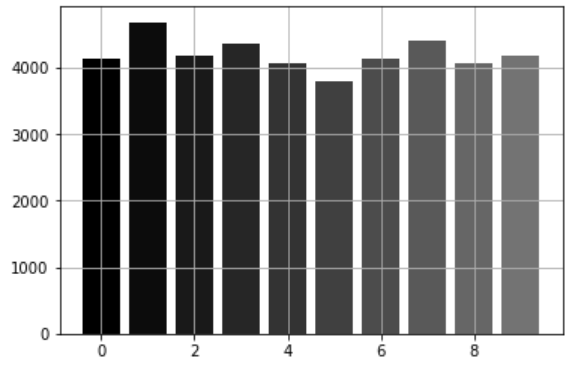

Figure 2: Data distribution

Methodology Various models will be compared to each other based on the accuracy of prediction. The accuracy will be based on error-rates generated by the models. For each model, the error rate has been calculated by dividing the dataset into training and test datasets. The training model is fed to each model to calculates respective weights/parameters. Then based on the number of correct predictions on the test data, the accuracy is calculated.

Multi-class classification In machine learning, multiclass or multinomial classification is the problem of classifying instances into one of three or more classes. While some classification algorithms naturally permit the use of more than two classes, others are by nature binary algorithms; these can, however, be turned into multinomial classifiers by a variety of strategies. For example, Support vector machines are based upon the idea of maximizing the margin i.e. maximizing the minimum distance from the separating hyperplane to the nearest example, the basic SVM supports only binary classification, but certain extensions can be used to handle the multiclass classification case as well. In these extensions, additional parameters and constraints are added to the optimization problem to handle the separation of the different classes.

2 Models

2.1 Ridge regression & Multi-class classification

Ridge regression can be used in binary classification problem, and in this classification problem, the number of classes is 10, so we can design 10 binary ridge regression classifiers to solve this classification problem. Take 3 classifiers as example, our idea of ridge regression and multi-class classification is shown in Figure 3.

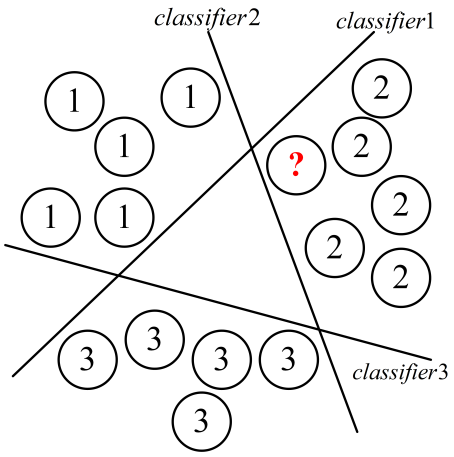

Figure 3: Ridge regression & Multi-class classification

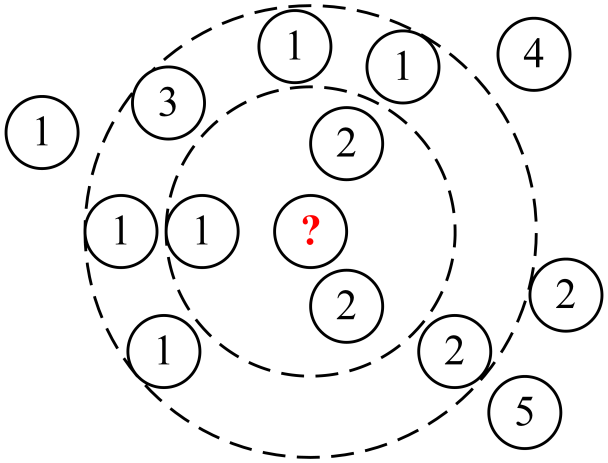

Figure 4: K-Nearest Neighbors algorithm

Take classifier 0 as an example, for all training samples, if their label is 0, then we set their labels to +1, otherwise, we set their labels to -1, and then we can formulate least problem from these classifier.

$$\hat{w} = \arg\min_{w} \|X \cdot w - y\| + \lambda \cdot \|w\|^2 = (X^T \cdot X + \lambda \cdot I)^{-1} \cdot X^T \cdot y$$

Where X is our training data, and y is the label, and $\hat{w}$ is the weight solved. And we do the same thing for classifier 1-9, and when we have a new sample, we use 10 different w from 10 classifiers to calculate the labels of the new sample, and we use the largest label calculated to decide the label of new sample.

2.2 K-Nearest Neighbors Algorithm

The k-nearest neighbors algorithm (k-NN) is one of the non-parametric methods used for classification. The idea of k-NN is straight forward. In k-NN classification, an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). Our idea of using k-NN in this classification problem is shown in Figure 4. If we choose k=3, then we choose 3 training samples, which are closer to new sample than other training samples, as shown in the first circle. In the first circle, we let the closest training samples vote for new sample, and because label 2 is the majority, we assign label 2 to the new sample. And if we choose k=8, then we choose 8 training samples, which are closer to new sample than other training samples, as shown in the second circle. In the second circle, we let the closest training samples vote for new sample, and because label 1 is the majority, we assign label 1 to the new sample. In this project, since the data are in 784 dimensional space, the computational cost is relatively high. It is sensible to pick a small k value. we run the algorithm for k from 1 to 30 with 2 as step size and compute the classification accuracy on the training data, and choose k which achieves best classification performance.

Algorithm 1 K-Nearest Neighbors algorithm

Input: Training samples, new sample, k **Output:** Label of new sample

1: Calculate distances between new sample with each training sample.
2: Order training samples according to distances in ascending order.
3: Choose first k training samples.
4: Choose the label of majority of chosen samples.
5: **return** Label chosen.

2.3 Support Vector Machine

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well.

2.3.1 Computing SVM

Computing the SVM classifier amounts to minimizing an expression of the form-

$$(1/n)\sum_{i=1}^{n} max(0, 1 - yi(w.xi - b)) + \lambda||w||^2$$

Henceforth we've discussed some of the common approaches adopted for this minimization problem, including the one which will be used in our SVM model.

Sub-gradient descent Sub-gradient descent algorithms for the SVM work directly with the expression:

$$f(w, b) = (1/n)\sum_{i=1}^{n} max(0, 1 - yi(w.xi - b)) + \lambda||w||^2$$

Note that f is a convex function of w and b. As such, traditional gradient descent (or SGD) methods can be adapted, where instead of taking a step in the direction of the functions gradient, a step is taken in the direction of a vector selected from the function's sub-gradient. This approach has the advantage that, for certain implementations, the number of iterations does not scale with n, the number of data points.

SVM and the hinge loss Another perspective is to see that the SVM technique is equivalent to empirical risk minimization with Tikhonov regularization, where in this case the loss function is the hinge loss.

$$l(y, z) = max(0, 1 - yz)$$

From this perspective, SVM is closely related to other fundamental classification algorithms such as regularized least-squares and logistic regression. The difference between the three lies in the choice of loss function: regularized least-squares amounts to empirical risk minimization with the square-loss-

$$l(y, z) = (y - z)^2$$

2.3.2 Parameter selection

The effectiveness of SVM depends on the selection of kernel, the kernel's parameters, and soft margin parameter C. A common choice is a Gaussian kernel, which has a single parameter γ . The best combination of C and γ is often selected by a grid search with exponentially growing sequences of C and γ , for example $C \in \{2^{-5}, 2^{-3}, \ldots, 2^{13}, 2^{15}\}$ and $\gamma \in \{2^{-15}, 2^{-13}, \ldots, 2^{1}, 2^{3}\}$ Typically, each combination of parameter choices is checked using cross validation, and the parameters with best cross-validation accuracy are picked. Alternatively, recent work in Bayesian optimization can be used to select C and γ , often requiring the evaluation of far fewer parameter combinations than grid search. The final model, which is used for testing and for classifying new data, is then trained on the whole training set using the selected parameters.
Our implementation of SVM is based on LibSVM(LIBSVM is an integrated software for support vector classification, regression and distribution estimation. It supports multi-class classification. It has been developed and maintained by Chih-Chung Chang and Chih-Jen Lin)[1]

2.3.3 Principle Component Analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. Such dimension reduction can be a very useful step for visualizing and processing high-dimensional data set, while still retaining as much of the variance in the data set as possible. To visualize the principle components, we have plotted the first and last component in the form of a grid, which is shown in Figure 5. The mix of black and white pixels in the image of first component shows that is captures a lot of variance, while the last principle component is dominated by black pixels, there-by showing that it captures very little variance of the original data set. Variance captured when we choose different number of PCA dimensions is shown in Figure 6.

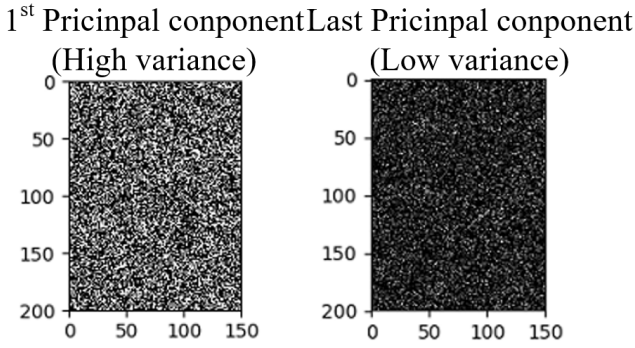

Figure 5: Principal components

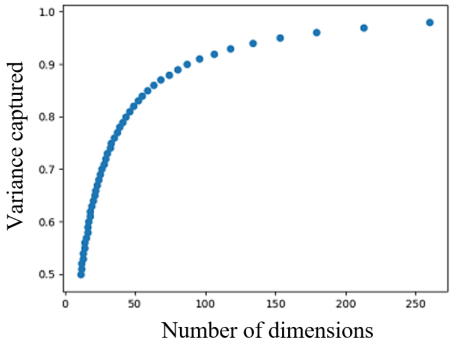

Figure 6: Variance captured

In regression analysis, the larger the number of explanatory variables allowed, the greater is the chance of overfitting the model, producing conclusions that fail to generalize to other datasets. One approach, especially when there are strong correlations between different possible explanatory variables, is to reduce them to a few principal components. We have exploited this idea while implementing the Support Vector Machine classification with PCA, i.e. we'll be performing SVM analysis on a dimensionally compressed dataset using PCA, without compromising on the accuracy of the SVM model. The figure that follows depicts the number of dimensions/principle components needed to capture varying amount of variance. It is exciting to note that 99% of variance has been captured using approximately half the number of dimensions we initially had in the dataset!

2.4 Neural Network

We used neural network on this classification task. The neural network we built consists of 784 nodes in input layer, 500 nodes in first hidden layer, 300 nodes in second layer, and 10 nodes in output layer, which is shown in Figure 7.

The input is $x_i, i = 1...784$ and x_0 is constant bias. The output of first hidden layer is $h^1_j, j = 1...500$ and h^1_0 is constant bias. The output of second hidden layer is $h^2_m, m = 1...300$ and h^2_0 is constant bias. The output of output layer is $\hat{y}_n, n = 1...10$. The weights between input layer and first hidden layer are $w_{i,j}$. The weights between first hidden layer and second hidden layer are $v_{j,m}$. The weights between second hidden layer and output layer are $u_{m,n}$.

Data preprocessing Firstly, because the value of every pixel of every sample is between0-255, we need to normalize these values to range 0-1 to avoid error caused by precision problem of computer. We implemented this by dividing sample data by 255. Then, we need to implement one hot encoding for labels of our data. One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. Because we have 10 classes in our labels, if a label is 1, then it's one hot encoding is $[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$. And python implementation is as follows. [5] showed that elastic distortion can improve the performance of neural network. So we used elastic transformation to preprocess our data. The results are shown in Figure 8.

Activation functions We used Relu (Rectifier Linear Unit) as the activation function of the hidden layers. This function is similar to the functioning of the biological neuron, and allow us to run the neural network model more fast. $R(x)$ and its derivative are as follows.

$$Relu : R(x) = max(x, 0); \;\; Derivative : R^{'}(x) = \mathrm{I}(x)$$

Where I is a indicator, it means if x is greater than 0, and the value of indicator is 1, otherwise, it's value is 0.

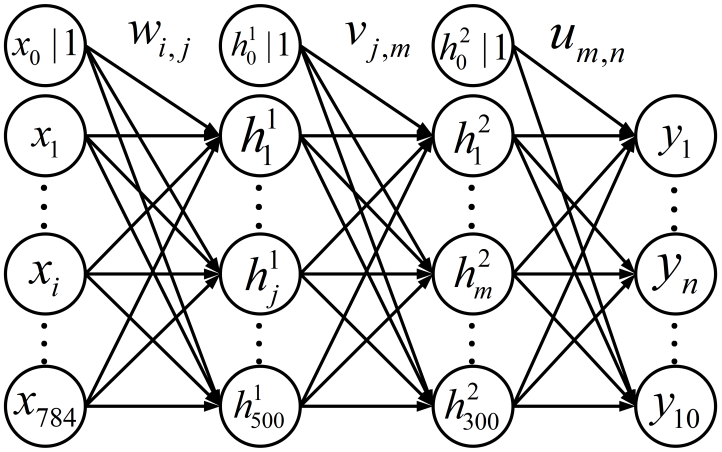

Figure 7: Neural Network

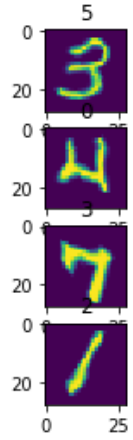

Figure 8: Elastic transformation

We used Softmax function as the activation function of the output layer. In probability theory, the output of the Softmax function can be used to represent a categorical distribution. So it's used in various multiclass classification. $S(\mathbf{x})_i$ and its derivative of are as follows.

$$Softmax : S(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{10} e^{x_j}}, i = 1...10; \;\; Derivative : S^{'}(\mathbf{x})_i = S(\mathbf{x})_i \cdot (1 - S(\mathbf{x})_i)$$

Dropout Dropout is technique for addressing overfitting problem in neural network. The key idea is to randomly drop units (along with their connections) from the neural network during training[6,7]. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods.

Metrics We choose Cross entropy loss function as our loss function.

$$Cross\ entropy\ loss : L(y|\hat{y}) = -\frac{1}{N} \cdot \sum_{i=1}^{N} [y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})]$$

Where N is the number of samples, and y is the correct label, and $\hat{y}$ is the predicted label. And the derivative of $S(\mathbf{x})_i$ is as follows. And we use accuracy rate to evaluate the performance of our neural network.

$$Derivative : L^{'}(y|\hat{y}) = -\frac{1}{N} \cdot \frac{y - \hat{y}}{\hat{y} \cdot (1 - \hat{y})}$$

Forward propagation When we move forward through the network from inputs to outputs, we have these calculations.

$$h_j^1 = R(\sum_{i=0}^{784} x_i \cdot w_{i,j}), j = 1...500;$$

$$h_m^2 = R(\sum_{j=0}^{500} h_j^1 \cdot v_{j,m}), m = 1...300; \;\; \hat{y}_n = S([\sum_{m=0}^{300} h_m^2 \cdot u_{m,n}]_{n=1...10})_n, n = 1...10$$

Stochastic gradient descent We use Stochastic gradient descent to train our neural network. Take one sample for example. Firstly, we calculate the gradient of loss function to $u_{m,n}$. Because the weight is only relevant with y_n,so we only care about loss caused by $\hat{y}_n$.

$$\begin{aligned} \nabla_{u_{m,n}} L(y_n|\hat{y}_n) &= \frac{dL}{d\hat{y}_n} \cdot \frac{d(\hat{y}_n)}{dI(\hat{y}_n)} \cdot \frac{dI(\hat{y}_n)}{u_{m,n}} \\ &= \frac{y_n - \hat{y}_n}{\hat{y}_n \cdot (1 - \hat{y}_n)} \cdot \hat{y}_n \cdot (1 - \hat{y}_n) \cdot h_m^2 \\ &= (y_n - \hat{y}_n) \cdot h_m^2 \end{aligned}$$

And then we calculate the gradient of loss function to $v_{j,m}$.

$$\begin{aligned}\nabla_{v_{j,m}} L(y|\hat{y}) &= \sum_{n=1}^{10} \frac{dL}{d\hat{y}_n} \cdot \frac{d(\hat{y}_n)}{dI(\hat{y}_n)} \cdot \frac{dI(\hat{y}_n)}{dh_m^2} \cdot \frac{dh_m^2}{dI(h_m^2)} \cdot \frac{dI(h_m^2)}{dv_{j,m}} \\ &= \sum_{n=1}^{10} \frac{y_n - \hat{y}_n}{\hat{y}_n \cdot (1-\hat{y}_n)} \cdot \hat{y}_n \cdot (1-\hat{y}_n) \cdot u_{m,n} \cdot \mathrm{I}(I(h_m^2)) \cdot h_j^1 \\ &= \sum_{n=1}^{10} (y_n - \hat{y}_n) \cdot u_{m,n} \cdot \mathrm{I}(I(h_m^2)) \cdot h_j^1\end{aligned}$$

And then we calculate the gradient of loss function to $w_{i,j}$.

$$\begin{aligned}\nabla_{w_{i,j}} L(y|\hat{y}) &= \sum_{m=1}^{300} \sum_{n=1}^{10} \frac{dL}{d\hat{y}_n} \cdot \frac{d(\hat{y}_n)}{dI(\hat{y}_n)} \cdot \frac{dI(\hat{y}_n)}{dh_m^2} \cdot \frac{dh_m^2}{dI(h_m^2)} \cdot \frac{dI(h_m^2)}{dh_j^1} \cdot \frac{dh_j^1}{dI(h_j^1)} \cdot \frac{dI(h_j^1)}{w_{i,j}} \\ &= \sum_{m=1}^{300} \sum_{n=1}^{10} \frac{y_n - \hat{y}_n}{\hat{y}_n \cdot (1-\hat{y}_n)} \cdot \hat{y}_n \cdot (1-\hat{y}_n) \cdot u_{m,n} \cdot \mathrm{I}(I(h_m^2)) \cdot v_{j,m} \cdot \mathrm{I}(I(h_j^1)) \cdot x_i \\ &= \sum_{m=1}^{300} \sum_{n=1}^{10} (y_n - \hat{y}_n) \cdot u_{m,n} \cdot \mathrm{I}(I(h_m^2)) \cdot v_{j,m} \cdot \mathrm{I}(I(h_j^1)) \cdot x_i\end{aligned}$$

The SGD in training have a lot of oscillations, so the momentum term was invented. The momentum term is used for soft the oscillations and accelerates of the convergence. Take weights between input layer and the first hidden layer as an example. $M_{i,j}$ is the momentum term, so the gradient descent process is like this.

$$\begin{aligned}M_{i,j}(k+1) &= \gamma \cdot M_{i,j}(k) + \eta \cdot \nabla_{w_{i,j}(k)} L(y|\hat{y}) \\ w_{i,j}(k+1) &= w_{i,j}(k) - M_{i,j}(k+1)\end{aligned}$$

2.5 Random forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification)[4]. This algorithm has a lot of advantages: (1) It can handle thousands of input variables without variable deletion; (2) It gives estimates of what variables are important in the classification; (3) It generates an internal unbiased estimate of the generalization error as the forest building progresses; (4) It runs efficiently on large data bases; For every tree in the forest, we used bootstrapping to choose samples to do the training, and choose f features from all features randomly to do the splitting. And when we get new samples, we used every tree to decide distribution of classes of the new sample, and make the final decision based on decisions of all trees. Let f be 2 (the number of features in experiment won't be so small), and an example of random forest is shown as Figure 9.

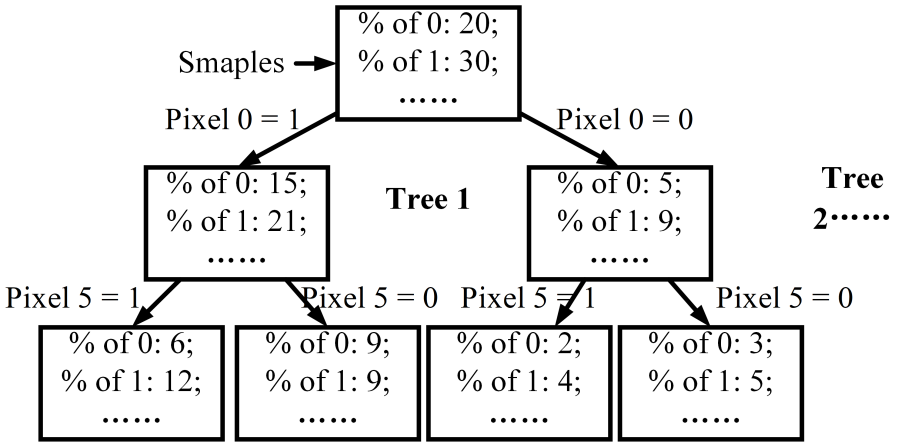

Figure 9: Random forest

3 Results and Conclusion

The results of all the discussed models have been summarized in Table 1.

Some very interesting conclusions can be drawn-
i) The Neural Network model has the highest accuracy rate (Accuracy: NN>k-NN>SVM>Random forest>Redge regression).

ii) Ridge regression is not useful in solving classification proble, although the trainig time is very low. K-NN can achieve the sceond higest accuracy rate, but the training cost is too high. NN and SVM have similar training cost, so NN is better than SVM. Random forest has similar performance with SVM, but the efficiency of RF is the highest.

iii) PCA is good way to improve efficiency. When implemented with PCA, efficiency of SVM is improved without deteriorating performance too much.

Table 1: Experiment results

Comparison of models		
Model	Accuracy (%)	Time(s)
Ridge Regression	84.17	7.93
SVM	95.21	306.44
k-NN clustering	96.45	674.65
Neural Network	98.65	287.54
Random Forest	94.07	2.74

SVM with PCA		
Dimension	Accuracy(%)	Time(s)
50	88.23	42.87
100	94.89	57.44
200	95.15	131.46
250	95.17	181.22
784(Without PCA)	94.21	306.44

Learning outcomes and connection to CS 532 Some of the methods we have implemented(NN,SVM,Ridge Reg.) were covered in the class. We were able to successfully apply them to a huge dataset, and obtain tangible results and knowledge,

Other methods(k-NN clus, Random Forests) were beyond the scope of the class, but because these models are also supervised learning, and building upon the concepts we learned throughout the course; we were able to develop a strong mathematical intuition and successfully implement these models on a large dataset.

Lastly, we were able to combine the concepts learned, like PCA for dimensionality reduction followed by SVM, and obtain great accuracy in prediction.

References

[1] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM[J]. Journal of Machine Learning Research 6, 1889-1918, 2005.
[2] Matrix Methods in Data Mining and Pattern Recognition by Lars Elden.
[3] Pattern Recognition and Machine Learning by Christopher M. Bishop.
[4] Bernard S, Adam S, Heutte L. Using Random Forests for Handwritten Digit Recognition[C]. International Conference on Document Analysis and Recognition. IEEE, 1043-1047, 2007.
[5] Simard P Y, Steinkraus D, Platt J C. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis[C]. International Conference on Document Analysis and Recognition. Proceedings. IEEE, 958-964,2003.
[6] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. Journal of Machine Learning Research, 1929-1958, 2014.
[7] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Proceedings of the 30th International Conference on Machine Learning, 1058-1066, 2013.