Machine Learning

Digit Classification Project

Nicholas Murphy

27 November 2017

Student ID: MRPNIC004

Problem Description

In this project we develop a classification scheme that decides whether a handwritten digit in the form of an image, is even or odd. The idea is to use data sets consisiting of pixel values (1 for black, 0 for white) for the images whereby the first column of the data set is the true value (0 to 9) of the digit shown in the image, and columns 2 to 785 are the pixel values of a given image which can be reconstructed into a 28 by 28 grid to reveal the image of the handwritten digit.
The classification schemes will be trained on a training set consisting of 2500 images which include labels of the images and tested on a set consisting of 2500 images which do not contain labels for the images. We will implement the following classification schemes:

1. Pocket Learning Algorithm
2. Logistic Regression
3. Support Vector Machine
4. Neural Network
5. Random Forests
6. Boosting
7. Regression Trees

We will compare various results of the schemes, the most important being the cross validation error. We will also compare the accuracy of the schemes for various sizes of the training and validation sets as well as compare the accuracies both with and without Principal Component Analysis applied to the training set. We will then apply the trained models to the test data set to produce predictions. We also provide various plots of the resluts to get more insight into the performance of the schemes.

1 Data

As mentioned above, the training set contains 2500 images all contained in a .csv format. We will denote the training set by $\mathbf{X}$ which is a matrix of dimension 2500×785 and the output vector by $\mathbf{y}$ which is a column vector of length 2500. Below in Figure 1, illustrates the first 6 images (rows 1-6) of the training set

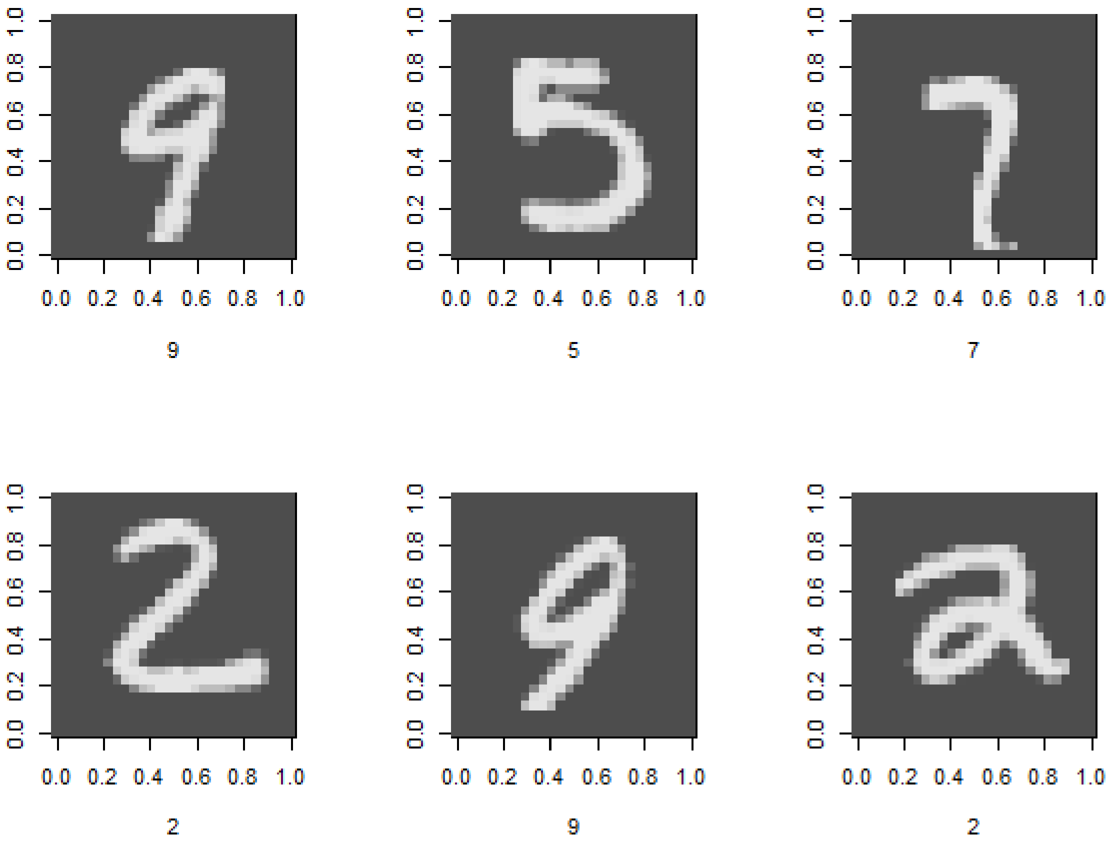

Figure 1: Illustrations of the first 6 digits in the training set with the true value of the digit shown below each image.

2 Dimension Reduction

Due to the nature of the data set, there are a large number of features/dimensions (pixels) and the data set itself consisits of a large number of images. Thus, it is often useful to reduce the dimensions of the data set that we feed into our algorithm in the attempt of getting a better sense of the relationship between variables and/or fetures.

2.1 Principal Component Analysis

Principal Component Analysis (PCA) is one such dimension reduction method. The basic idea of PCA is to represent the input as a lower dimensional input by finding an orthonormal basis that explains most of the variance in the data set which we can use to approximate the full set of features (785) using fewer features. In this project, we will not scale the images since the pixels can only take on a value of 0 and 1 and hence scaling will not lead to drastic improvements than if we were to have images with pixels ranging between 0 and 256.

We will choose the number of principal in our orthonormal basis to be the principal components that explain 99% of our variance. This is done by finding the ratio of the cumulative sum of the eigenvalues divided sum of all eigenvalues. Once this ratio reaches 99%, we have enough principlal components and in our case that number was 44.

3 Methods

3.1 Pocket Learning Algorithm

The Pocket Learning Algorithm (PLA) is a classification algorithm and is a modified version of the Perceptron Learning Algorithm. The Perceptron Learning Algorithm is designed to classify linearly separable data and hence will not terminate if the data is non-separable. The PLA is modified to 'keep in pocket' the weight which gives the lowest in sample error up to the current iteration [1]. The weight which gives the lowest in sample error throughout all of the iterations will be the final hypothesis chosen by the algorithm. The PLA is a linear model that creates a signal using a linear combination of the inputs[1]. A binary function will then be applied to the signal to produce an output of 0 or 1. In this project, the PLA will iterate through the training set and pick incorrectly classified images as represented by the rows of the training set and will attempt to correctly classify the image as being odd (0) or even (1) by using the updated weights as mentioned above.
The error measure of the model will be the number of incorrectly classified images remaining after the algorithm terminates divided by the number of images in the set (2500).

3.2 Logistic Regression

The PLA above is a linear classification algorithm which produces a binary output (± 1) however if we want to get the certainty/uncertainty of this output, we can implement Logistic Regression which outputs a probability value (output $\in [0, 1]$). This is accomplished by applying a logistic function[2] to the signal to get an output in the range [0,1].
The error of the measure is found by showing that maximimizing the likelihood of getting all the y_n's from the corresponding $x_n's$ is equivalent to:

[1] $\boldsymbol{x}^T\boldsymbol{w}$ where $\boldsymbol{x}$ is a vector of pixels for a given image and $\boldsymbol{w}$ is the weight vector to be updated by the algorithm

[2] The logistic function for the signal is: $\frac{e^{\boldsymbol{x}^T\boldsymbol{w}}}{1+e^{\boldsymbol{x}^T\boldsymbol{w}}}$

$$E_{in} = \frac{1}{N}\sum_{n=1}^{N} ln(1 + e^{-y_n \mathbf{w}^T \mathbf{x}})$$

The idea is to then use stochastic gradient descent to minimize this quantity.
Logistic regression performs well for linearly separable data. For non linearly separable data (as in our case) we would expect it to perform similarly to SVM.

3.3 Support Vector Machine

Support Vector Machine (SVM) is a classification scheme that attempts to find the optimal separation boundary (hyperplane) to separate the classes of the points. The key to finding the optimal boundary is maximizing the margin of the boundary. The margin is the distance between the boundary separating the points and the points that lie closest to it. The nice thing about SVM is that it is relatively simple and more importantly it copes well with non linearly separable data. It also deals well with high dimensional inputs and doesnt slow down drastically compared to other algorithms.
For non linearly separable data the idea is to allow for points which viloate the bounary to be misclassified and control the number of such points with a cost parameter, C. This is known as a soft margin SVM and C is the soft-margin cost function which controls the influence of each support vector on the overall optimization. Will now have a non linear boundary which will cause problems in calculating the distance between the boundary and the closest points. This problem is solved by introducing a kernal. The kernal allows us to compute vector products of inputs from a lower dimensional space in a transformed higher dimensional space which is rendered separable. Thus, we are able to classify non linearly separable data in a robust and simple manner.

3.4 Neural Network

A neural network is made up of a series of layers where each layer consisits of a series of nodes. The first layer is called the input layer where the input data is fed into the network. The hidden layers consist of weighted sums of the inputs. The weights are the parameters which we are searching for to optimize the model. The output layer then gives the output of the network which is then a weight sum of the node values in the last hidden layer. The algorithm will feed forward the information through the network to produce an output. We then feed this error back (backpropogation) through the network to update the weights (using stochastic gradient descent) and so the process continues until we get a desired result after a number of iterations. The difficulty with nearal networks is that we dont know how many nodes and layers is optimal.

3.5 Random Forests

Random Forests start by sampling points from the data set (Bootstrapped samples). We then build a decision tree from each subsample (bagged trees). However, eacg time a split is considered in these trees, a random sample of m¡p[3] (where p is the number of features in the training set) predictors are chosen as split candidates. This decorrelates the trees leads to better predictive performance.

3.6 Stochastic Gradient Boosting

Gradient boosting has 3 main components: a loss function to be optimized, a series of tree functions called weak learners used to make predictions and a function to sequentially add new trees to the model to reduce the loss function. Boosting tries to find the additive function $F : \mathbb{R}^d \to \mathbb{R}$ that adds the trees in such a way to minimize the loss:

$$L(F) = \sum_{i=1}^{n} log(1 + exp[-2F(x_i)\tau_i])$$

where F that minimizes $L(F)$ is given by:

$$F(x_i) = \frac{1}{2} log \frac{\text{\#observations} : x_i, \tau = 1}{\text{\#observations} : x_i, \tau = -1} \approx \frac{1}{2} log - odds(x)$$

Gradient descent will then be used used to minimize the loss function when adding trees. Stochastic gradient descent can then be used instead of gradient descent by taking a random subsample of the training data at each iteration. The subsample is then used to fit the base learner, instead of the full data set being used [3].

4 Cross Validation

Validation (cross validation) is a key component of machine learning as it provides us with an estimate of the out-of-sample error based on information available in sample [1]. We will focus on cross-validation (R-fold cross validation) which is a modified version of validation and gives us an unbiased estimated of the out of sample error by basically taking an average of validation errors. More specifically, we partition the training data into disjoint sets, $D_1, \dots, D_K$, each with a size of N/K where $N = |D|$. We then train on each set D_k to get a final hypothesis g^- which is then validated on the complement of the data set. We then take an average of these K validation values to get the cross-validation error.

5 Results

5.1 Pocket Learning Algorithm

We implemented the Pocket algorithm as explained above and wrote our own function for it. The result was a cross validation error 53.4% which was really poor.

[3] In R, the default value is m≈ $\sqrt{p}$

5.2 Logistic Regression

We use the ‘caret’ package and use the ‘train’ function with the ‘glm’ option. Since we are doing binomial classification, we choose the bernoulli method for the function. Due to the slow computational speed of the function, we use standard validation by choosing a training set to be the first 80% of the training data and the validation set to be the last 20%. The function is slow due to the gradient descent algorithm trying to minimize the cross entropy error for a data set of high dimension.

Logistic regression had a validation error of 83.5%.

5.3 Support Vector Machine

We implemented SVM using the ’e1071’ package in R. We tested the performance of the model by computing a 5-fold cross validation error for different combinations of cost parameters (C) and kernels. We chose 5-fold cross validation instead of 10-fold cross validation since 10-fold only improved on 5-fold by a few decimals but has much more expensive computing time. The table below shows the results of the 5-fold cross validation without PCA:

	0.01	1	5	30	80
polynomial	5.36%	5.36%	5.36%	5.36%	5.36%
linear	21.28%	21.28%	21.28%	21.28%	21.28%
radial	49.80%	49.80%	49.80%	49.80%	49.80%

Table 1: Without PCA: 5-fold cross validation errors for different values of C shown in the top of the table vs different kernals shown on the left of the table.

It is clear that the cost parameter (C) does not affect the outcome of the process for any of the kernals. It is also obvious that the polynomial kernal outperforms the linear and the radial basis kernal. We thus choose a cost parameter of 1 and the polynomial kernal for predicitng on the test set.

The table below shows the results of 5-fold cross validation with PCA:

	0.01	1	5	30	80
polynomial	5.56%	5.56%	5.56%	5.56%	5.56%
linear	16.08%	42.32%	42.32%	42.32%	42.32%
radial	51.24%	51.24%	51.24%	51.24%	51.24%

Table 2: With PCA: 5-fold cross validation errors for different values of C shown in the top of the table vs different kernals shown on the left of the table.

Thus doing PCA does not improve the accuracy. From the 2 tables above, the optimal accuracy for SVM is: 94.64%

5.4 Neural Network

We implement a Neural Network using the ‘nnet’ function in R. Instead of implementing cross validation (since the computational time will be hours), we use standard validation by choosing a training set to be the first 80% of the training data and the validation set to be the last 20%. We use 1 hidden layer with 40 nodes. We chose our activation function to be the logistic function since our true output is chosen to be 0 or 1 so we can round the final output of the neural network to get predictions which we can compare to our true output.

We got a validation error 15.8% of and hence and accuracy of 84.2%. The algorithm was the slowest of all and took over 40 minutes to compute with a performance not much better than any of the other algorithms.

5.5 Random Forests

We implemented Random Forests using the ’randomForest’ package in R. We implement 5-fold cross validation to compute the cross validation errors for different values for the number of trees used in he algorithm. The table below shows the results of Random Forest without PCA:

No. Trees	E_{cv}
10	8.88%
30	7.36%
60	6.12%
100	6.56%
200	6.48%
300	6.04%

Table 3: Without PCA: 5-fold cross validation errors for different numbers of trees used in the random forest function.

There is no drastic improvement in the cross validation error for trees larger than 60. In fact, 100 and 200 trees leads to a larger error than for 60 trees so we choose 60 trees to be our optimal number of trees to predict on the test data.

The table below shows the 5-fold cross validation errors for Random Forests with PCA:

No. Trees	E_{cv}
10	13.32%
30	10.32%
60	8.72%
100	8.48 %
200	8.32 %
300	8.28 %

Table 4: Without PCA: 5-fold cross validation errors for different numbers of trees used in the random forest function.

Thus, PCA does not improve the cross validation errors.
The optimal (optimal parameter values) accuracy for Random Forests is: 93.88%.

The figure below shows the error plot for random forest for different numbers of trees when the model is trained on the full training set. As can be seen, as we increase the number of trees past 100, there is barely an increase in the accuracy of the model.

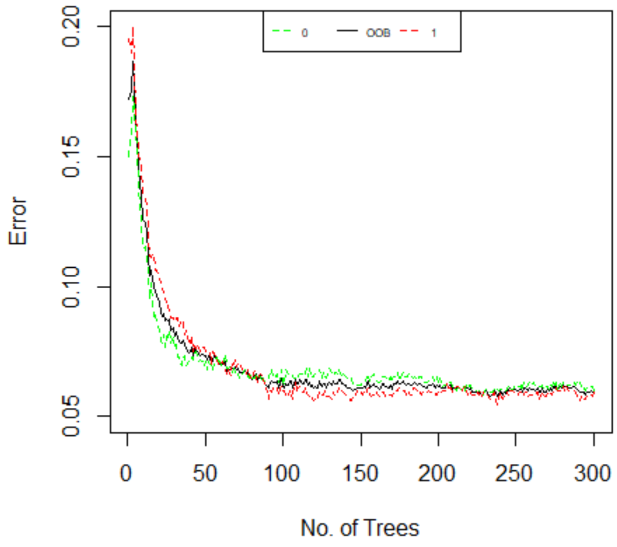

Figure 2: Overall Out of Bag error and the errors for each of the classes (0 and 1).

5.6 Boosting

To implement Boosting, we implement the 'gbm' package in R. We implement 5-fold cross validation to choose the optimal combination of shrinkage parameter and number of trees used in the model. The table below shows the cross validation errors for the various combinations when PCA is not used:

trees/ shrinkage	0.1	0.01	0.001
100	15.08%	24.52%	27.32%
400	13.48%	18.84%	26.64%
800	13.60%	16.28%	25.36%
1500	13.80%	14.72%	22.60%

Table 5: Without PCA: 5-fold cross validation errors for different combinations of shrinkage parameters shown in the top of the table and different numbers of trees shown on the left.

It is clear that using more than 400 trees doesnt lead to smaller cross validation errors for a shrinkage parameter of 0.1 which gives us our lowest error. Hence, we choose a shrinkage parameter of 0.1 and 400 trees to train our boosting model on all of the training data to produce a prediction on the test data.

The table below shows the cross validation errors for the various combinations when PCA is used:

trees/ shrinkage	0.1	0.01	0.001
100	17.84%	26.16%	31.88%
400	14.28%	21.68%	31.80%
800	13.08%	17.80%	27.68%
1500	13.08%	15.84%	25.40%

Table 6: With PCA: 5-fold cross validation errors for different combinations of shrinkage parameters shown in the top of the table and different numbers of trees shown on the left.

The optimal (optimal parameter values) accuracy for Boosting is: 86.52%.

5.7 Recursive Partitioning Trees

We implent classification trees using the recursive partitioning tree function, 'rpart', in R. We use the 'rpart.control' function to create a stopping criterion for the tree. We choose the minimum number of observations that must exist in a node in order for a split to be attempted to be 10 and the minimum number of observations in the terminal nodes to be 5.

We get an accuracy of 84.72% without PCA and 81.76% with PCA.

Figure 3 shows the recursive partitioning tree trained on the full training set. The algorithm, as illustrated by the diagram, starts with the complete data set in the first node right at the top of the tree and chooses the best feature to split into the next 2 nodes. In each box, the number at the top represents the way that the node has voted, the numbers below that tell us the proportion of pixels are less (left) and greater than (right) the pixel value shown below the box and the last number in the box is the percentage of the total data set used in the decision at a given node.

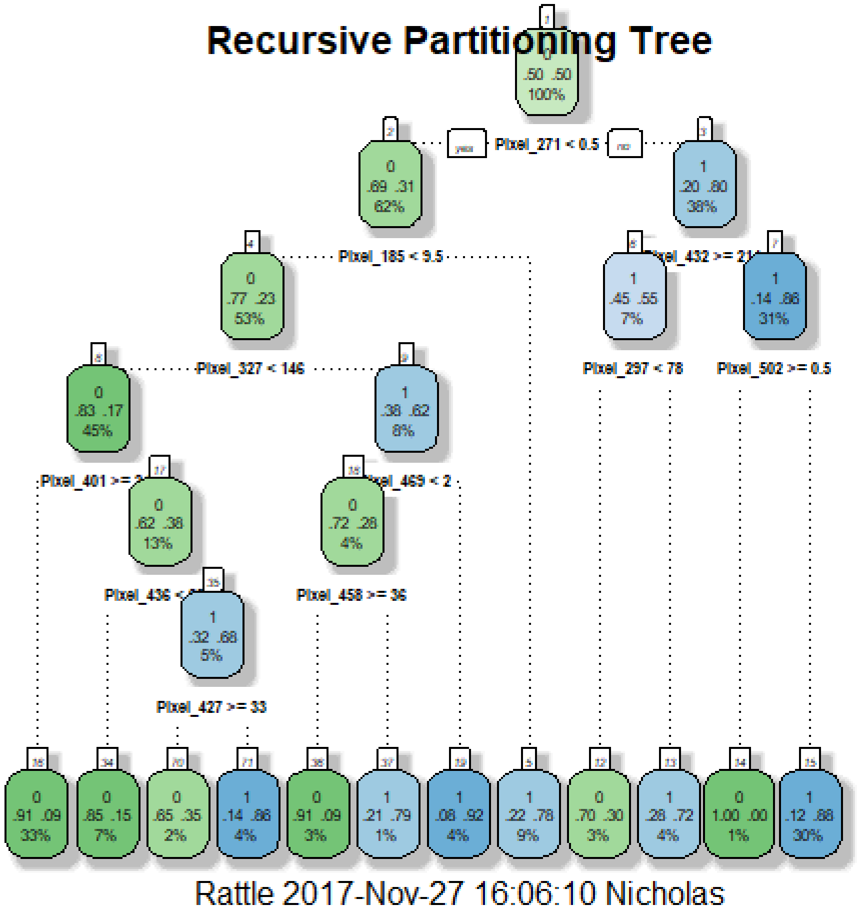

Figure 3: Recursive partitioning tree for the full training set.

6 Conclusion

It is clear that the use of PCA did not lead to drastic improvements in the accuracy of the algorithms although it did lead to slight improvements in the speed of the algorithms. SVM outperformed all of the other algorithms with an accuracy of 94.64%. The Random Forest algorithm also performs relatively well and has an accuracy of 93.88%. The neural network was the slowest algorithm of all and took over 40 minutes to run without cross validation. Logistic Regression was also very slow due to the gradient boosting step needed to minimize the in sample (cross entropy) error. The simpler models, Boosting and Recursive Partitioning trees didnt perform as well, and had accuracies of 86.52% and 84.72% respectively.

The attached .csv file, Predictions.csv, includes the predictions of all of the studied algorithms on the test set which does not include the true values of the digits. It would be interesting to test whether SVM outperforms the other on the test set as well to conclude that it has great in sample and out of sample performance for the chose parameters.

References

[1] Yaser S Abu-Mostafa, Malik Magdon-Ismail, & Hsuan-Tien Lin. Learning from data, volume 4. AML Book New York, NY, USA:, 2012.

[2] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. The Elements of Statistical Learning, volume 1. Springer Series in Statistics New York, 2001.

[3] J. Friedman (1999). Stochastic Gradient Boosting.

Appendix

R Code

```
## Machine Learning Project
#
#                 Author: N. Murphy
#
## Version: Masters-Coursework-ML-ML-Project-NMURPHY.R
#
## Version Control:
# ~/R/ML/Assignents/Project/
#
# Problem Specification: This script implements Q1 of assignment 2
# Data Specification: None
# Configuration Control: userpath/MATLAB/Master/Coursework/ML
# Version Control: No version control
# References: None

## 1. Data Description
#
# None

## 2. Clear workspace
rm(list=ls()) # clear environment

## 3. Paths
# 3.1. setwd("<location of your dataset>")
rootp <- getwd()
setwd("..") # move up one level in the directory tree
# setwd(path.expand('~'))
filentrain <- "/Train_Digits_20171108.csv"
filentest <- "/Test_Digits_20171108.csv"
fpath <- "~/R/ML/Assignments/Project"
ffilentrain <- paste(fpath,filentrain,sep="")
ffilentest <- paste(fpath,filentest,sep="")

## 4. Load Train Data
traindata <- read.csv(ffilentrain)
testdata <- read.csv(ffilentest)

## 5. Load libraries
library(e1071)
library(randomForest)
library(neuralnet)
library(nnet)
library(rpart)
library(rattle)
library(gbm)
library(tree)
library(glm)


####################
```