CHARACTER RECOGNITION

STAT 602 Final Project

Thomas Pattara

August 25,2019

Contents

Abstract

Off-line handwriting recognition involves the automatic conversion of text in an image into letter codes which are usable within computer and text-processing applications. The data obtained by this form is regarded as a static representation of handwriting. Off-line handwriting recognition is comparatively difficult, as different people have different handwriting styles. And, as of today, OCR engines are primarily focused on machine printed text and ICR for hand "printed" (written in capital letters) text.

1. Problem Statement

In handwritten character recognition, the characters are written by the individuals on a sheet of paper which is then stored into the computer by using scanner and after that these characters are recognized. Off-line handwriting recognition is comparatively difficult, as different people have different handwriting styles.

2. Background

We have been given two datasets:

- letters.unlabeled.csv: Is a csv file composed of 10000 unlabeled handwritten characters.
- letters.labeled.csv: Is a csv file composed of 1000 labeled handwritten characters.

The goal: To build a classifier from the given daata that cab recognize a user written symbol belonging to a certain alpha-numeric character. To achieve this, and to get more data for training, we will label the unlabelled dataset and use it as the training data, and validate the models using the labeled data as the validation set.

3. Exploratory Data Analysis

3.1 Visualizing the Data

First, we look at the structure of the labeled data (letters.labeled.csv) and unlabeled data (letters.unlabeled.csv).

```
## Labeled data dimensions: 1000 3138
```

```
## Unlabeled data dimensions: 10000 3137
```

Table 1: Snippet of the Labeled Data (First 10 Rows and First 7 Columns)

X	Letter	Pixel.1	Pixel.2	Pixel.3	Pixel.4	Pixel.5
1	t	0	0	0	0	0
2	r	0	0	0	0	0
3	d	0	0	0	0	0
4	t	0	0	0	0	0
5	e	0	0	0	0	0
6	d	0	0	0	0	0
7	v	0	0	0	0	0
8	p	0	0	0	0	0
9	3	0	0	0	0	0
10	z	0	0	0	0	0

Table 2: Snippet of the Unlabeled Data (First 10 Rows and First 7 Columns)

X	Pixel.1	Pixel.2	Pixel.3	Pixel.4	Pixel.5	Pixel.6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0

The labeled data has 1,000 rows or observations and 3,138 columns. The first column is simply an index or observation number, which we can drop. The second column (*Letter*) is the label or class for each character, i.e. it is the response variable. Columns three to 3,138 (*Pixel.1* to *Pixel.3136*) represent the 3,136 pixels in the 56 by 56 bitmap. These pixels are the predictors and are binary, taking on a value of 0 if the pixel is white and a value of 1 if the pixel is black.

The unlabeled data has 10,000 rows or observations and 3,137 columns. Similar to the labeled data, the first column of the unlabeled data is an index, which we can drop; however, the unlabeled data does not contain *Letter* (the response variable).

```
## Number of NA values: 0
```

A good thing about both the labeled and unlabeled data is that there are no missing values, which makes our analysis easier.

Figure 1 plots the pixels of the first 36 characters from the labeled data, and **Figure 2** plot the pixel means of each character. We see that there is a large amount of variance in the data.

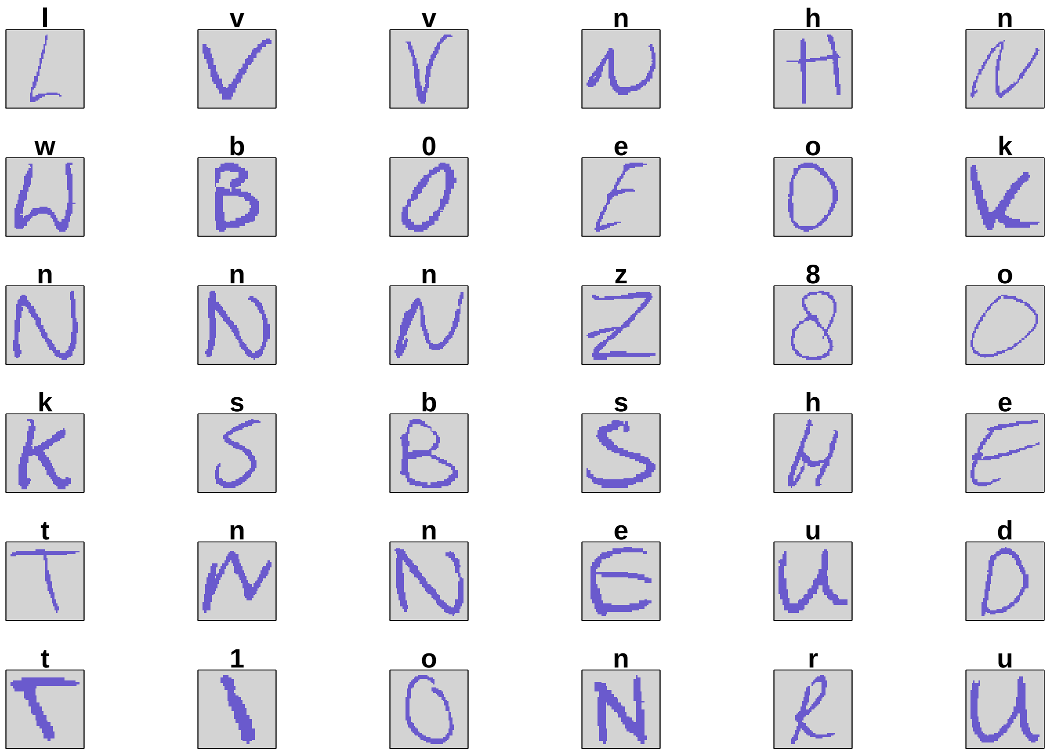


Figure 1: Plot of the pixels of the first 36 characters from the labeled data.

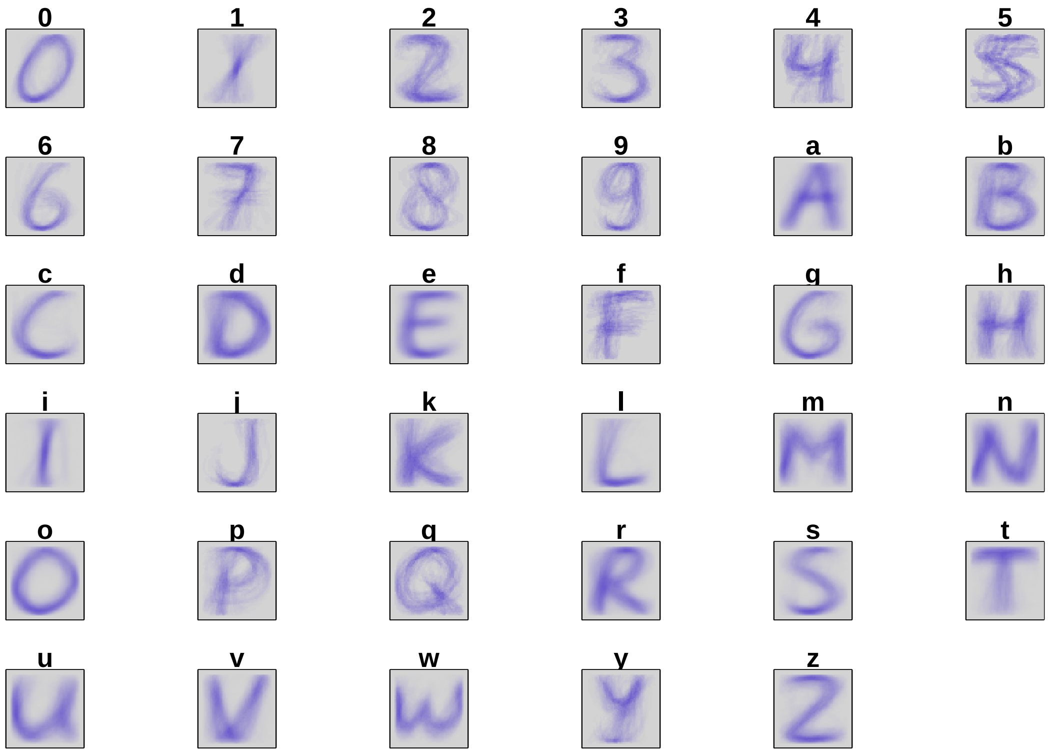


Figure 2: Plot of the pixel means of each character from the newly labeled unlabeled dataset.

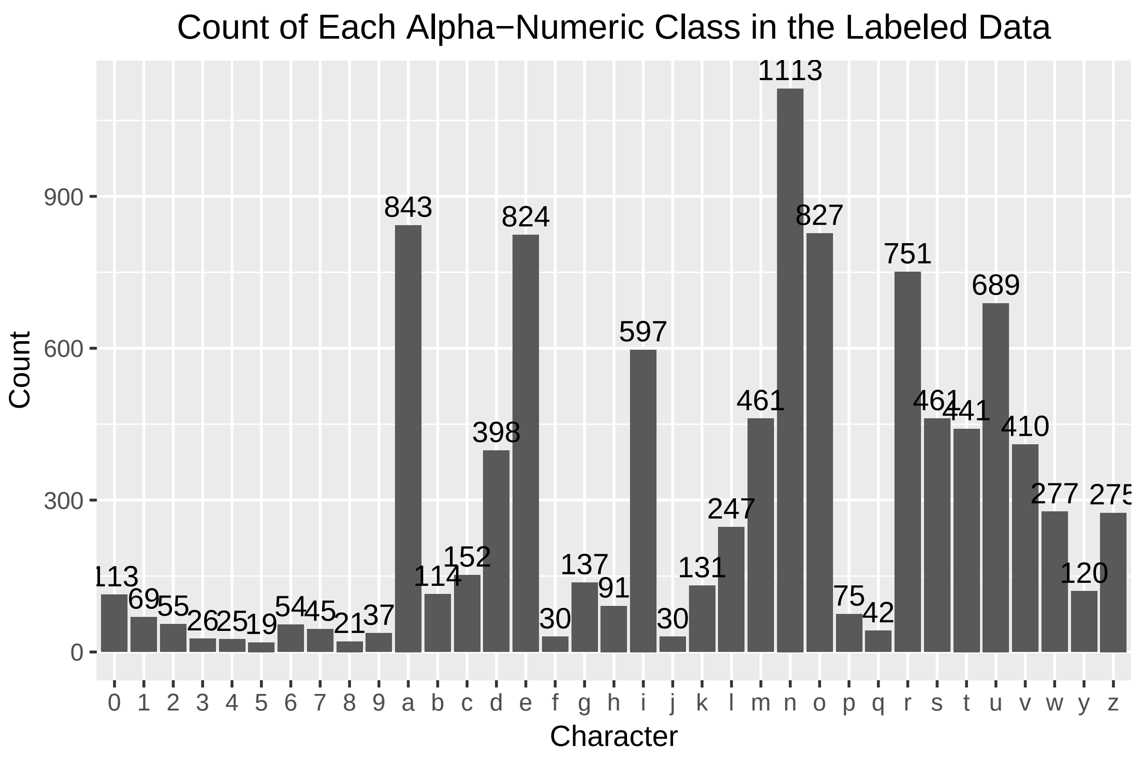


Figure 3: Count of each alpha-numeric class in the labeled data.

Figure 3 shows the count of each alpha-numeric class in the labeled unlabeld data. We see that the classes are heavily unbalanced, ranging from 1113 observations for the letter n and 30 observations each for f and j. We should also note that there is no letter x in this data. Overall, there are far fewer numbers than letters (464 numbers vs. 9536 letters).

3.2 Data Partitioning

Training data set will be the unlabeled data set (10,000) that we labeled Test data set will be the labeled data set (1,000) that we were given

```
## Training set dimensions: 10000 3137
```

```
## Validation set dimensions: 1000 3137
```

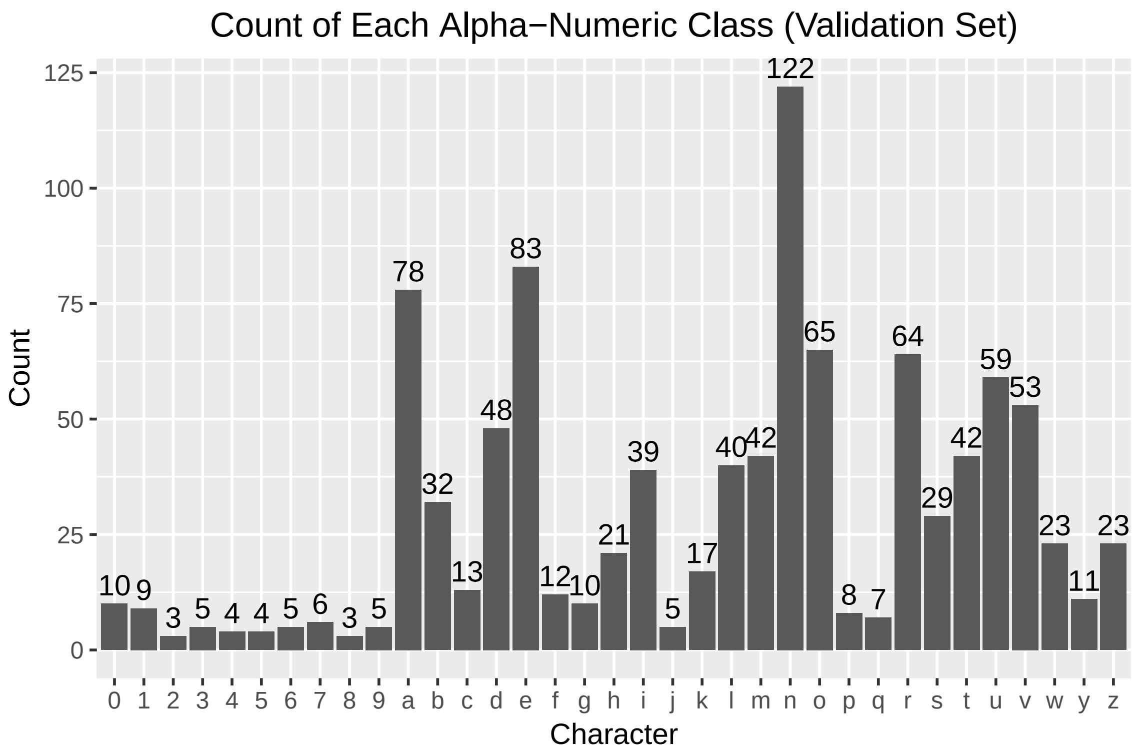


Figure 4: Count of each alpha-numeric class in the validation set.

Figures 3 and **4** show the count of each alpha-numeric class in the training and validation sets, respectively.

3.3 Principal Component Analysis (PCA)

We can use PCA to reduce dimensionality in order to visualize the data in two-dimensional space. In situations like this in which the number of predictor variables is very large, PCA can also be used to reduce dimensionality in order to deal with multicollinearity. As an unsupervised learning technique, PCA can extract patterns from data with only a set of features and no associated response variable.

PCA allows us to summarize the original variables with a smaller number of variables that collectively explain most of the variability in the data [1]. The first principal component is a linear combination of the predictors that has the largest variance. Each succeeding principal component has the largest variance out of all linear combinations with the constraint that it is orthogonal to the preceding principal components. In this way, we can use the principal component scores as the predictors instead of the original predictors/pixels.

```
## PVE by first five principal components: 6.72% 5.42% 3.78% 2.74% 2.48%
```

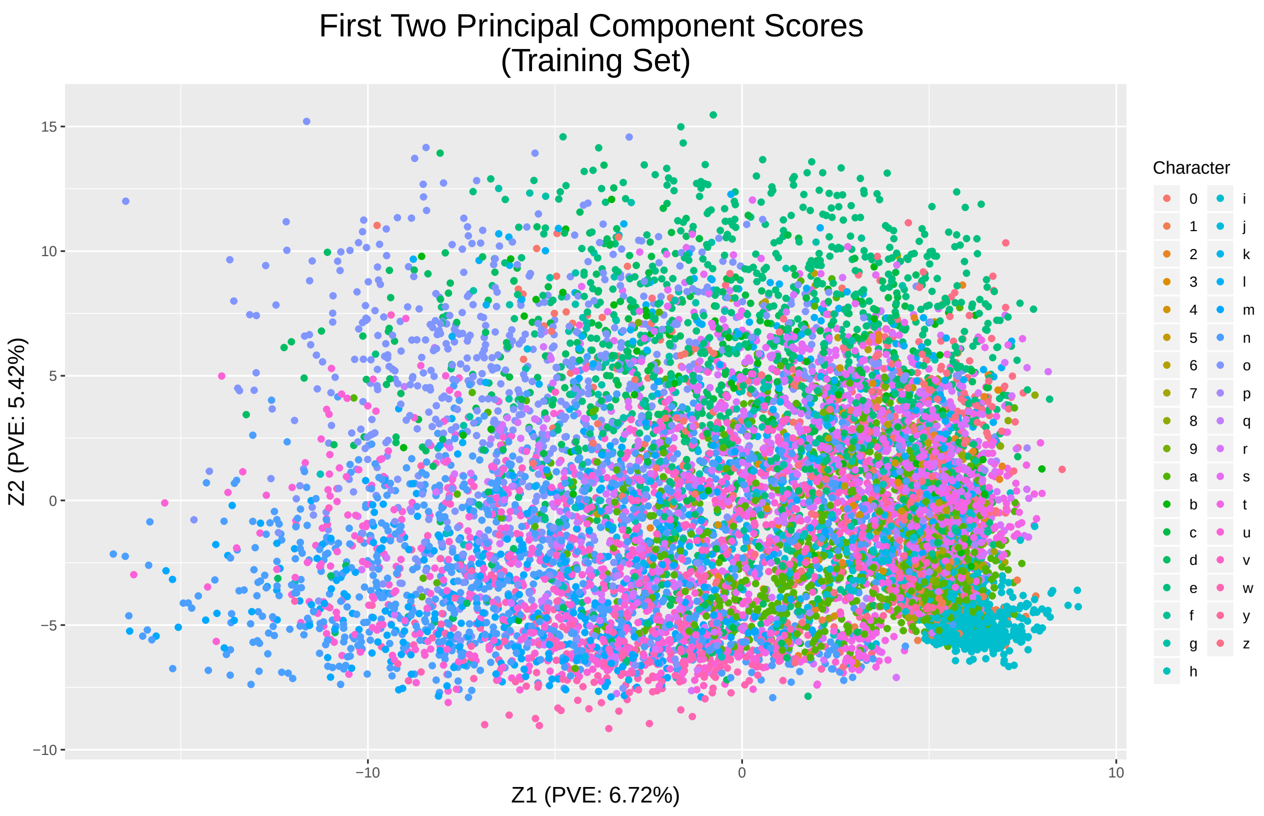


Figure 5: First two principal component scores.

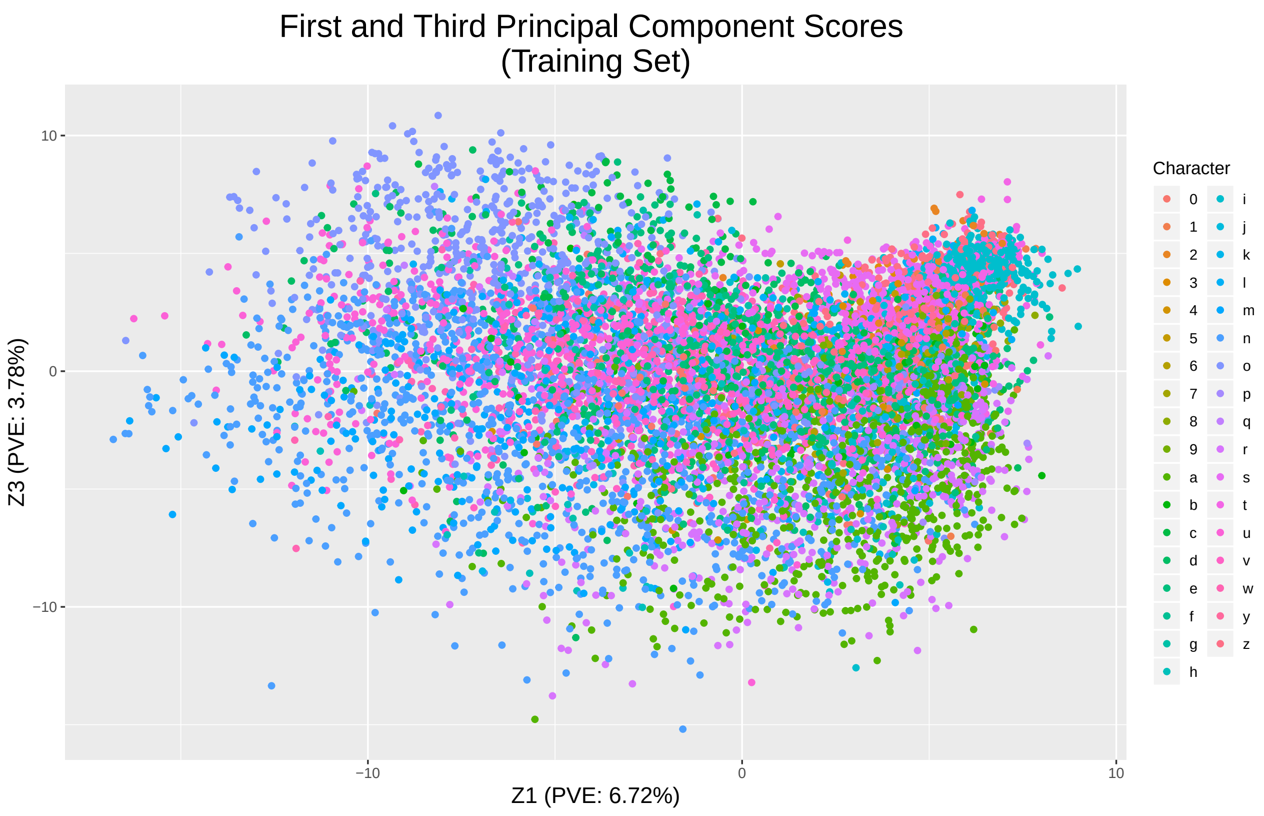


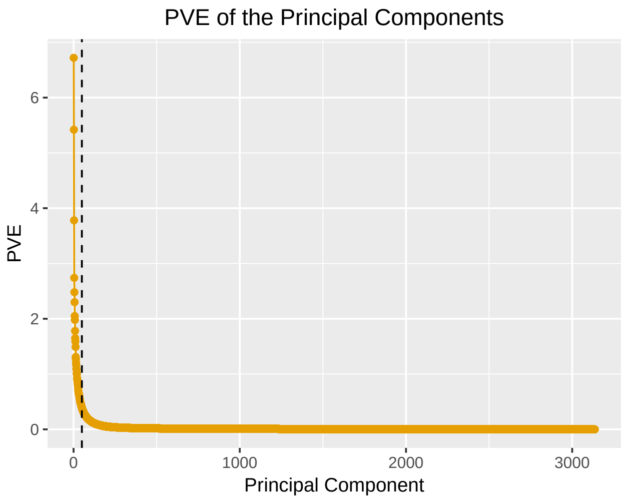
Figure 5: First and third principal component scores.

We see that the first principal component explains only 6.7% of the variance in the data, the second principal component explains only 5.4% of the variance, and so forth.

```
## [1] 3136 3136
```

Figurs 5 and **6** shows the projections of the training observations onto the first three principal components. We see barely any separation of the observations, so the first three principal components are not helpful in this regard.

```
## PVE of the first 50 components: 61.4%
```



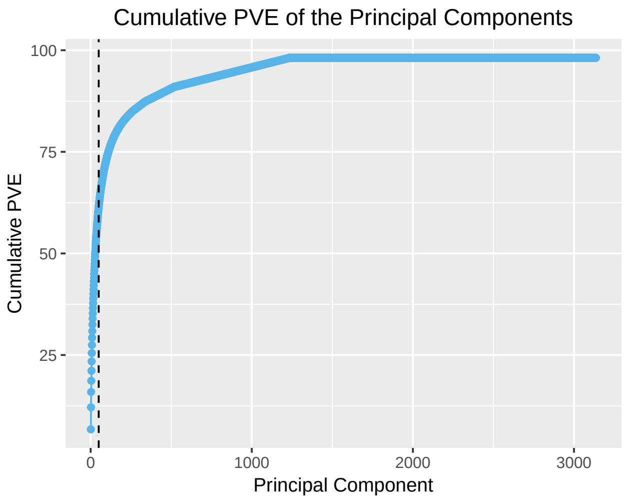


Figure 7: The PVE of the principal components of the training set (left) and Figure 8: The cumulative PVE of the principal components of the training set (right).

Figure 7 shows the proportion of variance explained (PVE) and the cumulative PVE of the principal components of the training set.

The first 50 components explain only about 61.4% of the variance in the data. However, the left-hand plot of **Figure 8** (scree plot) shows there is an "elbow" after approximately 50 components; that is, there is a marked decrease in the PVE by further components. Thus, we can use the first 50 component scores as the new predictors for the training and validation sets of the labeled data and the unlabeled data. Compared to the original 3,316 predictors/pixels, we have greatly reduced the dimensionality.

4. Classification Methods

We will fit various classification models to the training set with *Letter* as the response and the first 50 principal component scores as the predictors. We then test and compare these models by evaluating their performance on the validation set (validation set approach). The models we will consider are the classification tree, random forest, linear discriminant analysis, K-nearest neighbors, and support vector machine.

4.1 Classification Tree (rpart)

The classification tree is an easy to explain and easily interpretable method to predict a qualitative response. Tree-based methods involve dividing the predictor space into a number of regions. Each observation is predicted to belong to the most commonly occurring class of training observations in the region to which it belongs. The classification tree is grown through recursive binary splitting, in which the Gini index is used as the criterion for making the binary splits. The Gini index is a measure of node purity – a small value indicates that a node contains mostly observations from a single class [1].

We use **rpart()** from the **rpart** package to construct a classification tree on the training set to predict the character labels. We draw the tree using **rpart.plot()** from the **rpart.plot** package.

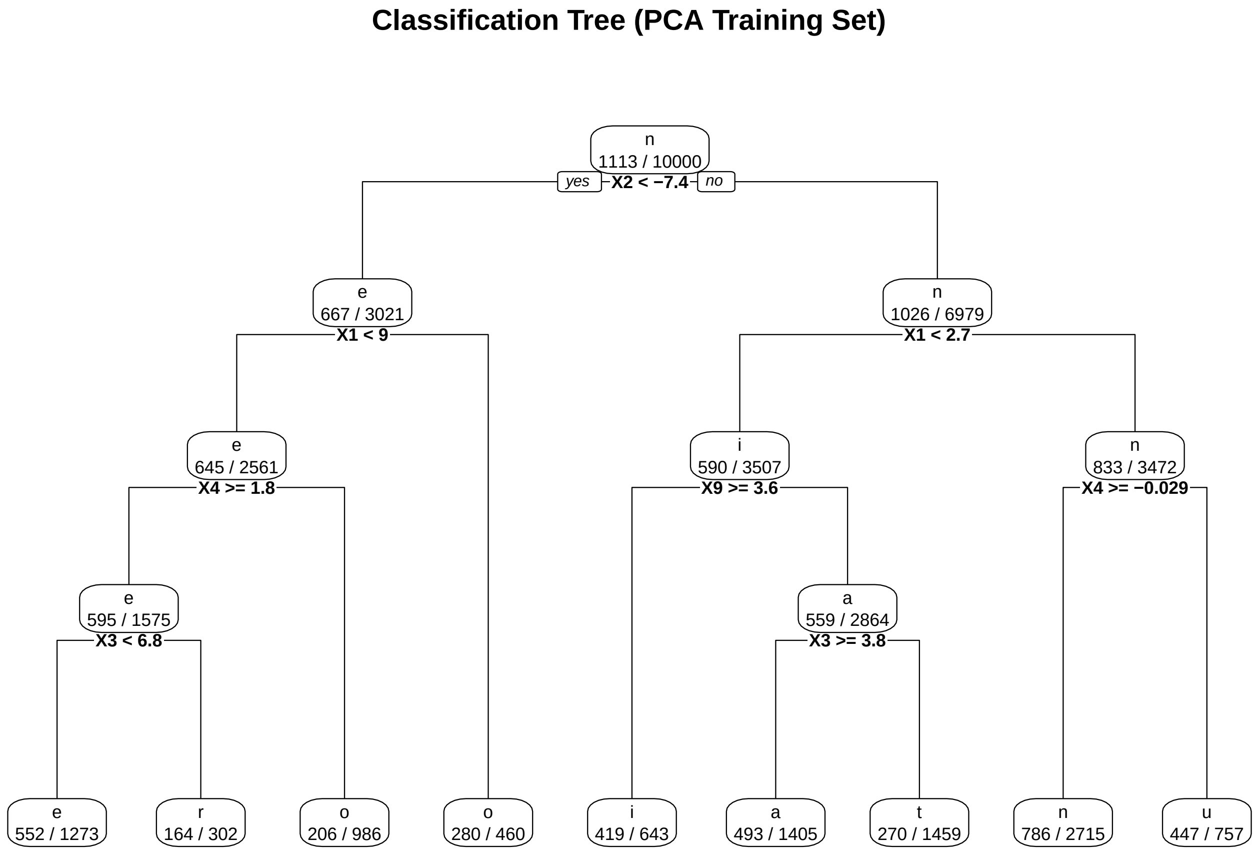


Figure 9: Classification Tree of the training set.

Figure 9 shows the fitted classification tree. We see that there are 15 terminal nodes. The label at the bottom of each internal node indicates the variable being split and the value of the split point. For example, the split at the top of the tree results in two branches. The left-hand branch corresponds to $PC2 < -7.4$, and the right-hand branch corresponds to $PC2 > -7.4$. The letter in each node is the mode or most commonly occurring class for the observations that fall there. The fraction in each node is the number of correct classifications over the number of observations in the node. For example, in the first terminal node, the mode is the letter e, and there are 552 correct classifications out of 1273 observations in that node.

Printing the cp table of the fitted tree produces some summary statistics.

```
##
## Classification tree:
## rpart(formula = train.y ~ ., data = train.data, method = "class")
##
## Variables actually used in tree construction:
## [1] X1 X2 X3 X4 X9
##
## Root node error: 8887/10000 = 0.8887
##
## n= 10000
##
##          CP nsplit rel error  xerror      xstd
## 1 0.065264      0   1.00000 1.00000 0.0035389
## 2 0.044841      1   0.93474 0.93507 0.0042169
## 3 0.043659      3   0.84505 0.86508 0.0047440
## 4 0.029031      4   0.80140 0.80365 0.0050838
## 5 0.022955      5   0.77236 0.77529 0.0052088
## 6 0.017554      6   0.74941 0.75267 0.0052955
## 7 0.013615      7   0.73186 0.73636 0.0053512
## 8 0.010000      8   0.71824 0.71869 0.0054054
```

We see that only 5 out of the 50 principal components were used in tree construction. The lowest cross-validation error occurs at 8 splits or 9 terminal nodes, which means we do not need to prune the tree.

```
## Accuracy rate: 0.657
```

The accuracy of the classification tree is 0.657

4.2 Random Forest

By aggregating multiple trees, using a method like random forest, we can construct more accurate prediction models. Random forest constructs multiple trees and makes predictions based on the mode or most commonly occurring class of the individual trees.

We use **randomForest()** from the **randomForest** package to perform random forest on the training set. In random forest, only a random subset of the predictors is considered for each split; by default, **randomForest()** uses $m \approx \sqrt{p}$ when building a random forest of classification trees. In this case, $m = 7$ since there are 50 predictors/principal components. This is in contrast to bagging, which considers all of the predictors for each split. Therefore, if there are strong predictors in the data, all of the trees will look similar to each other, leading to highly correlated predictions. Using a subset of the predictors decorrelates the trees, which makes the average of the resulting trees less variable and hence more reliable [1].

```
## Accuracy rate: 0.704
```

The accuracy of random forest is 0.704. This is a significant improvement over the single classification tree.

4.3 Linear Discriminant Analysis (LDA)

Linear discriminant analysis models the distributions of the predictors separately for each of the classes, and then it uses Bayes' theorem to estimate the probability of each class [2]. A test observation is assigned to the class with the largest probability. To use LDA, we assume that the observations within each class are drawn from a normal distribution with a class-specific mean vector and a common covariance matrix.

We use **lda()** from the **MASS** package to fit a LDA model to the training set.

```
## Accuracy rate: 0.665
```

The accuracy of LDA is 0.665.

4.4 K-Nearest Neighbors (KNN)

K-nearest neighbors is a non-parametric classifier. In the KNN algorithm, a test observation is assigned to the class most common among its K nearest neighbors (K closest training observations) in the feature space, where K is a positive integer. For example, with K=1, a test observation is simply assigned to the class of the single closest training observation.

We use **knn()** from the **class** package to perform KNN on the training set. A simple rule of thumb is to use $K = \sqrt{n}$, where n is the number of training observations [5]. In this case, $K = \sqrt{613} = 100$. However, we should experiment with different values of K and choose the value that produces the lowest cross-validation error. For a sequence from 2 to 100, we find that the best value of K is 89.

```
## Best value of K: 89
```

```
## Accuracy rate: 0.654
```

The accuracy of KNN (K=89) is 0.654.

4.5 Support Vector Machine (SVM)

A support vector machine is a classifier defined by a separating hyperplane. In two-dimensional space, this hyperplane is a line that separates the classes. Given a labeled training set, a SVM constructs an optimal hyperplane that creates the largest separation, or margin, between the classes. A test observation is then classified based on which side of the hyperplane it lies.

We fit a support vector classifier (linear SVM) to the training set with various values of cost (0.001, 0.01, 0.1, 1, 5, 10, 100). The errors for these models can be accessed using **summary()** (not shown due to computation time).

We find that cost=0.1 results in the lowest error rate at 0.406. The summary of the fitted support vector classifier using cost=0.01 is shown below:

```
##
## Call:
## svm(formula = train.y ~ ., data = train.data, kernel = "linear",
##     cost = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##       gamma:  0.02
##
## Number of Support Vectors:  5330
##
##  ( 137 215 501 89 166 107 91 320 412 118 165 21 233 146 312 299 67 348 285 55 360 115 26 198 38 72 28
##
##
## Number of Classes:  35
##
## Levels:
##  0 1 2 3 4 5 6 7 8 9 a b c d e f g h i j k l m n o p q r s t u v w y z
```

Next, we fit an SVM with radial basis kernels with various values of cost (0.001, 0.01, 0.1, 1, 5, 10, 100) and gamma (0.001, 0.01, 0.1, 1). We use **tune()** to perform 10-fold cross-validation on the set of models under consideration. The cross-validation errors for these models can be accessed using **summary()** (not shown due to computation time).

We find that cost=5 and gamma=0.01 result in the lowest error rate at 0.361. The summary of the fitted radial SVM using cost=5 and gamma=0.01 is shown below:

```
##
## Call:
## svm(formula = train.y ~ ., data = train.data, kernel = "radial",
##     cost = 5, gamma = 0.01)
##
##
## Parameters:
```

```
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  5
##       gamma:  0.01
##
## Number of Support Vectors:  6276
##
##  ( 153 242 635 89 182 113 91 414 467 124 199 21 302 185 412 329 69 466 326 55 473 127 26 221 39 73 2
##
##
## Number of Classes:  35
##
## Levels:
##  0 1 2 3 4 5 6 7 8 9 a b c d e f g h i j k l m n o p q r s t u v w y z
```

Finally, we fit an SVM with polynomial basis kernels with various values of cost (0.001, 0.01, 0.1, 1, 5, 10, 100) and degree (2, 3, 4, 5). The error rates for these models can be accessed using **summary()** (not shown due to computation time).

We find that cost=5 and degree=2 result in the lowest error rate at 0.424. The summary of the fitted radial SVM using cost=5 and degree=2 is shown below:

```
##
## Call:
## svm(formula = train.y ~ ., data = train.data, kernel = "polynomial",
##     cost = 5, degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  5
##      degree:  2
##       gamma:  0.02
##      coef.0:  0
##
## Number of Support Vectors:  7111
##
##  ( 162 240 787 88 204 113 86 592 537 119 227 21 346 207 480 337 64 583 360 54 557 131 25 228 41 74 2
##
##
## Number of Classes:  35
##
## Levels:
##  0 1 2 3 4 5 6 7 8 9 a b c d e f g h i j k l m n o p q r s t u v w y z
```

Among the SVM's, the SVM with radial basis kernels performed the best; hence, we will use that model to compute the validation set test error.

```
## Accuracy rate: 0.77
```

The accuracy of the SVM with radial basis kernels (cost=5, gamma=0.01) is 0.77

4.6 Summary of Classification Methods

The test error rate estimates from the validation set approach (VSA) and 10-fold cross-validation for the various classification methods are compared in **Table 3**.

Table 3: Test Error Rate Comparison for the Classification Methods

Method	VSA
Classification Tree	0.657
Random Forest	0.704
LDA	0.665
KNN (K=89)	0.654
Radial SVM (cost=5, gamma=0.01)	0.77

We should note that the SVM with radial basis kernels performed the best since it has the lowest validation set test error. Thus, we recommend the SVM with radial basis kernels (cost=5, gamma=0.01) as the best classifier for the character recognition data.

The error rates are relatively ok, but we must keep in mind we introduced bias into the model since some of the labels in the training and validation sets are wrong.

6. Conclusion

In this project, we were given a labeled dataset with 1,000 labeled handwritten characters and an unlabeled dataset with 10,000 unlabeled handwritten characters. We performed PCA on the unlabeled dataset that was labeled prior to training to increase the count for each class. This works because PCA is an unsupervised learning technique that allows us to summarize the original variables with a smaller number of variables (principal components) that collectively explain most of the variability in the data. Based on the proportion of variance explained (PVE) or "scree" plot, we chose 50 as the number of principal components to use; thus, we have reduced the dimensionality from 3,316 predictors or pixels to 50 predictors or principal component scores.

With the first 50 principal component scores as the new predictors, we fit the following models on the training set of the newly labeled data from the unlabeled dataset: classification tree, random forest, LDA, KNN, and support vector machines (linear, radial, and polynomial kernels). We then used the validation set approach with the labeled data to estimate the test error rate of each model. It was found that the SVM with radial basis kernels (cost=5, gamma=0.01) performed the best in validation set approach (error rate = 0.23).

Finally, we used our chosen classifier (SVM with radial basis kernels) to predict the labels of the finalunlabeled data (30,000). This is just the first step in constructing an automated handwriting recognition system. For future work, we could focus on improving discernibility between "0" and "O", "1" and "I", "5" and "S", "D" and "0 or O", "Z" and "2", "R" and "K", "G" and "6", "U" and "K", "N" and "W" and "M" and "H". We can also try binning these characters and assign them based on human judgment. Since the characters in the data are heavily unbalanced, we can improve our results by obtaining more examples for the sparse characters such as the digits.

References

[1] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R.* New York, NY: Springer.

[2] Peixeiro, M. (2018, December 11). *Classification – Linear Discriminant Analysis.* Retrieved from https://towardsdatascience.com/classification-part-2-linear-discriminant-analysis-ea60c45b9ee5

[3] By Agor153 - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=24350615

[4] By Agor153 - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=24350617

[5] Thirumuruganathan, S. (2010, May 17). A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm [Blog post]. Retrieved from https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/

[6] User:ZackWeinberg, based on PNG version by User:Cyc [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)]