# On Classification: An Empirical Study of Existing Algorithms Based on Two Kaggle Competitions

Wilson Florero-Salinas (Team Leader), Sha Li (Team Leader)
Xiaoyan Chong, Dan Li, Minglu Ma, Abhirupa Sen, Carson Sprock, Yue Wang
Faculty Advisor: Guangliang Chen

November 14, 2016

**Abstract**

Classification is a machine learning application used to predict group membership for data instances. In this paper we present various classification techniques including nearest-neighbors, decision trees, logistic regression, support vector machines and neural networks. Performance is compared based on two data sets from data science competitions by Kaggle. The first one is handwritten digits recognition and the second is to predict which customers will respond to a direct mail offer. The goal of this paper is to provide an overview of different classification techniques in the literature.

## Contents

# 1 Introduction

## 1.1 Introduction to the Kaggle.com Competitions

Kaggle is an online platform for data science competitions. This platforms lets companies and researchers post their data so that statisticians and data scientists compete to produce the best predictive models. Kaggle's method of operation consists of first having the competition host prepare the data and description of the problem. Next, participants experiment with different machine learning techniques and compete against each other to produce the best predictive model. Competitors are also allowed to communicate with each other and share ideas or code publicly in the platform, and even join the competition as a team. Participants may test their models and submit their results various times during the competition until the deadline. After the competition deadline, models are evaluated with hidden solution files.

## 1.2 The Data

### 1.2.1 MNIST Data

Computer image recognition has been a growing field intersecting many disciplines such as computer science, mathematics, and engineering. The interest in these type of problems has increased over the years because of the potential applications this has lead to, ranging from face detection and recognition, surveillance, national security, bank transactions, and artificial intelligence. In the MNIST Digit Recognition competition, the problem is to identify the digits from images of handwritten digits. More precisely, given the image of a handwritten digit, correctly determine or predict the digit displayed in the image.

The MNIST data set is a subset of a larger data set collected by NIST, the US National Institute of Standards and Technology. It consists of 28x28 images of handwritten digits, ranging from 0 to 9. Table 1 shows a distribution of each of the 10 digits. The images are size-normalized, centered, and partitioned in two sets, 60,000 for training and 10,000 for testing. Figure 1 shows a small sample from the MNIST data set.

| **digit** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 5923 | 6742 | 5958 | 6131 | 5842 | 5421 | 5918 | 6265 | 5851 | 5949 |

Table 1: Distribution of the 10 digits

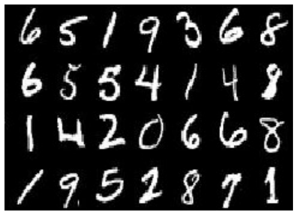


Figure 1: A small subset of the MNIST data set.

The natural variation in handwriting style between people poses several challenges. Due to the orientation in which the digit was written (left handed vs right handed), the digits in the dataset have a natural slant that can influence their classification. The thickness of each digit is another challenge and as shown in Figure 2, some digits are so thick that it loses an important characteristic of that digit, namely the closing of the lower "hole" in the "8".

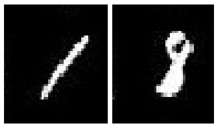


Figure 2: (1) The digit 1 was written with a slant (2) The second digit corresponds to an 8, but could be misclassified as a 1.

The images can be represented as $28 \times 28$ matrices where each entry is the grayscale value of the corresponding pixel. There are some classification methods that work directly with the matrices or we may form 784-dimensional vectors by stacking the columns of the image matrices. Representing the data as vectors this way allows us to use dimension reduction techniques based on linear algebra. Many of the classification methods we use also require the data to be in the form of vectors.

### 1.2.2 Springleaf

Springleaf is a consumer finance company that uses direct mail marketing to reach its customers. As part of its direct marketing efforts, the company uses a database of customers to create models to predict which customers are most likely to use their services and respond to their marketing.

The database is large and unstructured with 1932 independent variables and a binary response variable. There are 145,231 observations representing customers and the binary response variable indicates whether the customer responded to direct marketing. Since the independent variables are unknown, model-building is difficult and we are challenged to construct new meta-variables and employ feature-selection methods to approach this dauntingly wide dataset.

Our task is to construct a prediction model using this data. Unlike the MNIST set, the Springleaf set is not split into training and test sets so the models can be constructed using the entire set or by splitting the dataset into randomly assigned test and training categories. Accuracy is to be evaluated based on the model's ability to correctly predict the response variable.

## 1.3 Classification

We now introduce the classification problem formally. Let $X$ be a set of points $x_k \in \mathbb{R}^m$ and $C$ a set of classes where each $x_k$ is in exactly one class. The number of classes is denoted with lower-case $c$. Together with its class, each point can be represented by the ordered pair $(x_k, i)$, $i \in C$. The set $(T, C)$ is called the *training set*. The objective of classification is to use the training set to find a function $f : \mathbb{R}^m \to C$ such that $f(x) = i$ for as many new points $(x, i)$ as possible. The function $f$ is called a learning machine and a learning machine designed for classification is a classifier. Figure 3 illustrates the classification process.

### 1.3.1 Multiple Classification

Whenever a data set is partitioned into more than two classes, we face the problem of multiple classification. There are classifiers such as nearest neighbor methods that are easily extended to multiple classes, however some methods such as support vector machines are not extended so easily. One way of constructing a multiple classifier is to construct multiple binary classifiers, the outputs of which are combined in some way to produce a single output. Multiple binary classifiers may also out perform their multiple-class counterparts. We use pairwise voting and one-vs-rest decision rules to construct multiple classifiers.

Let $(T, C)$ be a data set with $|C| = c > 2$ classes. We can construct a pairwise multiple classifier

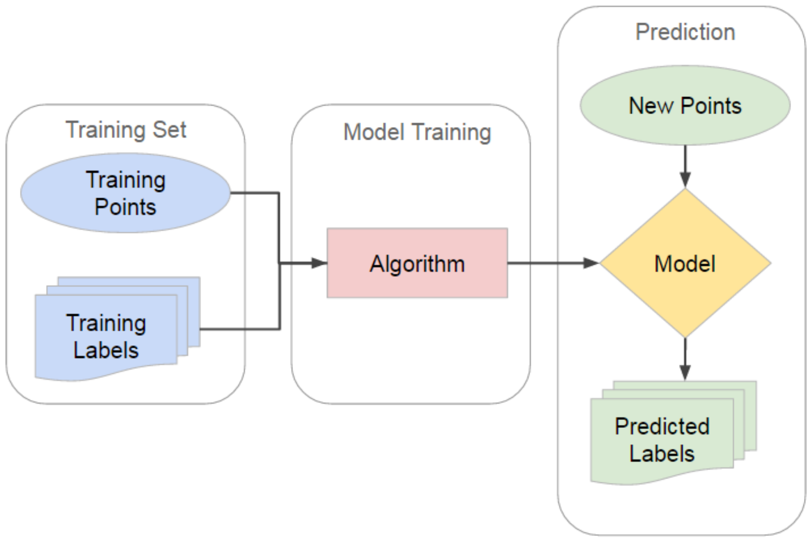


Figure 3: The Classification Procedure

$f : \mathbb{R}^m \to C$ by constructing $\binom{c}{2}$ binary classifiers $f_{ij} : \mathbb{R}^m \to \{i, j\}$, $1 \leq i < j \leq c$ with the decision rule

$$f(x) = \arg\max_{i \in C} \Big\{ \sum_{p=1}^{\binom{c}{2}} w_p I_i(f_p(x)) \Big\} \tag{1}$$

where $p$ ranges over the pairs of classes and $w_p$ are optional weights. $I_i$ is the indicator variable for the $i$th class. When all the weights are equal to one, the above expression is the decision rule for simple plurality voting where $x$ is assigned to the class to which a majority of the pairwise classifiers have assigned it to. In the event of ties, other weighting schemes may have to be used. [24]

The one-versus-rest method works by comparing class $i$ with rest of the classes $\neg i$. $f$ can be constructed from the $c$ learning machines $f_i : \mathbb{R}^m \longrightarrow \{i, \neg i\}$, $i = 1, \ldots c$ where $f_i(x) = \arg\max_{\{i,\neg i\}}\{\alpha(i), \alpha(\neg i)\}$ where the $\alpha$'s are scalars the generated by the decision function used by $f_i$. The multiple classifier can be constructed from the $f_i$ using the rule

$$f(x) = \arg\max_{i \in C}\{g(\alpha(i))\} \tag{2}$$

where $g$ is some weighting function. [24]

# 2 Data Preprocessing and Visualization

## 2.1 Preprocessing the Springleaf Data

Springleaf's dataset comes with 145,231 observation and 1932 anonymous data features. Anonymous data means that any personal identifiable information is removed from each observation and each variable is unlabeled. The binary response variable target is 0 if that customer did not respond to the mail offer and 1 means the offer was responded to.

We have divided the dataset into training and testing sets to construct our models. Two-thirds of the data are used as training set and remaining third as the test set. There is about a 4:1 ratio between the "not responded" and "responded" observations. There are 1876 numerical variables, 51 character variables, and 5 constant variables in the dataset. About 67% variables have 100 unique values or less so although some variables appear numeric they could be categorical.

There are three kinds of missing values in our dataset. "", "NA" are the default missing values for all input methods. "[]" and "-1" are the missing values for character variables and "-99999", "99", "97", "98", etc. are the missing values for numerical variables. About 27% of our data set is missing and about 25.3% variables have missing values. Every observation has three or more variables with missing values and more than 60% of observations have ten or more missing values. This added complexity necessitates the preprocessing of the data, which will be a crucial part of this project.

In order to find a model to best predictive ability we must remove data that do not contribute information useful for classification and to identify the most important variables. We group the missing values "[]" and "-1" with "NA". Some classifiers do not work with character variables so they must be replaced with numeric values. For variables like gender, geological locations, we can replace them with the level number of its values. For some other variables, such as phone number, Social Security Number, we would group them as *existed* = 1, or *non-existed* = 0. Date variables are strings, which do not provide much information to classify the response variable, therefore, we have divided each date into 4 fields – Month, Day, Hour, pair-wise difference.

We have used three different ways to process the missing values in our dataset. The most generally used is to replace missing values by its median value of each variable. One problem this could cause is to introduce a dominant class when it really shouldn't. To prevent this, we also tried to fill missing values by a multinomial random variable with each value's distribution probability. For some methods, we also tried to regard missing values as a new group.

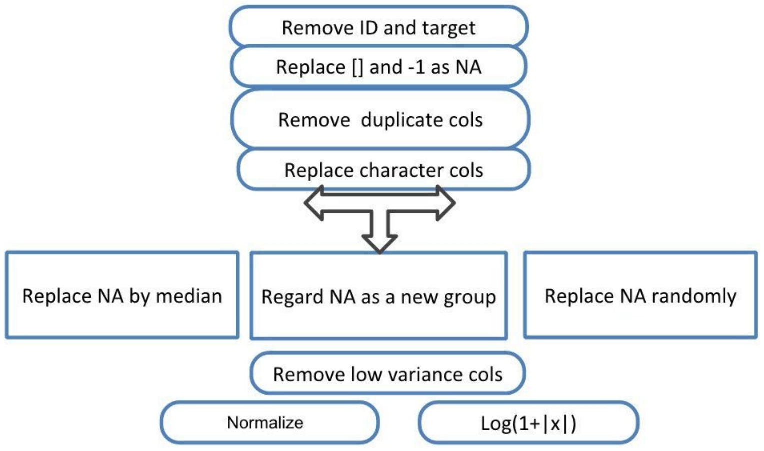


Figure 4: *Data Preprocessing Flow Chart*

The following, also shown in Figure 4, is the process we have performed on our data for most of the classification methods:

- Read in data, "" and "NA" string are treated as missing value by default.
- Replace "[]" and "-1" as missing values.
- Remove variable "ID" and "target".
- Remove duplicate columns.
- Replace character variables by its level number.
- Process missing values – replace missing values by the median of existing values.
- Remove columns with very low variance.
- Normalize the cleaned-up data, or take the log of $(1+|x|)$ for different classification methods.

## 2.2 Visualizing MNIST with t-SNE

t-distributed Stochastic Neighbor Embedding, or t-SNE, is a method of visualizing high-dimensional data by giving each data point a location in two or three-dimensional space. Even though t-SNE can be used as a dimension reduction technique it is mostly used for visualization. In our project we used t-SNE as both a dimension reduction (see next section) technique and to visualize the ten classes in the MNIST set.

The method is divided into two main parts: The first part of the algorithm consists of constructing a probability distribution between the high-dimensional data points in such a way that similar points have a high probability of being chosen for the embedding, while dissimilar points have a small probability of being picked. To be more precise, let $x_i$ and $x_j$ be two high-dimensional points, and define the conditional probability

$$p_{j|i} := \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

where $p_{ij}$ is set as $p_{ij} = \dfrac{p_{j|i} + p_{i|j}}{2}$. Here $p_{j|i}$ is a conditional probability that $x_i$ would pick $x_j$ as its neighbor if neighbors were picked in proportion to their probability density under the Gaussian centered at $x_i$. For similar points $p_{j|i}$ will be relatively high while for dissimilar points (far apart) it will be almost infinitesimal. The bandwidth of the kernel, $\sigma_i$, is adapted to the density of the data, where smaller values of $\sigma_i$ are used in denser regions of the dataset.

In the second part, a similar process is done for the lower dimensional points (which are to be constructed) in which the KullbackLeibler divergence is minimized between the two distributions with respect to the locations of the points in the map. More precisely, for the lower-dimensional counterparts $y_i$ and $y_j$ of $x_i$ and $x_j$, respectively, a conditional probability $q_{j|i}$ is constructed. In t-SNE a Student t-distribution with one degree of freedom is employed as the heavy-tailed distribution in the low-dimensional map so that the probability $q_{j|i}$ is given by

$$q_{j|i} := \frac{(1+||y_i - y_j||^2)^{-1}}{\sum_{k\neq l}(1+||y_k - y_l||^2)^{-1}}$$

Finally t-SNE uses gradient descent to minimizes the Kullback-Leibler divergence between the joint probabilities $p_{ij}$ in the high-dimensional space and the joint probabilities $q_{ij}$ in the low-dimensional space. The Kullback-Leibler divergence between the two joint probability distributions $P$ and $Q$ is given by

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

.

In our experiments we used the MATLAB and R implementations of t-SNE. Figure 5 shows output of the MNIST data set using the R version of t-SNE. Both the 2D and 3D visualizations of the data show the differences in the ten different classes, but also show how some digits were

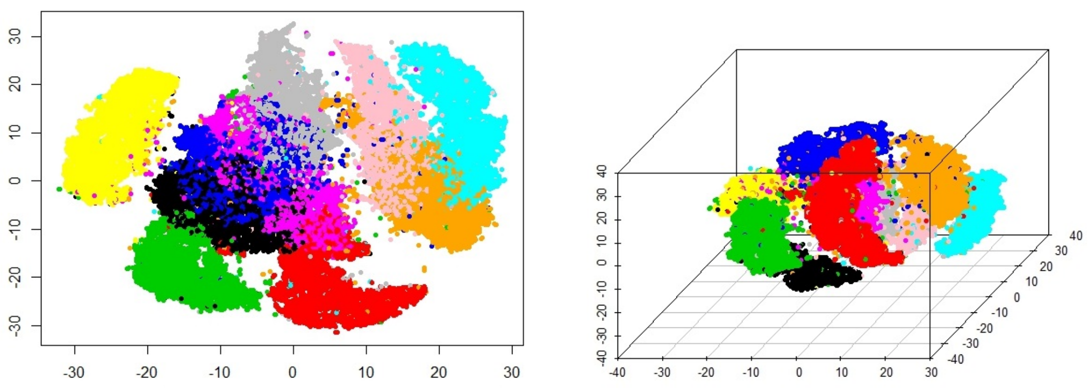


Figure 5: a 2D and 3D visualization of the MNIST data set, respectively

inevitably placed in other clusters. These misclassified digits correspond to distorted digits that in some cases are even difficult for humans to correctly identify.

In an attempt to better separate all ten clusters for visualization, we also used more sophisticated variations of t-SNE, which include parametric t-SNE [26], and kernel t-SNE [7], but these methods only gave small improvements in plots that were not significantly better than t-SNE. In addition to this, t-SNE is a tool for visualization rather than dimensionality-reduction. We also tried combining t-SNE with other classifier methods, but our experiments did not provide good results.

## 2.3 Dimension Reduction

Dimension reduction methods map high dimensional data into a lower dimensional space while retaining the most important variables under some criteria. Dimension reduction helps mitigate computational costs and is particularly useful in data visualization. It has also been widely adopted to improve performance of various machine learning models [2, 20]. The MNIST data set in our case has 784 dimensions, which is difficult to deal with.

Formally let $X \in \mathbb{R}^{m \times n}$ be a data set where the column vectors $x_k \in \mathbb{R}^m$ $k = 1, \ldots, n$ represent observations in some $m$-dimensional space. The goal of dimension reduction is to find a lower dimensional representation of the data that preserves or enhances some of its features. For this reason, dimension reduction is often referred to as feature extraction and involves finding a linear transformation $W \in \mathbb{R}^{d \times m}$ $d < m$ such that $Y = WX$ is a representation of the original data that optimizes some property. A learning machine then can be trained on the set $(Y, C)$ and classification of a new point $x$ accomplished by classifying its image $y = Wx$.

Some dimension reduction methods such as discriminant analysis use the class information in the training set. Such methods are "supervised" in the sense that they require class information in the training set to be observed by the analyst. "Unsupervised" methods such as principle component analysis do not require class information and can be applied regardless of prior knowledge about class membership.

Mechanically,

$$WX = \begin{bmatrix} - \; w_1^T - \\ \vdots \\ - \; w_m^T - \end{bmatrix} X = \begin{bmatrix} w_1^T X \\ \vdots \\ w_m^T X \end{bmatrix}$$

The vectors $w_i^T X$ are called *features* and are the projection of the data $X$ onto the $i$th component of the transformation $W$. [13]

### 2.3.1 Principal Component Analysis

Dimension reduction is a process which maps data to a lower dimensional space while retaining the most important variables. Dimension reduction helps mitigate computational costs and is particularly useful in data visualization. It has also been widely adopted to improve performance of various machine learning models [2, 20]. The MNIST data set in our case has 784 dimensions, which is difficult to deal with. Preprocessing with the data using appropriate dimension reduction methods appears to be helpful. One important dimension reduction method we have studied is Principal Component Analysis (PCA), which has successful application in pattern extraction from high dimensional data [10].

In this section, we will give a short introduction to PCA. The basic idea of PCA is projecting the data to a low dimensional space by maximizing a certain objective function, which is the variance in the case of PCA [11].

Given $n$ points $x_i \in R^m$, we would like to find a linear transformation $W \in R^{d\times m}$ $(d < m)$ that maps $x$ to $y = Wx$. Without loss of generality, we assume $\sum x_i = 0$. If this is not true, we can always centralize the data by $\tilde{A} = \sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T$ where $\mu = \frac{1}{n}\sum x_i$. $y$ is the low dimensional representation of the original data $x$ and we have the recovered data $\tilde{x} = W^T y$ [27]. The goal of PCA is to find the mapping $W$ that minimize the total squared distance between the original data $(x)$ and the recovered data $\tilde{x}$. This can be done by solving a quadratic problem

$$\min_{W\in R^{d,m}} \sum_{i=1}^{n} ||x_i - W^T W x_i||^2. \tag{3}$$

Since $W$ is orthonormal, we have

$$\begin{aligned} ||x - W^T W x||^2 &= ||x||^2 - x^T W^T W x \\ &= ||x||^2 - \mathrm{Tr}(W x X^T W^T), \end{aligned} \tag{4}$$

where $WW^T = I$ is used. Therefore, the optimization problem reduces to

$$\max_{W\in R^{d,m}} \mathrm{Tr}(W \sum_{i=1}^{n} x_i x_i^T W^T). \tag{5}$$

The matrix $A = \sum_{i=1}^{n} x_i x_i^T$ is symmetric and can be decomposed as $A = VDV^T$, where $D$ is diagonal and $V^T V = VV^T = I$. The diagonal elements of $D$ are the eigenvalues of $A$ and the corresponding eigenvectors are stored in $V$. Let $v_1, v_2, \ldots, v_k$ be $k$ eigenvectors of $A$ associated with the largest $k$ eigenvalues. The solution of PCA is thus $W = [v_1, v_2, \ldots, v_k]$. In this study, we use singular value decomposition (SVD) to perform PCA.

As an example of data visualization, Figure 6 plots digit 0 to 9 from MNIST training set respectively featuring the top two PCA components.

### 2.3.2 Classwise PCA

For digit recognition, conventionally, PCA is performed on the whole training data set and the test data is projected to the top $k$ principal components [28]. This PCA method can be also called global PCA. In this study, we present an alternative approach which we call local PCA. In the local PCA, PCA is performed on the individual groups of the training data and the training data is projected to the subspace according to its group. Intuitively, this approach extracts distinguished features from each group which helps separate the groups better. As a result, it will improve the performance of the classifiers (KNN, SVM, etc.). Specifically, in MNIST data, we obtain 10 different subspaces (digit $0, 1, \ldots, 9$ respectively), each group can have different number of PCA components. If we use SVM as classifier, we would train 10 models based on the 10 subspaces (one-vs-rest method). In the classification stage, the test data is projected to the 10 different subspaces before being sent to the 10 trained SVM models [12]. Our method shows improved results compared to conventional method. Detailed results can be found in 6.4.2.

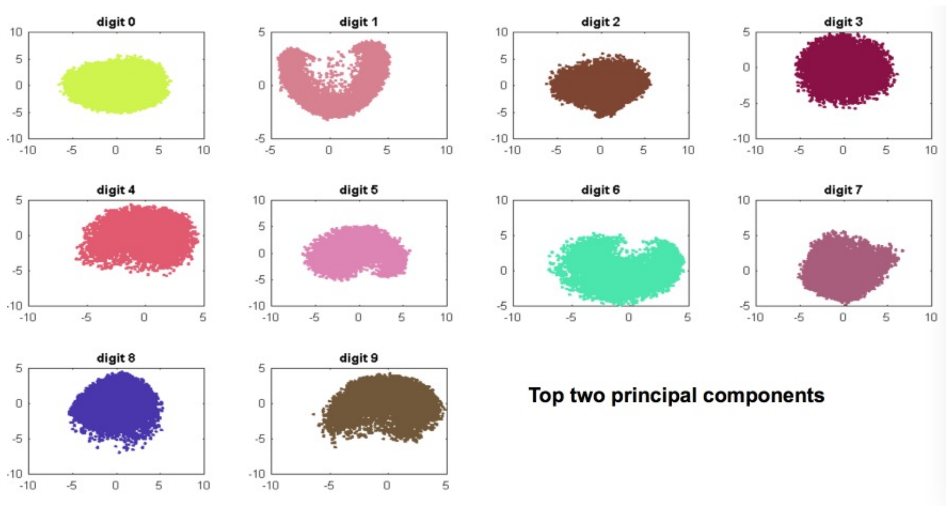


Figure 6: The new coordinates of digit 0, 1, 2, 3, 4, 5, 6, 7, 8, and digit 9 corresponding to the top 2 principle directions

## 3 Instance-based Classifiers

### 3.1 Methods description

In this section we will be talking about the instance-based classifiers we used to classify the digits. Instance-based classifiers are lazy learners. These algorithms, instead of a generalized classification, compare each new instance of data with the instances seen during training and hypothesize accordingly. The instance-based methods we have tried are $k$-nearest neighbor method, its variant weighted k nearest neighbor, $k$-means and its variant local $k$-means.

### 3.2 Nearest Neighbors Methods

The nearest neighbor (NN) method works by taking an unclassified point $x$ and assigning it to the class of the nearest point in the training set according to some distance metric $d$. Here we take $d(x,y) = \sqrt{(x-y)^T(x-y)}$ to be the Euclidean distance between the points $x$ and $y$.

The $k$-nearest neighbor (kNN) method extends this idea by using the classes of local points to classify $x$. Let $N_k(x) = \{x_1, \ldots, x_k\}$ be the set of $k$ closest points and let $d_j = d(x, x_j)$ be the distance from $x_j$ to $x$, indexed such that $d_j < d_{j+1}$. The kNN rule assigns $x$ to the class to which the majority of the points in $N_k(x)$ belong. If we want to give more importance to points that are closer to $x$, we may assign weights to the points in $N_k(x)$ based on their distance from $x$. The decision rule for weighted $k$-nearest neighbors (WkNN) can be expressed as

$$f(x) = \arg\max_{i \in C}\{\sum_{j=1}^{k} \delta_i(x_j) w_j\} \tag{6}$$

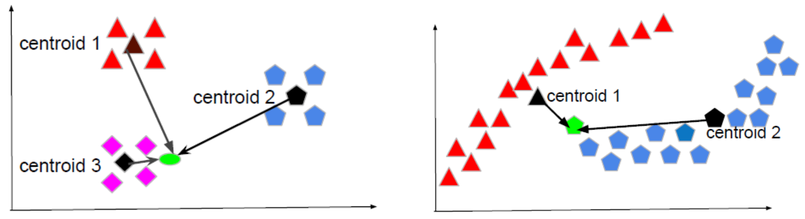
where $\delta_i = 1$ if $x_j \in i$ and 0 otherwise. $x$ is assigned to the class that maximizes this sum. When $w_i = 1$ for all $i$, the above reduces to kNN [8]. We test several different weighting schemes, which are shown in the following table.

| | |
|---|---|
| $w_j = \frac{d_k - d_j}{d_k - d_1}$ | proportional weights [8] |
| $w_j'^{Dual} = w_j \times \left(\frac{d_k + d_1}{d_k + d_j}\right)$ | Dual weighted kNN (DWkNN) [8] |
| $w_j'' = \frac{1}{\sqrt{2\pi}} e^{-\frac{D_j^2}{2}}$ | Gaussian weighted [9] |

In the Gaussian weights $w''_j$, $D_j = \frac{d_j}{d_k}$ is the distance from $x_j$ to $x$ scaled by the largest distance in the set of neighbors.

## 3.3 Local $k$-means

The $k$-means rule is equivalent to the NN rule applied to the set of class means $\{\mu_i\}_{i\in C}$, that is new points are assigned to the class of the nearest class center. $k$-means generally gives good results when the classes are well separated as in Figure 7A, where the new point (shown in green) is assigned to the pink class. However, many data sets have classes that are not well-separated or are non-convex. In non-convex sets the class mean $\mu_i$ may not be a good measure of the center. Figure 7B shows how using the class center may result in misclassification under the $k$-means rule. In this example, the new point belongs to the blue class but the red class center is closer and $k$-means would incorrectly assign it to the red class.



(a) Well-separated classes (b) Non-convex classes

Figure 7: $k$-means

The above discussion suggests that in order to apply a $k$-means method to a non-convex data set we must take into account the local structure of the data. The local $k$-means rule does this by using the centers of sets of points near the test point.

Formally, let $N_k(x, i)$ be the set of $k$ nearest points to $x$ belonging to the $i$th class and let $\mu_{k,i}$ be its center. Under the local $k$-means rule, $x$ is assigned to the class of the closest $\mu_{k,i}$. This is equivalent to the NN rule applied to a set $\{\mu_{k,i}\}_{i\in C}$. That is, a new observation $x$ will be assigned to a class according to the rule

$$f(x) = \underset{i\in C}{\operatorname{argmin}} ||x - \mu_{k,i}|| \tag{7}$$

In less technical terms, to apply local $k$-means one shall: for each class (1) find the $k$NNs of the new observation (2) compute the (local) centers of each set of $k$NN, (3) compute the distance between the new observation and each of the local centers, and finally (4) assign the new observation to the class of the $k$NN which gives the closest local center. Figure 8 shows how local $k$-means correctly assigns the new point.

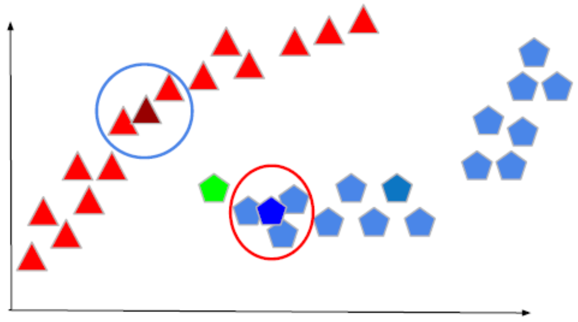


Figure 8: Local $k$-means with non-convex classes

### 3.3.1 MNIST Results

We applied kNN, k-means and the three versions of WkNN over a range of $k$. Classification accuracy decreased as $k$ grew large ($> 20$). Among instanced based method, local $k$-means performed the best. When applied to the MNIST data set and without any preprocessing, local $k$-means gave an error of 1.75% with $k = 14$. By deskewing the data error was reduced to 1.17%, but with $k = 10$. Summary results using instance based methods in combination with different dimensionality reduction methods are shown in Table 2

| **Method** | **Error** | $k$ |
|---|---|---|
| local k-means (deskewed) | 1.17% | 10 |
| local k-means | 1.75% | 14 |
| kNN | 2.95% | 3 |
| WkNN | 2.76% | 8 |
| DWKNN | 3.09% | 1 |
| Gaussian WkNN | 2.83% | 3 |
| tsne + local kmeans | 2.76% | 10 |

Table 2: Summary of Instance based methods on the MNIST data set.

### 3.3.2 Springleaf Results

We have used $k$NN on the Springleaf data. From the Figure 9, we can see that for every $target = 1$ (in Red) observation, it is surrounded by $target = 0$ (in Black) observations, which leads us to believe that the accuracy on the kNN method will not be very high.

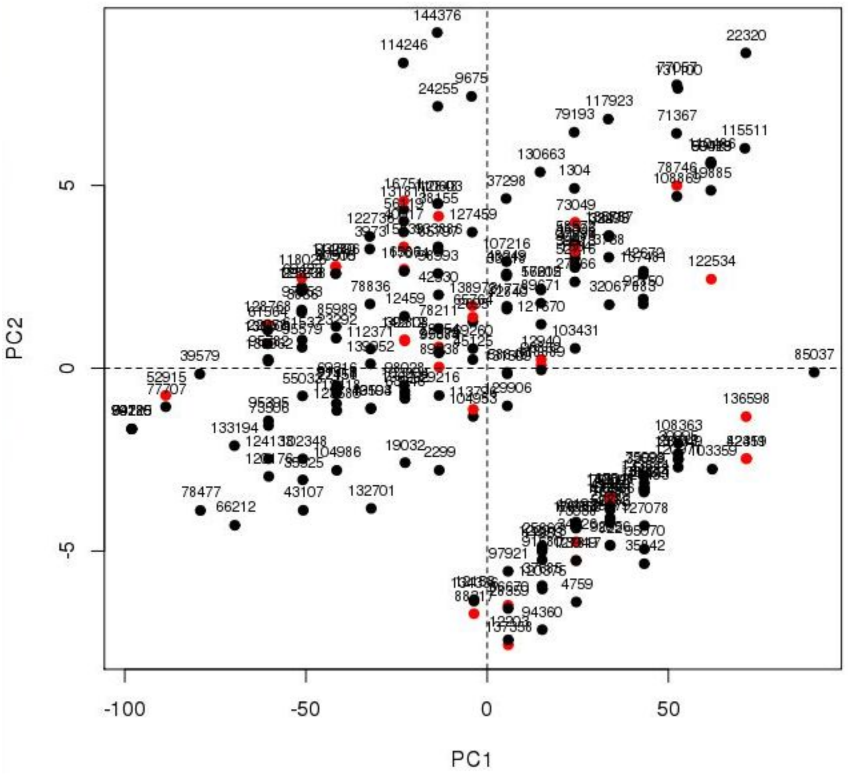


Figure 9: *kNN Neighbors on 2 Dominant PCs*

The following figure shows the result from kNN algorithm for each k we picked. From the figure, we can see the higher the k, the higher the overall accuracy, but the less Target =1 can be identified.

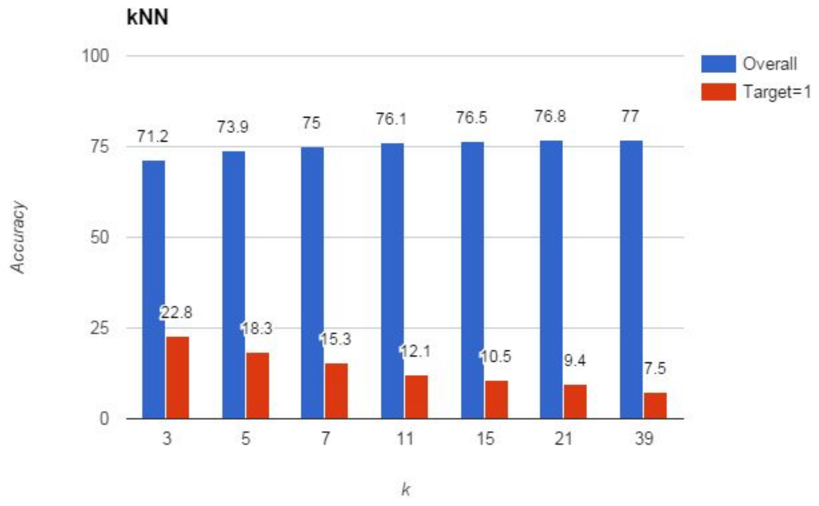


Figure 10: *kNN Classification Results*

The weighted kNN algorithm runs a lot slower than some implementation of kNN, it takes hours to complete the prediction. However, the result is not much better. For k=7, the accuracy of the prediction increased by 1.7%.

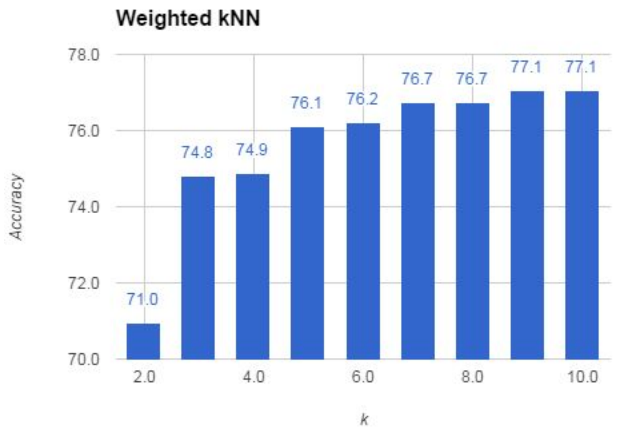


Figure 11: *Weighted kNN Classification Results*

## 4 Discriminative Methods

### 4.1 Linear Discriminant Analysis

For the purposes of classification, it is desirable to have data where the class members are clustered closely together but the different class clusters are relatively far apart. LDA makes use of two quantities that together capture the degree to which this property is present in the data set. Let $n_i$ be the size of the $i$th class and $n$ the size of the whole dataset, where $n = n_1 + \cdots + n_c$. The class mean $\mu_i$ and grand mean $\mu$ are defined respectively:

$$\mu_i = \frac{1}{n_i}\sum_{j=1}^{n_i} x_j$$

$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j$$

*Within-class scatter* captures the spread of the points in the $i$th class about the class mean $\mu_i$ for each class while *between-class scatter* captures the spread of the class means about the grand mean $\mu$. The ratio between between- and within-class scatter measures the separation property. LDA seeks to find a transformation that maximizes this ratio in the lower dimensional space. Define the within-class scatter matrix

$$S_w = \sum_{i=1}^{c}\sum_{j=1}^{n_i}(x_j - \mu_i)(x_j - \mu_i)^T \tag{8}$$

and the between-class scatter matrix

$$S_b = \sum_{i=1}^{c} n_i(\mu_i - \mu)(\mu_i - \mu)^T \tag{9}$$

We seek a transformation $W$ that will maximize the ratio

$$J(W) = \frac{|W^T S_b W|}{|W^T S_w W|} \tag{10}$$

The solution to this problem is given by the solution to the eigenvalue problem $S_b W = \lambda S_w W$, that is, $W$ consists of the eigenvectors of the matrix $S_w^{-1} S_b$ that correspond to the non-zero eigenvalues.[6]

Since $S_b$ is the sum of rank-one matrices, $S_b$ has rank of at most $c-1$. There are at most $c-1$ non-zero eigenvalues of $S_w^{-1} S_b$ which limits the dimension of the transformed data to at most $c-1$. [6][16][15] However, we try to make use of the zero eigenvalues to try and extend the number of features. See Results at the end of the section.

In practical applications with high dimensional data sets the matrix $S_w$ may be singular, as is the case with our data. One way of overcoming the singularity issue is to apply a dimension reduction method such as PCA before using LDA. Another approach called regularization substitutes the matrix $S_w$ with

$$S'_w = S_w + \beta I$$

where $\beta > 0$ is a "bump" term that makes $S'_w$ nonsingular for sufficiently large $\beta$. [15]

### 4.2 Nonparametric Discriminant Analysis

Nonparametric discriminant analysis (NDA) uses a formulation of the within class scatter matrix that utilizes the local structure of the data. Let $x$ be a point and $\mu_{k,i}(x)$ is the local center of the closest $k$ points to $x$ from class $i$. Define $d_{k,i}(x) = d(x, \mu_{k,i}(x))$. The NDA between class scatter matrix, introduced by [16] is defined by

$$S_b = \sum_{i \in C} \sum_{j=1}^{n_i} \sum_{\ell \neq i} w_{j\ell i}(x_j - \mu_{k,\ell}(x_j))(x_j - \mu_{k,\ell}(x_j))^T \tag{11}$$

The second sum is taken over all points in the $i$th class. For each point in class $i$, the third sum is the sum of squared distances between the point and the local $k$ centers of the rest of the classes $\ell \neq i$. This matrix captures the scatter between the points in a class with the rest of the classes. The weight term is given by

$$w_{j\ell i} = \frac{\min\{d_{\ell,k}(x_j), d_{i,k}(x_j)\}}{d_{\ell,k}(x_j) + d_{i,k}(x_j)} \tag{12}$$

which is the ratio of the smallest of the two distances to their sum.[16]

NDA has several advantages over LDA. The matrix $S_b$ is usually nonsingular. Another is that $S_w^{-1}S_b$ has more than $c-1$ nonzero eigenvalues, so more features can be extracted using NDA.[16][6]

## 4.3 LDA/QR

LDA is an alternative to PCA for dimensionality reduction that was considered for the MNIST data set, which results in an eigen decomposition problem. As discussed previously, applying LDA to the MNIST data set is not feasible because of the singularity of the scatter matrices. Another alternate version to classical LDA is LDA/QR [30], which is described in two main steps. In the first step, LDA/QR maximizes the separation between different classes by applying a QR decomposition to a small sized matrix. In the second step, LDA/QR incorporates both between-class and within-class information by applying LDA to a "reduced" version of the scatter matrix obtained from the first step [29]. More formally, let $X \in \mathbb{R}^{m \times n}$ be the data matrix, where each column consists of a $m-$dimensional observation, and let $n_i$ be the size of each class, $1 \leq i \leq c$, where $c$ is the number of classes. Define *between-class, within-class*, and *total scatter* matrices $S_b, S_w$, and $S_t$, respectively so that $S_b = H_b H_b^T$, $S_w = H_w H_w$, and $S_t = H_t H_t^T$, where [29]:

$$\begin{aligned} H_b &= [\sqrt{n_1}(\mu_1 - \mu), \cdots, \sqrt{n_c}(\mu_c - \mu)] \in \mathbb{R}^{d \times c} \\ H_w &= X - [\mu_1 e_1^T, \cdots, \mu_c e_c^T] \in \mathbb{R}^{d \times n}, \text{ and} \\ H_t &= X - \mu e^T \in \mathbb{R}^{d \times n}. \end{aligned} \tag{13}$$

Here $e_i = [1, \cdots, 1]^T \in \mathbb{R}^{n_i \times 1}$, $e = [1, \cdots, 1]^T \in \mathbb{R}^{n \times 1}$, $\mu_i$ is the mean of the $i$th class, and $\mu$ is the global mean. Then we have that $S_t = S_b + S_w$. In the first stage of LDA/QR the algorithm aims to solve the optimization

$$G = \underset{G^T G = I}{\operatorname{argmin}}\, tr\left(G^T S_b G\right)$$

which can be solved by either solving the eigenvalue problem on $S_b$ or by finding the $QR$ decomposition of the centroid matrix $C = [\mu_1, \mu_2, \cdots, \mu_c]$ [31]. In the latter case, we decompose $C$ as $C = QR$, where $Q \in \mathbb{R}^{n \times c}$ with orthonormal columns and $R \in \mathbb{R}^{c \times c}$ is an upper triangular matrix. Letting $G = QV$ solves the optimization problem for any orthogonal matrix $V$. The Algorithm can be summarized as follows:

**LDA/QR**

1. Construct the centroid matrix $C$.
2. Obtain the QR decomposition of $C$ as $C = QR$, where $Q \in \mathbb{R}^{n \times c}$ and $R \in \mathbb{R}^{c \times c}$.
3. Let $Y := H_b^T Q$ and $Z := H_t^T Q$.

4. From step 3, construct $B := Y^T Y$ and $T := Z^T Z$, the "reduced between-class scatter matrix" and "reduced total scatter matrix", respectively.
5. Compute the $c$ eigenvectors $v_i$ of $(T + \alpha I_c)^{-1} B$ with decreasing eigenvalues.
6. Store the reduced vectors in $G$, where $G := QV$ and $V = [v_1, \cdots, v_c]$.

The main advantages of LDA/QR over LDA is that (1) it has much lower cost in time and space, (2) bypasses the singularity of the scatter matrix, and (3) produces $c$ rather than $c-1$ dimensions after dimensionality reduction [30].

#### 4.3.1 MNIST Results

Although LDA and LDA/QR seemed like promising dimensionality reduction methods for the MNIST data set, we found that the dimensionality reduction was too aggressive by compressing the current space to only 10 dimensional space. LDA works well with linearly separable (classes that are separable by a line or plane) data sets and our work here suggests that MNIST does not fall in this category. In later sections we discuss the use of dimensionality reduction methods in conjunction with classifications methods and show LDA/QR results.

### 4.4 2D-LDA: Two Dimensional LDA

When working with images, the choice must be made to represent the images as vectors or to work directly with the image matrices themselves. 2D-LDA is a version of LDA which uses the images matrices directly. All other methods we examined use vector representation.

Let $A_k$ be the $m \times d$ matrix representing the $k$th image in the data set, where $m = d = 28$. The class center for the $i$th class is $\bar{A}_i = \frac{1}{n_i} \sum_{k \in i} A_k$ where the mean is taken element-wise over the set of image matrices. Similarly let $\bar{A}$ be the grand mean of all images in the dataset. $\bar{A} = \frac{1}{n} \sum_{i=1}^{c} \sum_{k \in i} A_k$ The between-class scatter is defined to be the matrix

$$S_b = \sum_{i=1}^{c} n_i (\bar{A}_i - \bar{A})^T (\bar{A}_i - \bar{A}) \tag{14}$$

and the within-class scatter

$$S_w = \sum_{i=1}^{c} \sum_{j=1}^{n_i} (A_j - \bar{A}_i)^T (A_j - \bar{A}_i) \tag{15}$$

Classification is done with nearest-neighbors after the transformation is applied to the data. kNN is implemented using matrix norms in place of Euclidean distance. The $p$-norm for an $m \times m$ matrix $A$ is

$$\|A\|_p = \left( \sum_{i,j=1}^{m} |a_{ij}|^p \right)^{1/p}$$

We use the L1-norm and L2-norm and the Fisher vector norm, which is the sum of squared Euclidean distances of the column vectors of two matrices. [14]. Figure 12 shows the results of kNN classification after the application of 2D-LDA. The best results were achieved using the L2 norm.

### 4.5 Springleaf Visualization with LDA

By applying LDA, we project all the points on to one dimension space. After projecting, the spread of points in each group is shown in Figure 13.

The two groups in one dimension both approximately follow normal distributions. Unfortunately, the projected means of each group are relatively close to each other comparing with their spreads. We can expect the two groups to significantly overlap in one dimension space. Generally, similar to PCA, LDA can not distinguish the two groups in a low dimension space. It implies

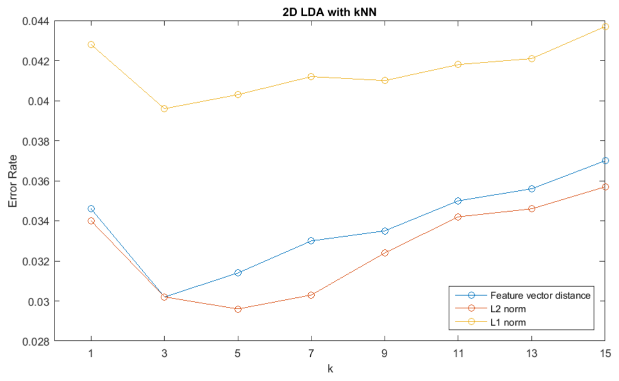


Figure 12: Results of 2D-LDA classification with k nearest neighbors

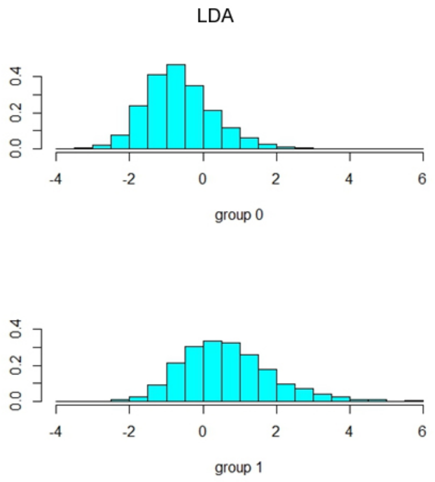


Figure 13: LDA

that our data is not linear separable and that we need to seek other nonlinear methods to do the classification.

# 5 Statistical Methods

## 5.1 Naive Bayes

The Naive Bayes classifier is based on Bayes' theorem with the assumption of independence between every pair of predictors. In a simple term, a Naive Bayes classifier assumes that the features in a dataset are mutually independent. For example, an animal that looks like a horse (feature 1), and with black and white stripes (feature 2) may be considered as a zebra. Even if these features depend on each other and/or depend on the existence of the other features, all these attributes independently contribute to the probability that this animal is a zebra. This is why it is known as "Naive" [18].

In order to understand Naive Bayes Classifiers [17], we need to understand conditional probability and Bayes Theorem first. Conditional probability is a probability that something will happen, given something else has already happened. We can describe the probability that event X will happen given event Y has happened as

$$P(X \mid Y) = \frac{P(X \cap Y)}{P(Y)}$$

And Bayes's Theorem gives the relationship between $P(X|Y)$ and $P(Y|X)$:

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)}$$

Which provides a way of calculating the posterior probability, $P(Y \mid X)$, from $P(Y)$, $P(X)$, and $P(X \mid Y)$. Naive Bayes classifier assumes that the effect of the value of a predictor $(X)$ on a given class $(Y)$ is independent of the values of other predictors.

This assumption of independence among predictors is called class conditional independence. It reduces the complexity of Bayes' theorem and dramatically reduces the number of parameters to be estimated when modeling $P(X \mid Y)$. Consider the case that $X = (X_1, X_2)$, with the conditional independence assumption,

$$\begin{aligned} P(X \mid Y) &= P(X_1, X_2 \mid Y) \\ &= P(X_1 \mid X_2, Y)P(X_2 \mid Y) \\ &= P(X_1 \mid Y)P(X_2 \mid Y) \end{aligned}$$

More generally [19], when $X$ contains $n$ attributes which are conditionally independent of one another given $Y$, we have

$$P(X_1, \ldots, X_n \mid Y) = \prod_{i=1}^{n} P(X_i \mid Y)$$

Notice that when $Y$ and the $X_i$ are boolean variables, we need only $2n$ parameters, instead of the original $2(2^n - 1)$, to define $P(X_i = x_{ik} \mid Y = y_j)$ for the necessary $i, j, k$.

Given a class variable $Y$ and a dependent feature vector $X_1$ through $X_n$, Bayes' theorem states the following relationship:

$$P(Y \mid X_1, \ldots, X_n) = \frac{P(Y)P(X_1, \ldots X_n \mid Y)}{P(X_1, \ldots, X_n)} \tag{16}$$

Using the naive independence assumption

$$P(X_i|Y, X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n) = P(X_i|Y),$$

We can rewrite the previous expression (16) as

$$P(Y \mid X_1, \ldots, X_n) = \frac{P(Y) \prod_{i=1}^{n} P(X_i \mid Y)}{P(X_1, \ldots, X_n)}$$

Since $P(X_1, \ldots, X_n)$ is constant given from the input, we can use the following classification rule:

$$P(Y \mid X_1, \ldots, X_n) \propto P(Y) \prod_{i=1}^{n} P(X_i \mid Y)$$

which can be simplified to the following:

$$\hat{Y} = \arg\max_Y P(Y) \prod_{i=1}^{n} P(X_i \mid Y) = \arg\max_Y \log P(Y) + \sum_{i=1}^{n} \log P(X_i \mid Y)$$

Please also note that for each predictor $X_i$ given $Y$, we generally assume that it is normally distributed, but we can also use kernel distribution to make it more flexible at some expense of speed.

Let's understand the classifier with an example [22]. For example, we have 1000 pieces of fruits ($Y$), consisting of `Banana` ($Y = 1$), `Orange` ($Y = 2$) and some `Other fruit` ($Y = 3$). We know three features ($X_i$) about each fruit: (i) whether it is `Long` ($X_1$); (ii) whether it is `Sweet` ($X_2$); and (iii) If its color is `Yellow` ($X_3$). Our training set were provided as table below, and we will use this to predict any type of new fruit as we encounter.

| Type | Long | Not Long | Sweet | Not Sweet | Yellow | Not Yellow | Total |
|---|---|---|---|---|---|---|---|
| Banana | 400 | 100 | 350 | 150 | 450 | 50 | 500 |
| Orange | 0 | 300 | 150 | 150 | 300 | 0 | 300 |
| Other Fruit | 100 | 100 | 150 | 50 | 50 | 150 | 200 |
| Total | 500 | 500 | 650 | 350 | 800 | 200 | 1000 |

Table 3: Naive Bayes Fruit Example "Traing set" Data

Here's what know about our fruit based on the "training set":

| | | |
|---|---|---|
| P(Banana) = P($Y = 1$) | = | 0.5 (500/1000) |
| P(Orange) = P($Y = 2$) | = | 0.3 |
| P(Other Fruit) = P($Y = 3$) | = | 0.2 |
| p(Long) = P($X_1 = 1$) | = | 0.5 |
| P(Sweet) = P($X_2 = 1$) | = | 0.65 |
| P(Yellow) = P($X_3 = 1$) | = | 0.8 |
| P(Long\|Banana) = P($X_1 = 1 \mid Y = 1$) | = | 0.8 |
| P(Long\|Orange) = P($X_1 = 1 \mid Y = 2$) | = | 0 |
| .... | | |
| P(Yellow\|Other Fruit) = P($X_3 = 1 \mid Y = 3$) | = | 0.25 |
| P(Not Yellow\|Other Fruit) = P($X_3 = 0 \mid Y = 3$) | = | 0.75 |

Now we were given a new fruit that is `Long, Sweet, and Yellow`. What kind of fruit is it? We can calculate the probabilities of whether it is `Banana, Orange` or `Other fruit`, and "classify" the unknown fruit to the class that has the highest probability based on our prior evidence.

$$\texttt{P(Banana|Long, Sweet and Yellow)} = \mathtt{P}(Y = 1|X_1 = 1, X_2 = 1 \text{ and } X_3 = 1)$$

$$= \frac{\texttt{P(Long|Banana) * P(Sweet|Banana) * P(Yellow|Banana) * P(banana)}}{\texttt{P(Long) * P(Sweet) * P(Yellow)}}$$

$$= \frac{\mathtt{P}(X_1 = 1|Y = 1) * \mathtt{P}(X_2 = 1|Y = 1) * \mathtt{P}(X_3 = 1|Y = 1) * \mathtt{P}(Y = 1)}{\mathtt{P}(X_1 = 1) * \mathtt{P}(X_2 = 1) * \mathtt{P}(X_3 = 1)}$$

$$= 0.8 * 0.7 * 0.9 * 0.5/\texttt{P(evidence)}$$

$$= 0.252/\texttt{P(evidence)}$$

`P(Orange|Long, Sweet and Yellow) =` $\mathtt{P}(Y = 2|X_1 = 1, X_2 = 1, X_3 = 1)$ `= 0`
Since we know `Oranges` are never long in all the fruit that we have.

`P(Other Fruit|Long, Sweet and Yellow)`

$$= \frac{\texttt{P(Long|Other fruit) * P(Sweet|Other fruit) * P(Yellow|Other fruit) * P(Other Fruit)}}{\texttt{P(evidence)}}$$

$$= \frac{\mathtt{P}(X_1 = 1|Y = 3) * \mathtt{P}(X_2 = 1|Y = 3) * \mathtt{P}(X_3 = 1|Y = 3) * \mathtt{P}(Y = 3)}{\texttt{P(evidence)}}$$

$$= (100/200 * 150/200 * 50/150 * 200/1000)/\texttt{P(evidence)}$$

$$= 0.01875/\texttt{P(evidence)}$$

Because (0.252 >> 0.01875), we classify the `Sweet/Long/Yellow` fruit as likely to be a `Banana`.

### 5.1.1 Results based on Springleaf Data

While the Naive Bayes classifier runs very fast compared with other methods we have tried, the results are not very satisfactory. We randomly picked 2/3 of the training data as the training set and 1/3 as the testing set and achieved an average error rate of roughly 27.2% over 4 trials. One of the sample confusion tables is shown below, we can see that the prediction on target 1 is only about 40.5% correct.

| | | truth |
|---|---|---|
| predict | 0 | 1 |
| 0 | 31260 | 7309 |
| 1 | 5859 | 3983 |

Table 4: Naive Bayes Confusion Table

## 5.2 Maximum A Posteriori Probability

In Bayesian statistics, a maximum a posteriori probability (MAP) estimate is a mode of the posterior distribution. According to wikipedia,” the MAP can be used to obtain a point estimate of an unobserved quantity on the basis of empirical data. It is closely related to Fisher’s method of maximum likelihood (ML), but employs an augmented optimization objective which incorporates a prior distribution over the quantity one wants to estimate.”

MAP is good for classification because it can to find the optimal point, line or hyperplane that separate observations to different groups. To understand MAP, we briefly review Bayes’ rule. We can calculate the prior probabilities from the frequencies of Group $A$(target = 0) and Group $B$(target = 1). From training data there are $n$ observations, $n_1$ 0s and $n_2$ 1s so

$$P(A) = \frac{n_1}{n};\ P(B) = \frac{n_2}{n}$$

Given another single event Y, the conditional odds of A and B are $P(A|Y)$ and $P(B|Y)$. So the probability of Y is:

$$P(Y) = P(Y|A)P(A) + P(Y|B)P(B)$$

If we want to know the test observations are belong to group A or group B, we need to calculate the probability of A or B, given Y. Here Y is referred to the new event we sample the customer from population.
So, we have:

$$P(A|Y) = \frac{P(Y|A)P(A)}{P(Y|A)P(A) + P(Y|B)P(B)}$$

$$P(B|Y) = \frac{P(Y|B)P(B)}{P(Y|A)P(A) + P(Y|B)P(B)}$$

In the two equations, we already know P(A) and P(B) from the training data. And we still need to have $P(Y|A)$ and $P(Y|B)$ to calculate $P(A|Y)$. Assume each group are normal distribution, so we have probability density functions

$$P(A|Y) = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{(y-\mu_1)^2}{2{\sigma_1}^2}}$$

$$P(B|Y) = \frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{(y-\mu_2)^2}{2{\sigma_2}^2}}$$

Above $\mu_1$ and $\sigma_1$ for Group A, $\mu_2$ and $\sigma_2$ for Group B.

For each test observation, we compare $P(A|Y)$ and $P(B|Y)$, If $P(A|Y) > P(B|Y)$, we assign it to Group A. If $P(A|Y) < P(B|Y)$, we assign it to Group B. If If $P(A|Y) = P(B|Y)$, we assign it randomly or using other method.

In order to simplify, we rewrite the two equations to one using ratio.

$$\frac{P(A|Y)}{P(A|Y)} = \frac{e^{-\frac{(y-\mu_1)^2}{2{\sigma_1}^2}}}{e^{-\frac{(y-\mu_2)^2}{2{\sigma_2}^2}}}$$

By extension from this simple two dimensions example, we have formula for k multiple groups.

$$P(E_i|Y) = \frac{e^{-\frac{(y-\mu_i)^2}{2{\sigma_i}^2}} n_i}{\sum_{i=1}^{k} e^{-\frac{(y-\mu_i)^2}{2{\sigma_i}^2}} n_i}$$

### 5.2.1 Results based on MNIST Data

The results of Naive Bayes and MAP can be found in Figure 14

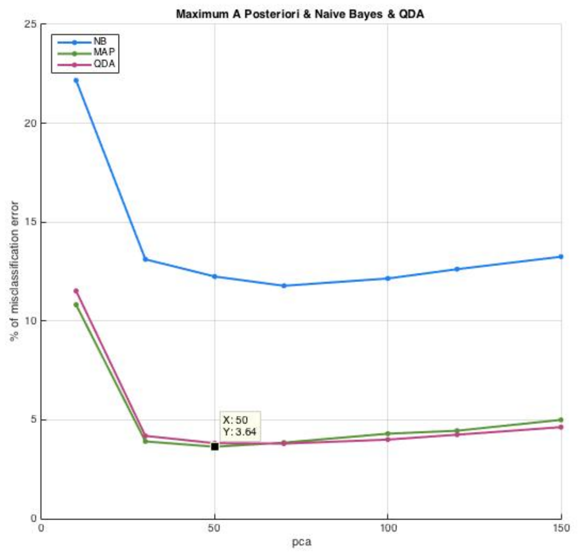


Figure 14

## 5.3 Logistic Regression

In statistics, logistic regression is a regression model where the dependent variable is categorical. In our case, the dependent variable only has two categories, which are 0 and 1. We call this a binary case. If the dependent variable has more than two categories, it is referred to as multinomial logistic regression.

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor variables.

Logistic regression can be seen as a special case of generalized linear model and thus analogous to linear regression. The model of logistic regression is based on different assumptions about the relationship between dependent and independent variables. First, the conditional distribution $y \mid x$ is a Bernoulli distribution rather than a Gaussian distribution, because the dependent variable is binary. Second, the predicted values are probabilities are restricted to (0,1) through the logistic function.

Before introducing logistic model first define the log odds ratio. For a binary logistic model, we have two categories 0 and 1. $P(y=1|X)$ is the probability that we get category 1 and $P(y=0|X)$ is the probability that we get category 0. The relation $P(y=0|X)+P(y=1|X)=1$ holds. Then

$$f(x) = logit(Y) = \log\frac{P(y=1|X)}{1-P(y=1|X)} = \log\frac{P(y=1|X)}{P(y=0|X)} \tag{17}$$

The expression in Equation (17) is called the log odds ratio. Based on this definition, we now introduce the logistic model. Suppose our independent variables are $x_1, x_2, ..x_d$, we define

$$z = f(X) = b + w_1x_1 + w_2x_2 + ... + w_dx_d = w^TX + b$$

where $w = [w_1, w_2, ..., w_d]$ and $X = [x_1, x_2, ..., x_d]$. In order to turn this into a probabilistic statement, we use a function whose target is restricted [0,1]. The logistic function

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{18}$$

where $z = w^TX + b$ is such a function. Equation (18) is called logistic function. As z goes from $-\infty$ to $\infty$, $\sigma(Z)$ goes from 0 to 1, which is the probability of particular outcomes. Based on the values of probability, people can select different thresholds that will categorize each observation into different groups and the value 0.5 is a typical one. Based on the predicted categories, we can compare the original data with predicted ones. As a result, the accuracy of the logistic regression model can be calculated.

### 5.3.1 Results based on Springleaf Dataset

In this part, we will show the result of logistic regression based on both original dataset and dataset based on principal component analysis. In the analysis, we divided our data into two parts: 2/3 are treated as train data, and 1/3 are treated as test data. After doing these separately, we will compare the two situations and give a conclusion.

**ROC curve and AUC**

First, we would like to introduce ROC curve and AUC, which are two important concepts and will be used in the several parts in our analysis.

Receiver operating characteristic, or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. This curve is built by plotting the true positive rate (TPR) against the false positive rate (FPR) at different threshold values.

The true positive rate (TPR) is called sensitivity, and it measures the proportion of positives that are correctly identified as such. False positive rate (FPR) is also called specificity and measures the proportion of negatives that are correctly identified as such. Thus sensitivity quantifies the avoiding of false negatives, as specificity does for false positives. Table 5 shows the definition of true positive and false positive.

Table 5: Contingency Table

| | | **True Condition** | |
|---|---|---|---|
| | | Positive | Negative |
| **Predicted Condition** | Positive | True Positive | False Positive (Type I error) |
| | Negative | False Negative (Type II error) | True Negative |

Generating a ROC curve is simple: write the probability density for belonging to the class as a function of a threshold parameter T , as $P_1(T)$ . Write the probability density for not belonging to the class as $P_0(T)$ . The false positive rate FPR is given by

$$\text{FPR}(T) = \int_T^\infty P_0(T')dT'$$

and the true positive rate is

$$\text{TPR}(T) = \int_T^\infty P_1(T')dT'$$

The ROC curve plots parametrically TPR(T) versus FPR(T) as a function of T. The optimal point on the ROC curve is (FPR, TPR) = (0,1). No false positives and all true positives which is ideal. An essential observation is that the curve is by definition monotonically increasing.

$$\text{FPR}(\theta) < \text{FPR}(\theta') \Rightarrow \theta > \theta' \Rightarrow TPR(\theta) \leq TPR(\theta')$$

All those features combined make it apparently reasonable to summarize the ROC into a single value by calculating the area of the convex shape below the ROC curve this is the AUC. The closer the ROC gets to the optimal point of perfect prediction the closer the AUC gets to 1.

**Results Based on Original Dataset**

In this Springleaf dataset, our dependent variable 'target' has two values: target = 0 means people will not accept the company's offer, and target = 1 means people will accept the offer. So our problem is actually a binary logistic regression case. It is reasonable if we plan to use logistic regression on our dataset.

Figure 15 shows the distribution of prediction probability for each data point. We can see most of the points are located between 0 and 0.5, and a small proportion of points locate between 0.5 and 1. It makes sense because in our data, the number of 0 is much more than number of 1. In order to see how good the prediction is, we need some more figures, tables and deep analysis.

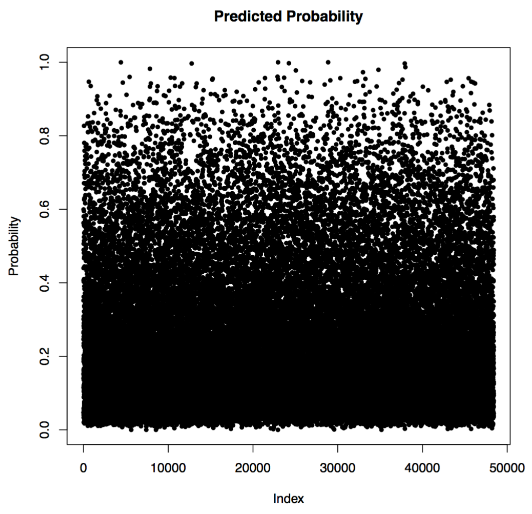


Figure 15: Predicted Probability

By default, people use 0.5 as the cutoff for classification in logistic regression, which means in our case, the points with probabilities in the interval (0,0.5) belong to category 0 and the points with probabilities in interval (0.5,1) belong to category 1. Based on this standard, we get the confusion matrix which is as shown in Table 6.

According to the confusion matrix, we get the predicting accuracy is 78.60%. We could also get the accuracy for the two categories separately, as shown in Table 6, the accuracy for category 0 is 94.25% and the accuracy for category 1 is 27.40%.

Table 6: Confusion Matrix

| | | **Predicted** | | **Accuracy** |
|---|---|---|---|---|
| | | 0 | 1 | |
| **True** | 0 | 34943 | 2129 | 94.25% |
| | 1 | 8232 | 3107 | 27.40% |

Actually, we can choose what value should be used in practice. So we can try different cutoffs and see which one will give us the highest accuracy. Based on this, we get Figure 16. This covers a range of cutoffs from [0.4, 0.6]. The results show that when we pick cutoff at 0.5, the accuracy is the best.

We also have the ROC curve as shown in Figure 17. The area under the curve is : 0.7608.

Accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test; an area of .5 represents a worthless test. A rough guide for classifying the accuracy of a diagnostic test is the traditional academic point system: $.90 - 1.0$ =excellent (A); $.80 - .90$ =good (B); $.70 - .80$ =fair (C);$.60 - .70$ =poor (D); $.50 - .60$ =fail (F).

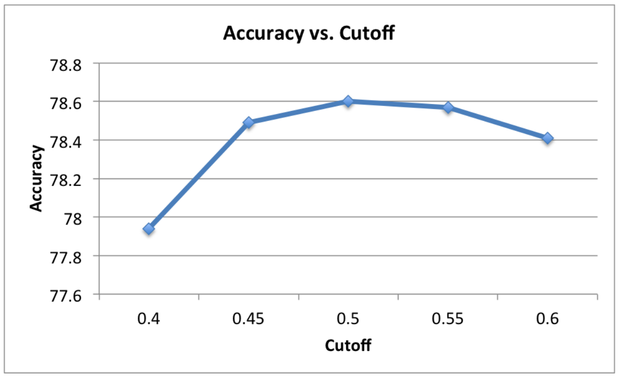


Figure 16: Accuracy vs. Cutoff

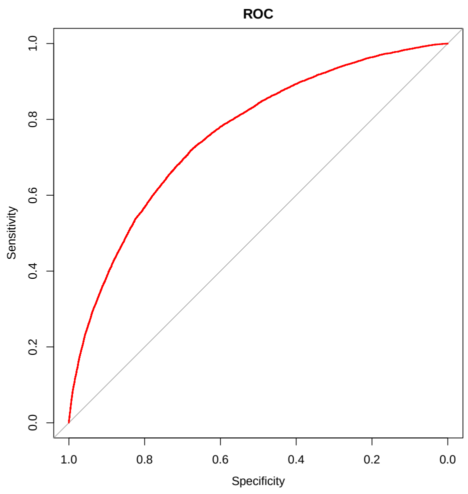


Figure 17: ROC Curve Based on Original Dataset

### Results Based on PCA

Since we have too many features in our original dataset, we consider to use different principal components to see whether less components will give us higher accuracy.

Figure 18 is a plot of number of principal component versus accuracyy. The largest number of principal components we choose is 320, which explained 90% of the total variation. Overall, we can see a trend that the more principal component, the more accurate our predictions are. The best result we have is 78.5%.

Figure 19 shows several different accuracies: green line is accuracy for target = 0, purple line is accuracy for target = 1, red line is overall accuracy, and blue line is area under the ROC curve. Through these lines, we also see the trend: the more components we have in the model, the better results we will get.

As a result, we conclude for logistic regression, the result based on original data (accuracy is 79.2) is better than result based on dataset obtained by PCA (best accuracy is 78.5%). Overall, logistic regression performs well on Springleaf dataset.

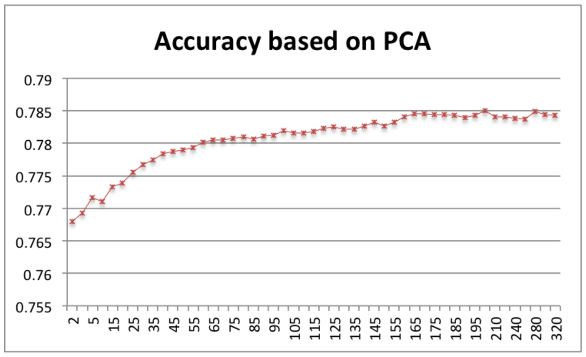


Figure 18: Accuracy based on PCA

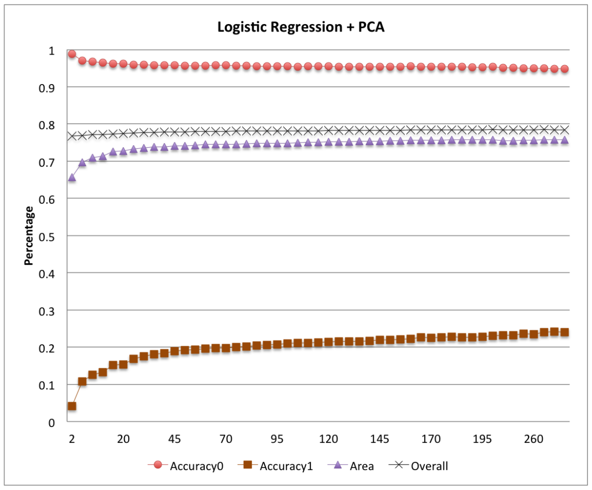


Figure 19: Accuracy + AUC

## 5.4 Using LDA and QDA as Classifiers

In the previous section, we have discussed LDA as dimension reduction method. In this section, we will give a short introduction of LDA and QDA as classifiers. LDA and QDA are very similar to each other, except for LDA we assume each group has the same covariance while for QDA each group has its own covariance.
To use LDA and QDA as classifiers, one commonly used approach is the naive Bayesian classifier. Consider the conditional distribution of the data $P(X|l=i)$ for each class $i$, the predictions can be obtained by using Bayes' rule

$$P(l=i|X) = \frac{P(X|l=i)P(l=i)}{P(X)} = \frac{P(X|l=i)P(l=i)}{\sum_j P(X|j=i)P(j=i)},$$

where we use the posterior $P(l=i|X)$ to predict if $l$ belongs to class $i$, and $P(l=i)$ is the prior probability of class $i$. $P(X|l=i)$ is the probability of $x$ when it belongs to class $i$. We predict the test data to the class $i$ which maximizes the above conditional probability [5].

Assume $P(X|l)$ for LDA and QDA is a multivariate Gaussian distribution with density

$$f(X|l=i) = \frac{1}{(2\pi)^{p/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(X-\mu_i)^T\Sigma_i^{-1}(X-\mu_i)}$$

The prior probability of class $i$ is $P(l=i)$, $\sum_{l=1}^{c} P(l=i)=1$. $P(l=i)$ is estimated simply by empirical frequencies of the training set

$$\widehat{P(l=i)} = \frac{n_i}{\sum_i n_i}$$

where $n_i$ is the number of class-$i$ samples.

Using Bayes rule, we can obtain the optimal function

$$arg \max_i P(l=i|X) = arg \max_i P(X|l=i)P(l=i)$$

## 5.5 Mathematical formulation of the QDA classifier

In QDA, the covariance of each group is different from each other. Estimate the covariance matrix $\Sigma_i$ separately for each class $i$, $i=1,2,\ldots,c$ [21]. $\pi_i$ is an estimator of $P(l=i)$ .

Following similar derivation from LDA, the Quadratic discriminant function is

$$\delta_i(x) = -\frac{1}{2}\log|\Sigma_i| - \frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i) + \log \pi_i$$

and the classification rule is

$$arg \max_i \delta_i(x),$$

where the decision boundary is quadratic.

### 5.5.1 MNIST results

We have applied both LDA and QDA on the MNIST data. Since the original MNIST data has singularity issue when we apply LDA/QDA directly, we use PCA to regularize the data first. When we use 250 PCA components and LDA as the classifier, the misclassification error is 0.125 based on original data and the misclassification error is 0.08 based on deskewed data. When using 250 PCA components and QDA as the classifier, the misclassification error is 0.0649 based on original data and the misclassification error is 0.0516 based on deskewed data. Our study shows that QDA performs better than LDA on MNIST data set. However, QDA has more parameters to tune and is harder to train.

## 6 Support Vector Machines

### 6.1 The linearly separable case

A data set in $\mathbb{R}^2$ is said to be linearly separable if there exists at least one line in the plane that can separate one class of the data set on one side of the line and the other class on the other side of the line. If the data set is multidimensional, then a hyperplane is used. Figure 20(a) shows an example of a linearly separable data set.

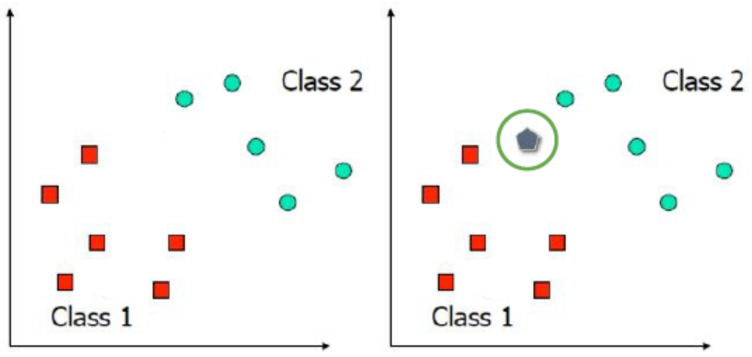


Figure 20: (a) A data set with two classes, red and cyan. The data set is linearly separable because both classes can be separated by a line. (b) A new observation, shown in gray, becomes available. The task is to determine whether it belongs to the red class or the cyan class

Suppose now that a new observation, as shown in Figure 20(b) is obtained and needs to be classified. Since the data set is linearly separable, a number of models (see Figure 21), each with their corresponding decision boundaries, can be obtained to classify the new observation. However, the models provide conflicting classifications. Two models predict the new observation as being red, while the other predicts the observation to be cyan. Each model shows how the choice of the decision boundary is critical to classification. For example, in model 1 of Figure 21, the decision boundary is closer to class 2, thus classifying new observations as being red more often. On the other hand, Model 2 does the exact opposite, classifying new observations as being cyan more often than not.

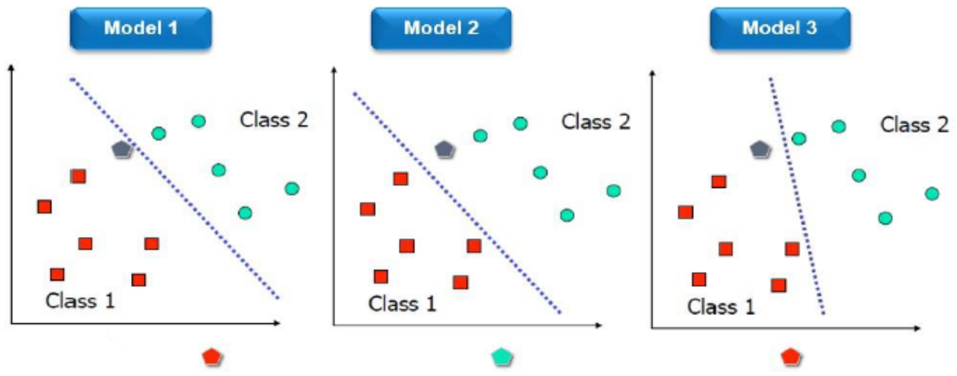


Figure 21: Each model has a different decision boundary for classification. From left to right, the new gray observation was classified as red, cyan, and red for models 1, 2, and 3, respectively.

Finally, the last model will have many misclassifications because part of the decision boundary is so close to one class: the new observation will be classified as red even when this new observation is much more similar ("close") to the cyan class. Figure 21 suggests that it is not clear which decision boundary is best, but this is where SVM comes in. The main idea of SVM, which stands for "Support Vector Machines" is to construct a decision boundary that maximizes the separation between both classes–that way, each class would have a "fair" chance of claiming the new observation as part of its class. This classification method was introduced by Vapnik et al. [25, 4]. The details follows next.

Let $\{x_i, y_i\}$ for $i = 1, \ldots, n$, and $x_i \in \mathbb{R}^d$ be the data set, and without loss of generality, suppose the labels are given by $y_i \in \{-1, 1\}$. Suppose further that the data set is linearly separable so that the linear boundary is given by the hyperplane

$$w^T x + b = 0 \tag{19}$$

where $w$ is normal to the hyperplane, as in Figure 22. Let $x_+$ and $x_-$ denote points belonging to the class with labels $+1$ and $-1$, respectively. Then for each data point, and with out loss of generality, we impose the decision rules[1] $w^T x_+ + b \geq +1$ and $w^T x_- + b \leq -1$, which can be written compactly as

$$y_i(w^T x_i + b) \geq 1 \tag{20}$$

For sample points $x_i$ that fall on the margin boundaries, we can define

$$y_i(w^T x_i + b) = 1 \tag{21}$$

Note that $w^T x + b = +1$ and $w^T x + b = -1$ are hyperplanes parallel to $w^T x + b = 0$ (see Figure 22). Moreover, since we want to maximize the separation of both classes, $w^T x + b = +1$ will contain the closest $x_+$ to $w^T x + b = 0$ so that $w^T x_+ + b = +1$. Similarly for the closest $x_-$ we have $w^T x_- + b = -1$.

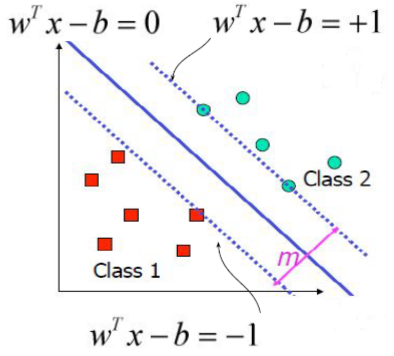


Figure 22: Formulation of the margin in SVM

Let $m$ be the width between the hyperplanes $w^T x_- + b = -1$ and $w^T x_- + b = +1$, often referred to as the *margin*. Then $m = (x_+ - x_-) \cdot \dfrac{w}{||w||}$, where $\dfrac{w}{||w||}$ is a unit vector perpendicular to $w^T x_- + b = 0$. Then using the fact that $w^T x_+ + b = +1$ and $w^T x_- + b = -1$, the $m$ simplifies to

$$m = \frac{2}{||w||} \tag{22}$$

Our goal here is to maximize separation between both classes by solving the problem: $\max \dfrac{2}{||w||}$. However for mathematical convenience this last maximization problem may be written as a minimization problem as follows

$$\begin{aligned} &\min_{w \in \mathbb{R}^d} \frac{1}{2}||w||^2 \\ &\text{subject to: } y_i(w^T x_i + b) \geq 1 \text{ for } i = 1, 2, \ldots, n \end{aligned} \tag{23}$$

[1]Had we chosen $w^T x_+ + b \geq +\delta$ and $w^T x_- + b \leq -\delta$, for $\delta > 0$ we can rescale by dividing through by $\delta$ to get $\tilde{w}^T x_+ + b \geq +1$ and $\tilde{w}^T x_- + b \leq -1$. It can be shown that the end result is the same.

Here $J(w) := \frac{1}{2}||w||^2$ is a quadratic function with linear constraints and so has a global minimum. This last minimization problem is equivalent to

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{i=1}^{n} \alpha_i\alpha_j y_i y_j x_i^T x_j \\ &\text{subject to: } \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ for } i = 1, 2, \ldots, n \end{aligned} \tag{24}$$

Suppose $\alpha = (\alpha_1, \ldots, \alpha_n)$ is the optimal parameter values, then it can be shown [ciiite] that this optimization problem can be solved using Lagrange multipliers and that the solution is given by

$$w = \sum_{i=1}^{n} (\alpha_i y_i)\, x_i, \text{ and } w_0 = \frac{1}{z_i} - w^T x_i$$

Moreover, for every sample $x_i$, one of the following must hold: Either (a) $\alpha_i = 0$ corresponds to a sample that is not a support vector[2], or $\alpha_i \neq 0$ and $y_i\left(w^T x_i + w_0 - 1\right) = 0$ in which case this is a support vector.

## 6.2 Soft-margin extension

If the data is not perfectly linearly separable or "almost" linearly separable we may allow some data points in one class to appear on the other side of the boundary. This situation can also arise in the presence of outliers, in which case some data points will not lie close to the bulk of the data (see Figure 23 (a)).

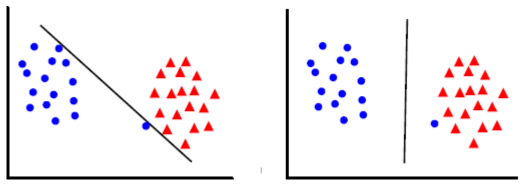


Figure 23: An example of a data set that is "almost" linearly separable. (a) In the left graph, the hyperplane is influenced by an outlier. Notice that in this case the margin would have to be thin. (b) In the right graph, allowing a mistake (misclassification) in the training set allows the construction of a line that separates both sets.

To handle this scenario, we can introduce *slack variable* $\xi_i \geq 0$ for each $x_i$ so that the minimization problem, with cost function $J = J(w, \xi_i, \ldots, \xi_n)$ and new constraints, can be written as [SVM tutorial]

$$\begin{aligned} &\min_{w\in\mathbb{R}^d, \xi_i\in\mathbb{R}^+} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} I(\xi_i > 0) \\ &\text{subject to: } y_i(w^T x_i + b) \geq 1 - \xi_i \text{ for } i = 1, 2, \ldots, n \end{aligned} \tag{25}$$

Here $\xi_i$ is a measure of deviation from the ideal sample $x_i$: (i) if $x_i$ is on the wrong side of the separating hyperplane, then $\xi_i > 1$. This is a misclassification. (ii) if a sample $x_i$ is on the right side of separating hyperplane but within the region of maximum margin, then $0 < \xi_i \leq 1$. This is a margin violation. (iii) if $\xi_i < 0$, the ideal case, then $x_i$ is on the correct side of the of the hyperplane, outside the margin. The three scenarios are depicted in Figure 24.

[2] DEFINE SUPPORT VECTOR

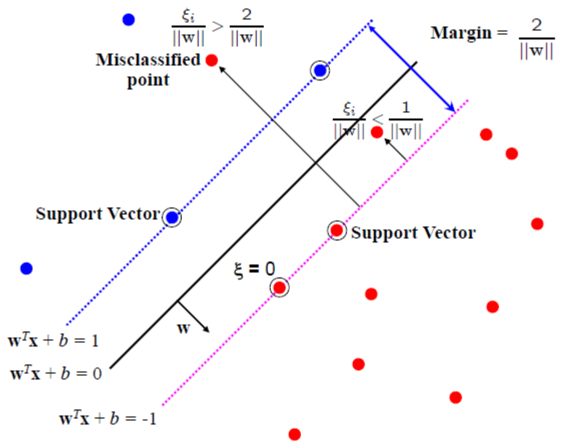

$\frac{\xi_i}{||w||} > \frac{2}{||w||}$
Misclassified point
Margin = $\frac{2}{||w||}$
$\frac{\xi_i}{||w||} < \frac{1}{||w||}$
Support Vector
Support Vector
$\xi = 0$
w
$w^T x + b = 1$
$w^T x + b = 0$
$w^T x + b = -1$

Figure 24: Data points $x_i$ corresponding to different values of $\xi_i$.

The term $I(\xi_i > 0)$ is 1 for $\xi_i > 0$ and 0 if $\xi_i \leq 0$. The constant $C$ is a *regularization parameter*. If $C$ is small, we allow a lot of samples not in ideal position giving a large margin. If $C$ is large, we want to have very few samples not in the ideal position thus giving a narrow margin. If $C \to \infty$, then all constraints are enforced giving the hard margin as in section 6.1. However, it is hard to solve this minimization problem because of the discontinuity of the indicator function, so instead we consider

$$\min_{w\in\mathbb{R}^d, \xi_i \in \mathbb{R}^+} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \xi_i \qquad (26)$$
$$\text{subject to: } y_i(w^T x_i + b) \geq 1 - \xi_i \text{ for } i = 1, 2, \ldots, n$$

where $\sum_{i=1}^{n} \xi_i$ is a measure of the number of misclassified examples (see the last paragraph of Section 6.1).

## 6.3 SVM with multiple classes

SVM is a binary classifier that can easily be extended to more than two classes. Two of the most common methods are (1) One vs. Rest, and (2) Pairs (also known as One vs. One). The concept of each method is straightforward. For example, for the one vs rest case, suppose we have three classes, red, green, and yellow (see Figure 25). Then for each class, a SVM model is constructed, where each model will classify between being red or not red, green or not green, and yellow or not yellow, respectively. Thus if there are $c$ classes, then $c$ binary SVM models must be constructed. Then for each new observation, each of the $c$ models makes its corresponding prediction.

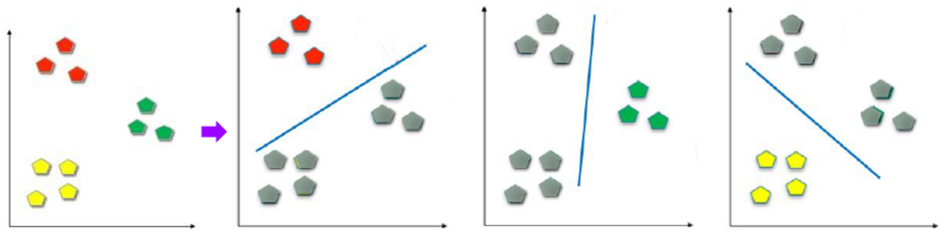


Figure 25: Application of the One vs. Rest method to SVM.

Note that for any given observation it is possible to have conflicting predictions (see Figure 26). A common method consists of taking a majority vote between all models, but a safer approach consists of taking the observation that is farthest from the decision boundary for each of the SVM models (see Figure 27)

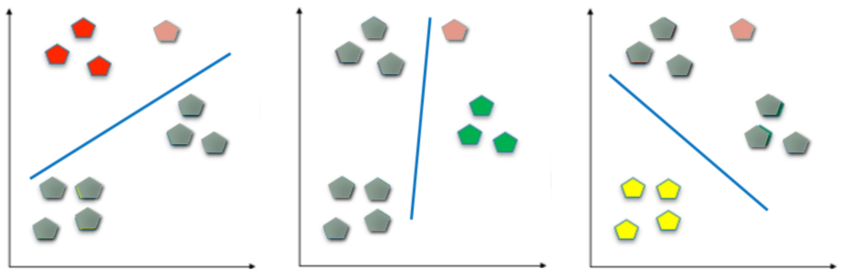


Figure 26: An example of when the one vs. rest method gives conflicting predictions. Here the first and third models disagree with the second model.

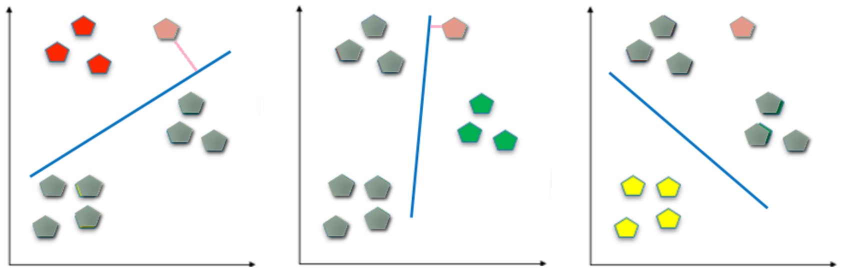


Figure 27: Using the farthest distance to the decision boundary to handle conflicting predictions.

### 6.3.1 MNIST Results

Our first attempt at the MNIST data using SVM consisted of using the multiclass version. Using the linear classifier provided errors greater than 1.60%. A small sample of results is provided in Table 7.

| **Method** | **Parameters** | **Error** |
|---|---|---|
| tsne + linear SVM | $d = 3,\ pcad = 30,\ perpl = 30,\ \theta = 0.5$ | 3.00% |
| global PCA + linear SVM (pairs) | $PCA = 41$ | 1.63% |

Table 7: Summary of Instance based methods on the MNIST data set.

The results are poor as expected, because most data sets fail to be linearly separable. Figure 28 shows a visualization of SVM in two dimensions applied to digits 0 and 1.

## 6.4 Kernel SVM

In the previous section, we discuss the formulation of linear SVM. However, real world datasets are usually not linearly separable [1], where linear SVM will fail to converge or yield ill results. Can we generalize the linear SVM method so that it can be applied to datasets which does not have linear boundaries? Notice that the solution for linear SVM [23] lies in maximize the following optimal function (24).
The above optimization only depends on the training data through the dot product $x_i^T x_j$. One simple remedy is using the "Kernel Trick". The idea is to map the data into a higher dimensional feature space $Q$, where the data is linear separable and the linear SVM is applicable. Following the derivation of [3], we write the transformation function $\Phi$ as

$$\Phi : R^m \to Q$$

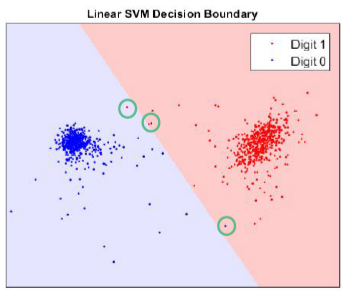


Figure 28: Visualization of a linear decision boundary for digits 0 and 1.

Next, we need to find the dot products of each paired data in the space $Q$. The form of the dot products between $x_i$ and $x_j$ in $Q$ can be written as

$$K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j),$$

which is the so-called kernel function, which avoids the complexity of dealing with data in the mapped high dimensional feature space. The Lagrangian in terms of the kernel function becomes

$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j \ K(x_i, x_j).$$

Since the kernel function only depends explicitly on the original data , we can apply the usual linear SVM in the original data space instead of the high dimensional feature space which may have infinite dimensions. We do not need to calculate the actual transformation $\Phi$, everything is embedded in the kernel function. In practice, there are three common used kernel functions. The Polynomial Kernel

$$K(x_i, x_j) = (x_i^T x_j + \theta)^p,$$

where $p$ and *theta* are user defined parameters. The Gaussian Kernel

$$K(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right),$$

where $\sigma$ is defined by the user. The Sigmoid Kernel

$$K(x_i, x_j) = \tanh(\eta x_i^T x_j + \theta),$$

with user defined parameters $\eta$ and $\theta$.

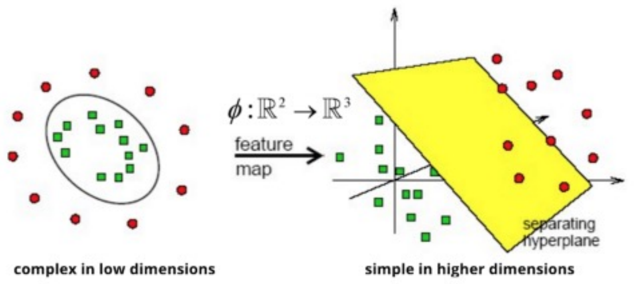

$\phi : \mathbb{R}^2 \to \mathbb{R}^3$
feature map
separating hyperplane
complex in low dimensions
simple in higher dimensions

Figure 29: The mapping of kernel SVM

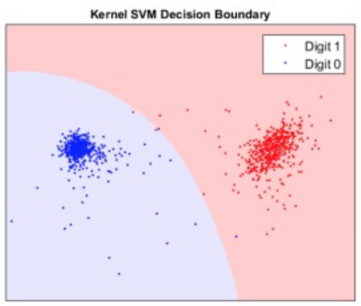


Figure 30: The decision boundary using kernel SVM based on digit 0 and digit 1 after applying NDA to the pair.

### 6.4.1 SVM Kernel Selection

Our study show that the Gaussian kernel has the best performance

$$K(x_i, x_j) = e^{-\frac{||x_i - x_i||_2^2}{2\sigma^2}}$$

The challenge with our approach is the parameter selection. For each SVM model, we need to tune the kernel parameter $\sigma$ and the outlier tolerance $C$. In practice, $C$ is relatively easier to choose. For $\sigma$, it is difficult to tune. In order to solve this problem, we employ the KNN search to find the best starting point for $\sigma$ search. This method has been widely used in other algorithms such as the spectral clustering. For each group, we find the $k$-th nearest neighbor of each observation and use the average of these distances as the starting point of parameter $\sigma$. This method works very well in our study.

For the classification accuracy, the combined global and local PCA with kernel SVM generate better results than kernel SVM alone. And the local PCA outperforms the global PCA by a small margin. The detailed results are presented in the next section.

In this study, we have applied both Polynomial Kernel and the Gaussian Kernel to the MINIST data. The results show that Gaussian Kernel outperforms Polynomial Kernel or linear SVM by a large margin in terms of accuracy. On the original data set, the smallest misclassification error we have achieved is 1.4% using Gaussian kernel. If we use the deskewed data, the smallest misclassification error is further reduced to 1.03%.

### 6.4.2 MNIST Results

This section lists the best accuracy we achieved for three methods on both the original data and deskewed data, local PCA combined with Gaussian SVM, the global PCA combined with Gaussian SVM, and the global PCA combined with linear SVM. From the following table, we can see that the Gaussian kernel SVM outperforms the linear SVM significantly. And the local PCA combined with SVM ("one-vs-rest") produces slightly better result than the global PCA combined with SVM ("one-vs-one"). Note that in the local PCA method, we achieve this accuracy with only 10 SVM models. As for the global PCA method, one has to train $10 \times (10 - 1)/2 = 45$ SVM models.

### 6.4.3 Springleaf Results

Since SVM is a powerful tool and can be used for analyzing classification problem, we decide to apply it on Springleaf dataset with kernel trick. One weakness is that SVM can only be used on numerical data, but in Springleaf dataset, we have a large proportion of categorical variables. This may influence the accuracy of our prediction.

First, we apply SVM based on the original data. We use 2/3 of the data for building the model, and 1/3 for testing. Figure 31 shows results when Kernel = Linear, Figure 32 shows results when

Kernel = Gaussian. Based on the results, the best accuracy we obtain is 78.10%. In Figure 31 to Figure 34, the x-axis ”index” indicates how many times we did the experiments, and each time we use different values for the parameters.

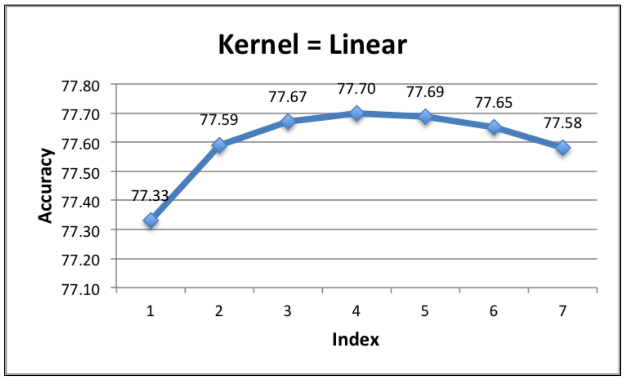


Figure 31: Accuracy when Kernel = Linear

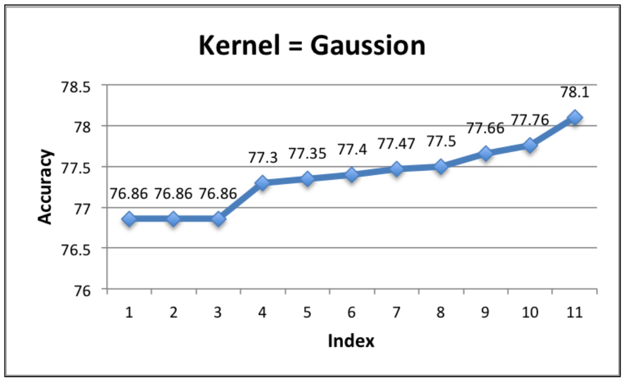


Figure 32: Accuracy when Kernel = Gaussion

As we know, the distribution of Springleaf data is not balanced: the ratio of target = 0 and target = 1 is about 4:1. We guess this unbalanced data structure may have some influence on the model we try to build. So we decide to pick equal observations for target = 0 and target = 1, and build a new model based on these data. Figure 33 shows results when Kernel = Linear, Figure 34 shows results when Kernel = Gaussian. The best accuracy is 78.00%. There is no big improvement when we use balanced data.

SVM is not an ideal method for Springleaf dataset. One reason is it can handle numerical data only, and another reason is SVM is computational expensive. The structure of Springleaf dataset is complex, and it takes many hours for each run when we use this method, which is very inefficient for tuning parameter. From the results of SVM, we can also see this method did not give us very good results. So we consider to use more efficient method.

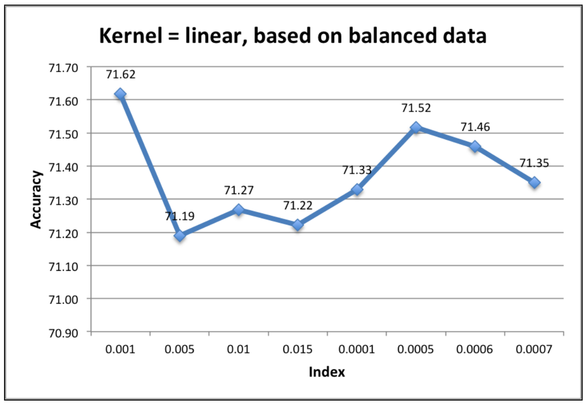


Figure 33: Accuracy based on balanced data (Kernel = Linear)

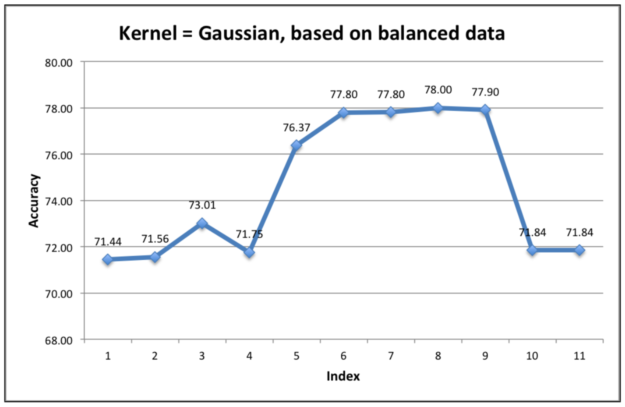


Figure 34: Accuracy based on balanced data (Kernel = Gaussion)

# 7 Ensemble Methods

The idea behind ensemble methodology is to combine a set of models, each of which solves the same original task, in order to obtain a better composite global model with more accurate and reliable estimates or decisions than can be obtained from using a single model. The pairwise and one-vs-rest multiclassifiers are examples of ensembles created from binary classifiers. The idea of building a predictive model by integrating multiple models has been under investigation for a long time. Ensemble methods can be also used for improving the quality and robustness of clustering algorithms.

In the past few years experimental studies conducted by the machine learning community show that combining the outputs of multiple classifiers reduce the generalization error. Ensemble methods are very effective, mainly due to the phenomenon that various types of classifiers have different "inductive biases". The individual estimators provide different patterns of generalization so diversity plays a crucial role in the training process. Ensemble methods can effectively make use of such diversity to reduce the variance-error without increasing the bias-error. The ensemble performs better when each individual machine learning system is accurate and errors are independent.

Given the potential usefulness of ensemble methods it is not surprising many methods are now available to researchers and practitioners. There are many ways to categorize ensemble techniques and there is a recognized group of techniques called data resampling which generates different training sets to obtain unique learner. In this group of methods are bagging, boosting and stacking.

Bagging is short for bootstrap aggregating and is one of the earliest and most intuitive ensemble-based algorithms. Diversity of classifiers in bagging is obtained by using bootstrapped replicas of the training data. That is, different training data subsets are randomly drawn with replacement from the training set. Each training subset is used to train a different classifier of the same type. Individual classifiers are then combined by taking a simple majority vote of their decisions. For any given instance, the class chosen by most number of classifiers is the ensemble decision. Since the training data may overlap substantially, additional measures can be used to increase diversity such as using a subset of the training data for training each classifier.

Boosting provides sequential learning of the predictors. The first one is learned on the whole data set, while the following are learned on the training set based on the performance of the previous one. In other words the instances which were predicted improperly are noted. Then these instances are more probable to appear in the training set of the next predictor. It results in different machines being specialized in predicting different parts of the data.

Stacking is a way of combining multiple models that introduces the concept of a meta-learner. Unlike bagging and boosting, stacking normally is used to combine models of different types instead of using a winner-takes-all approach. We combine the base classifiers, possible nonlinearly, to achieve the highest generalization accuracy.

Next, we will introduce three specific models that apply the above ensemble methods. Firstly, random forest applies the general technique of bootstrap aggregating, or bagging, to tree learners. Secondly, XGBoost is an implementation of the gradient boosting algorithm. Lastly, a specific stacking model will be introduced based on the results of previous methods.

## 7.1 Random Forest

Bagging or bootstrap aggregation is a technique for reducing the variance of an estimated prediction function. Bagging seems to work especially well for high-variance, low-bias procedures, such as trees. For regression, we simply fit the same regression tree many times to bootstrap-sampled versions of the training data and take the average result. For classification, a committee of trees each cast a vote for the predicted class. The random forests algorithm is very much like the bagging algorithm. It is developed based on bagging.

Random forest is an ensemble learning method for classification and regression and is based on a set of decision trees that grow in randomly selected subspaces of data, where each tree in the ensemble is grown in accordance with a random parameter. Final predictions are obtained by aggregating over the ensemble. On many problems the performance of random forests is very similar to boosting, and they are simpler to train and tune. As a consequence, random forests are popular and are implemented in a variety of packages.

Since random forest is convenient and very useful, we decide to use this method on both of our topics. As we know that random forest can handle both categorical variables and numerical variables, it seems perfect for Springleaf project which includes both types of variables.

### 7.1.1 Algorithm

The essential idea is to average many noisy but approximately unbiased models, and hence reduce the variance. Trees are ideal candidates, since they can capture complex interaction. The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

1. For b = 1 to B:
   (a) Draw a bootstrap sample $Z^*$ of size N from the training data.
   (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:
*Regression* : $\hat{f}_r f^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.
*Classification*: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{rf}^B(x)$ = majority vote $\hat{C}_b(x)_1^B$.

When used for classification, a random forest obtains a class vote from each tree, and then classifies using majority vote. When used for regression, the predictions from each tree at a target point x are simply averaged. In addition, the inventors make the following recommendations:

- For classification, the default value for m is $\sqrt{p}$ and the minimum node size is one.
- For regression, the default value for m is $p/3$ and the minimum node size is five.

In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters. Figure 35 shows how we get the final results based on random forest algorithm.

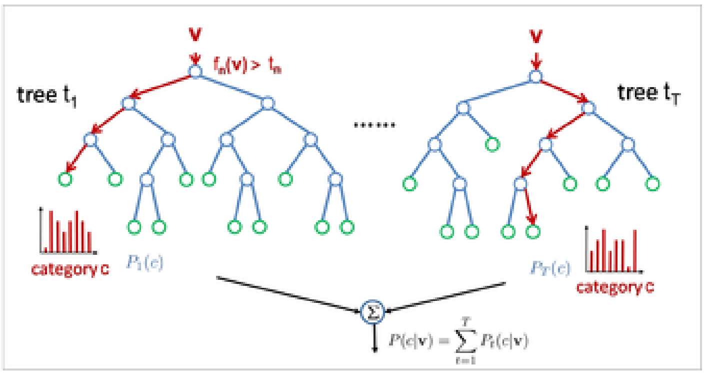


Figure 35: Process of Random Forest

### 7.1.2 Result based on Springleaf Dataset

In this part, we will show the results of random forest based on both original dataset and dataset obtained by principal components analysis. In the analysis, we divided our data into two parts: 2/3 are treated as train data, and 1/3 are treated as test data.

The following results are based on the dataset which removes columns which have more than 50% NA in each column.

Figure 36 shows the results when the number of maximum tree is 100. Figure 37 shows the results when the number of maximum tree is 300. If we focus on Figure 36 (A), the upper line represents error rate for target = 0, the lower line represents error rate for target = 1, and the middle line represents the overall error rate. We can see from tree = 0 to tree = 100, the error line has a decreasing trend. According to Figure 37 (A), when maximum tree is 300, we can see the overall error rate tends to be a constant. In practice, the more trees we generate, the smaller our error rate will be.

Table 10 and Table 9 are the corresponding confusion matrix. The accuracy when tree = 100 is 78.77%, and accuracy when tree = 300 is 78.98%. This verifies that the more trees we use, the better result we will get. Comparing to results we get from other algorithm, we conclude 78.98% is very good in Springleaf case.

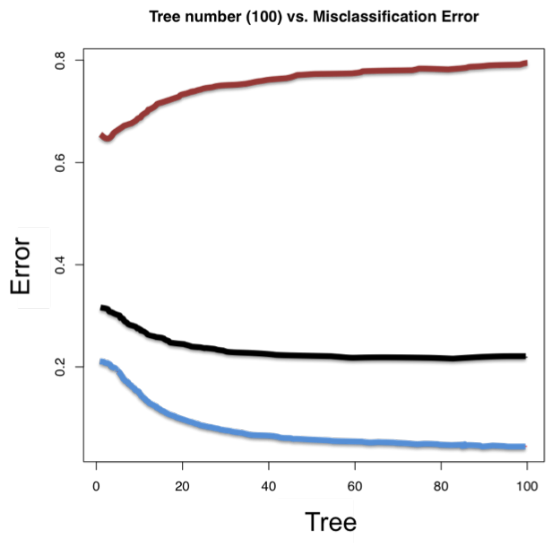
In addition, Figure 36 (B) and Figure 37 (B) give ROC results. The area under the curve when tree = 100 is 0.7734, and area under the curve when tree = 300 is 0.7742, which are very close to each other.

Table 8: Tree = 100

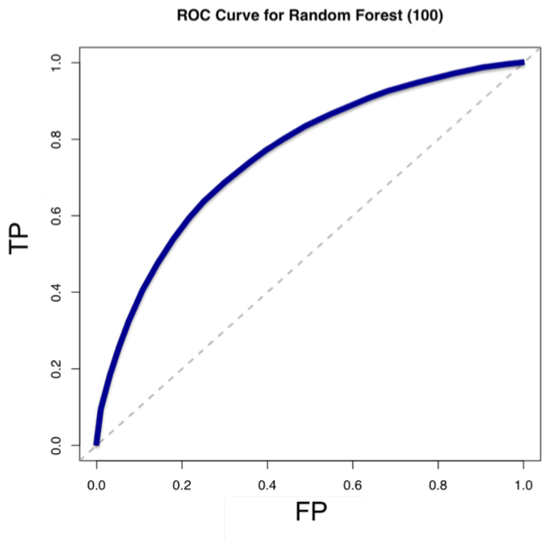
| | | **Predicted** | |
|---|---|---|---|
| | | 0 | 1 |
| **True** | 0 | 35936 | 1241 |
| | 1 | 9038 | 2196 |

Table 9: Tree = 300

| | | **Predicted** | |
|---|---|---|---|
| | | 0 | 1 |
| **True** | 0 | 36018 | 1173 |
| | 1 | 9005 | 2215 |

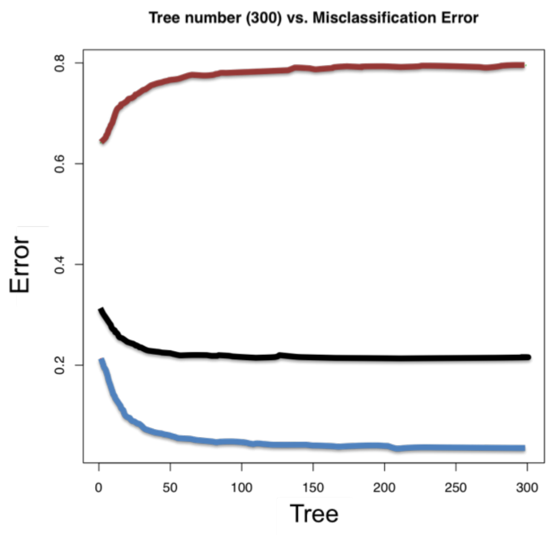


(A) Error vs Number of trees

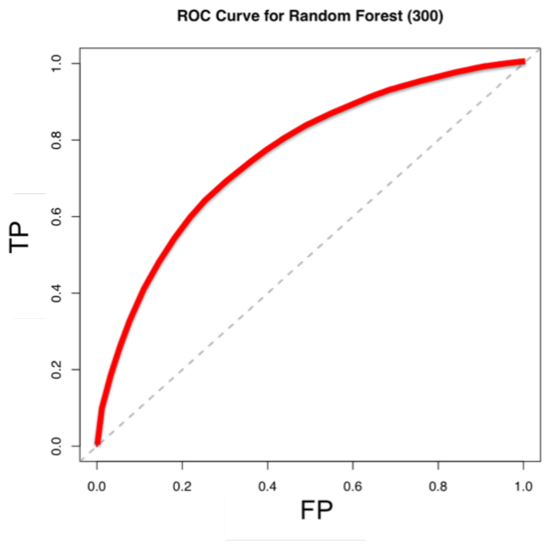


(B) ROC curve

Figure 36: Tree = 100
(TP: True Positive Rate; FP: False Positive Rate)



(A) Error vs Number of trees



(B) ROC curve

Figure 37: Tree = 300
(TP: True Positive Rate; FP: False Positive Rate)

In sum, random forest has many advantages when deal with Springleaf dataset. It can do automatic variable selection, handle missing values, and be robust to noise. The results based on random forest are very stable in general. It performs very well in Springleaf case. The major weakness is that it takes long time when tune parameter. So we consider to use a fast-speed ensemble learning method XGBoost, which will be introduced in next section.

### 7.1.3 Result based on MNIST Dataset

Ensemble methods like random forest has been used on MNIST data set. We have used this method with a variant number of trees and on deskewed data. The best results were obtained with 500 trees. The following table gives the results.

Table 10: Tree = 500

| Preprocessing | # of Candidate Variables per Split | Error Rate |
|---|---|---|
| Deskewing | 60 | 2.18% |
| None | 50 | 2.61% |
| None | 60 | 2.66% |
| None | default | 2.73% |

Random forest method did not give MNIST data as much success as SVM or $k$-means method.

## 7.2 XGBoost

XGBoost is short for "Extreme Gradient Boosting" and is an implementation of the gradient boosting algorithm. Due to its extremely fast speed and scalability, XGBoost is a very popular tool in Kaggle competitions. It is used for supervised learning problems, where the training data $x_i$ is used to predict the label of new data $y_i$.

### 7.2.1 Algorithm

**Objective Function**

The objective function of XGBoost contains two parts: training loss and regularization:

$$Obj(\Theta) = L + \Omega$$

where L is the training loss function and $\Omega$ is the regularization term. The training loss measures how predictive our model is on training data. A commonly used training loss is mean squared error. The regularization term controls the complexity of the model to avoid overfitting.

**Tree Ensemble Model** The model of XGBoost is a tree ensemble. The tree ensemble model is a set of classification and regression trees (CART). We classify each point into different leaves, and assign them the score on corresponding leaf. A CART is a bit different from decision trees, where the leaf only contains decision values. In CART, a real score is associated with each of the leaves, which gives richer interpretations that go beyond classification. This also makes the unified optimization step easier.

Usually, a single tree is not strong enough to be used in practice. What is actually used is the so-called tree ensemble model that sums the prediction of multiple trees together. The new trees try to complement the old trees. Mathematically, the model is write as:

$$\hat{y_i} = \sum_{k=1}^{K} f_k(x_i), f_k \in F \tag{27}$$

where K is the number of trees, $f_k$ is a function in the functional space $F$, and $F$ is the set of all possible CARTs. Therefore the objective functions to optimize can be written as:

$$Obj(\Theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{28}$$

**Additive Training**

XGBoost uses an additive strategy to train each tree instead of training all the trees at once. It adds one new tree at a time to fix the already learned ones. The prediction value at step t is $\hat{y}_i^{(t)}$, we have:

$\hat{y}_i^{(0)} = 0$
$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$
$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$
...
$\hat{y}_i^{(t)} = f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$

A new tree is added at each step to optimize the objective:

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \end{aligned}$$

If considering MSE as loss function, it becomes the following form.

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + constant \end{aligned}$$

The form of MSE is friendly, with a first order term (usually called residual) and a quadratic term. For other losses of interest (for example, logistic loss), it is not so easy to get such a nice form. So in the general case, we take the Taylor expansion of the loss function up to the second order:

$$Obj^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant \tag{29}$$

where the $g_i$ and $h_i$ are defined as:

$$\begin{aligned} g_i &= \partial_{\hat{y}_i^{(t)}} l(y_i, \hat{y}_i^{(t-1)}) \\ h_i &= \partial^2_{\hat{y}_i^{(t)}} l(y_i, \hat{y}_i^{(t-1)}) \end{aligned}$$

After we remove all the constants, the specific objective at step $t$ becomes the following equations (30). This becomes our optimization goal for the new tree. One important advantage of this definition is that it only depends on $g_i$ and $h_i$. This is how xgboost can support custom loss functions. We can optimize every loss function, including logistic regression and weighted logistic regression, using the exactly the same solver that takes $g_i$ and $h_i$ as input:

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{30}$$

**Model Complexity**

Another important thing about the above objective function is the regularizationthe complexity of the tree $\Omega(f)$. First, refine the definition of a tree $f(x)$ as:

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \to \{1, 2...T\}$$

where w is the vector of scores on leaves, q is a function assigning each data point to the corresponding leaf and T is the number of leaves. In XGBoost, the complexity is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

**Structure Score**

After reformalizing the tree model, we can write the objective value with the t-th tree as:

$$Obj^{(t)} \approx \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 =$$
$$\sum_{j=1}^{T}(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T$$

where $I_j = \{i|q(x_i) = j\}$ is the set of indices of data points assigned to the $j_{th}$ leaf. Notice that in the second line we have changed the index of the summation because all the data points on the same leaf get the same score. We could further compress the expression by defining $G_j = \sum_{i\in I_j} g_i$ and $H_j = \sum_{i\in I_j} h_i$:

$$Obj^{(t)} = \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T \tag{31}$$

In this equation $w_j$ are independent of each other, the form $G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2$ is quadratic. The best $w_j$ for a given structure q(x) and the best objective reduction we can get is as equation (32) and (33). The last equation measures how good a tree structure $q(x)$ is.

$$w_j^* = -\frac{G_j}{H_j + \lambda} \tag{32}$$

$$Obj^* = -\frac{1}{2}\sum_{i=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T \tag{33}$$

**Learn the tree structure**

Now that we have a way to measure how good a tree is, ideally we would enumerate all possible trees and pick the best one. In practice it is intractable, so we will try to optimize one level of the tree at a time. Specifically we try to split a leaf into two leaves, and the score it gains is:

$$Gain = \frac{1}{2}[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_l + G_R)^2}{H_L + H_R + \lambda}] - \gamma \tag{34}$$

This formula can be decomposed as 1) the score on the new left leaf 2) the score on the new right leaf 3) The score on the original leaf 4) regularization on the additional leaf. We can see an important fact here: if the gain is smaller than $\gamma$, we would do better not to add that branch. This is exactly the pruning techniques in tree based models.

### 7.2.2 MNIST Results

Xgboost was applied to the MNIST data set giving the smallest result of 1.77% error, without using PCA and using the corresponding R packages. Parameter settings were, for example,

```
"objective" = "multi:softmax","eval_metric" = "mlogloss", "num_class" = 10, "eta" = 0.2, "cols
```

where the last parameters mostly influenced the error. We also noticed that using PCA increased error rather than decreased it and was no longer pursued. After deskewing the data, our error was reduced to 1.41% by using the parameters.

```
param = list("objective" = "multi:softmax","eval_metric" = "merror", "num_class" = 10,
             "eta" = 0.18, "colsample_bytree"= 0.45,"subsample" = 3, "verbose"= 0, "seed" = 10)
```

After comparison with other machine learning methods we decided to focus on other algorithms that had more promising results for the MNIST dataset and further analysis was discontinued. Sample code and more results with a few variations in the parameters are found in the code section of this paper.

### 7.2.3 Result on Springleaf Dataset

Figure 38 show the error rate tendency of XGBoost. The horizontal axis is the number of trees and the vertical axis is the error rate of prediction. The points in the lower curve and upper curve are the error rates for predicting training data and test data separately. As the new trees in XGBoost is to complement the old trees, we can see that the error rates of prediction decreased with the number of trees increasing.

In the parameter tuning in XGBoost, we are very careful about the over fitting problem. The error rate of prediction for test data and training data decreased with the number of trees. As we increase the number of trees, the error rate of training data would keep decreasing while error rate of test data would be flat as shown in Figure 38. This would imply that more than 1000 trees will make the model over fitted.

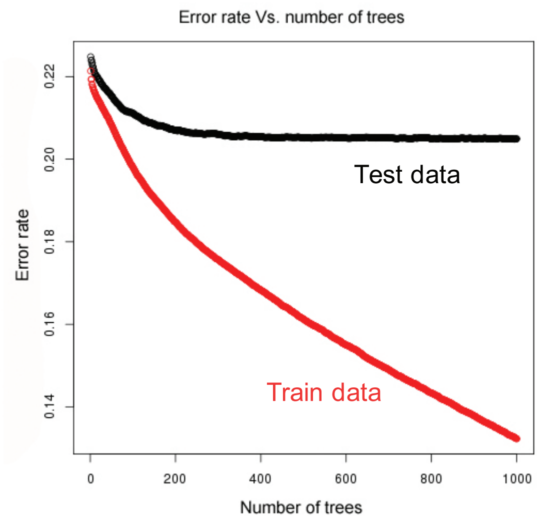


Figure 38: Error rate of prediction for test and training data

After tuning the parameters in XGBoost, the best accuracy we got is 80.0%. The accuracy for target=0 is 96.06% and the accuracy for target=1 is 26.78%. The confusion matrix is shown in table 11.

Table 11: Confusion Matrix

| | | **True** | | **Accuracy** |
|---|---|---|---|---|
| | | 0 | 1 | |
| **Predicted** | 0 | 35744 | 8201 | 96.06% |
| | 1 | 1467 | 2999 | 26.78% |

We also get the AUC(the area under ROC curve) plot shown in Figure 39. As we mentioned before, the larger the area under ROC curve is, the more accurate it would be. The points in the upper curve and lower curve are the AUC for predicting the training data and test data separately. As we can see, the AUC of both test data and training data increased with the number of trees. The largest area under the curve based on XGBoost is 0.7901 which is the best result among all the classifiers we tried before. It implies that XGBoost which can naturally handle both categorical and numerical variables are very suitable in our case.

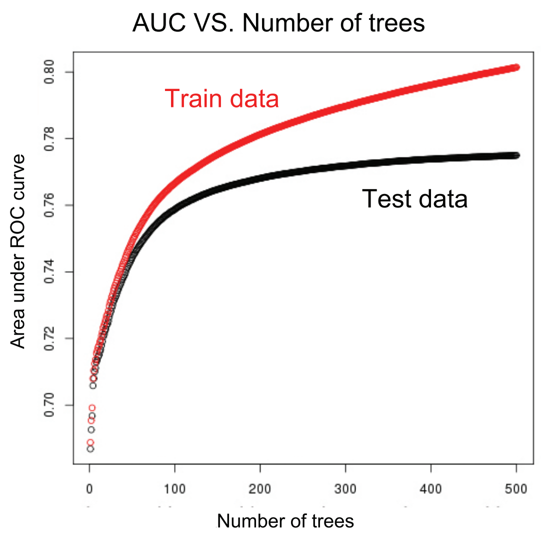


Figure 39: AUC for predicting test and training data

## 7.3 Stacking

Stacking normally is used to combine models of different types. The procedure is as follows: split the training set into two disjoint sets. Train several base classifiers on the first set. Test the base learners on the second set. In the third set, using the predictions from base classifiers as the inputs, and the correct responses as the outputs, train a higher level classifier which is also called Meta-classifier. Note that steps 1 to 3 are the same as cross-validation, but instead of using a winner-takes-all approach, we combine the base classifiers, possible nonlinearly, to achieve the highest generalization accuracy.

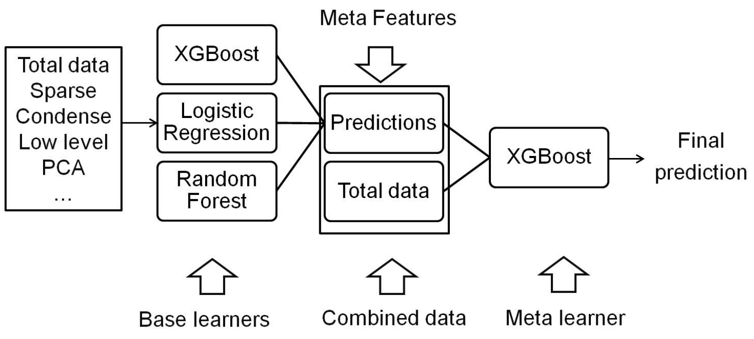


Figure 40: stacking generation

In the Springleaf case, the total stacking model is shown in Figure 40. The entire training data set is divided into several different blocks to match different classifiers and introduce more diversity into stacking model. For example, the low level features are more likely to be categorical features, so we train them with XGBoost or logistic regression which can handle categorical variables. As for the high level features, we suppose they are continuous variables, SVM or KNN seem to be more suitable for them. We also have other blocking methods to construct the subsets of training data, like sparse and condense data which are distinguished by how many zeros are in the features. Sparse data is the subset of training data with all the features whose most frequent elements are the default value 0 and null. On the contrary, features whose elements are meaningful values are condensed data. The underlying idea of the separation of sparse data and condense data is to avoid the former ones being overwhelmed by latter ones.

We use the total data set and the above subsets as the input to train multiple classifiers we introduced before. The classifiers with the most accurate predictions are chosen as the base classifiers. As shown before, XGBoost, Logistic regression and random forest have better performance than other models. They are the base classifiers in our stacking model. The results from the base model is shown in table12.

Table 12: Results of base models

| **Base model generation** | **Accuracy** | **Accuracy (target=1)** | **AUC** |
|---|---|---|---|
| XGB + total data | 80.0% | 28.5% | 79.1% |
| XGB + condense data | 79.5% | 27.9% | 78.9% |
| XGB + Low level data | 79.5% | 27.7% | 78.4% |
| Logistic regression+ sparse data | 78.2% | 26.8% | 77.6 % |
| Logistic regression+ condense data | 79.1% | 28.1% | 77.9% |
| Random forest + PCA | 77.6% | 20.9% | 76.4% |

The outputs of these base classifiers, along with the actual correct labels constitute the training dataset for the meta-classifier which is XGBoost classifier. We also tried some unsupervised approaches like applying kmeans or average on the outputs of base classifiers to cluster the dataset into two groups. However, they did not give us better result than using XGBoost as a meta-classifier. The final results of the meta classifier are shown in Table 13. After stacking, the accuracy increased by 1% comparing to the best result of base classifiers. It is a significant improvement for the Kaggle competition.

Table 13: Results of meta models

| Base model generation | Accuracy | Accuracy (target=1) | AUC |
|---|---|---|---|
| XGB | 81.11 % | 29.21% | 79.6% |
| Averaging | 79.44% | 27.31% | 77.2% |
| Kmeans | 77.45% | 23.91% | - |

## 7.4 Neural networks

### 7.4.1 Introduction of Artificial Neural Network

Neural networks, also known as Artificial Neural Network (ANN), is a popular information processing method in data mining and machine learning areas. It is inspired by biological nervous systems and is able to learn by example, like humans. ANN provides the best solutions to many problems, such as pattern recognition, data classification and language processing.
The following figure shows a the structure of ANN

1. The leftmost layer($x_n$) inputs features, and the rightmost layer($y_n$) outputs results for the user. The layer(s) between them are named hidden layer(s).

2. The solid circles represent neurons, which process inputs from previous layer and output results for next layer (or the user).

3. The lines connect neurons and usually are assigned weight.

The network may have more than 1 hidden layer (called a deep network). Deep network is currently a very hot research field in the deep learning area.

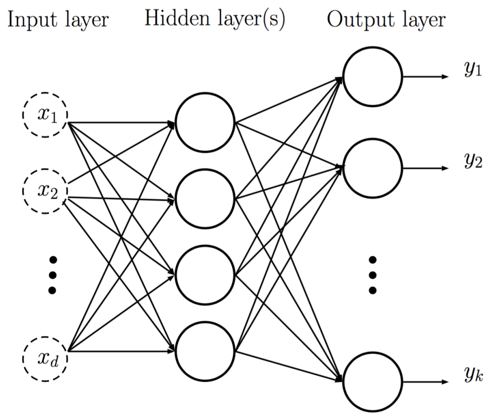


Figure 41: A mathematical model of Artificial Neural Network

### 7.4.2 What is a biological neuron?

Neurons, also known as nerve cells, are special cells that process and transmit information by electrical signaling. The picture below shows human brain nerve cells. A neuron connects to other neurons to form a network. In fact, human brain has around $10^{11}$ neurons, and each of them may be connected up to 10,000 others.

A single nerve cell is made up of four parts, which are soma (body of the neuron), dendrites, axon and axon terminal. The dendrites of a neuron are cellular extensions with many branches;

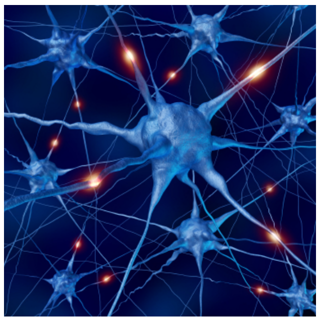


Figure 42: Biological neurons from the human brain)

these are the majority of input to the neuron. The axon's main function is to carry signals away from the soma.

### 7.4.3 Artificial Neurons and Activation Functions

When the human brain is simulated, each of the human brain neural cell connects with many each other cells and works independently like a small project itself. At the same time, they exchange information one by one and sum up their analysis results to make decisions. ANN algorithm works in a similar way by running a simulation of a bunch of heavily interconnected little mini-programs, called neurons.

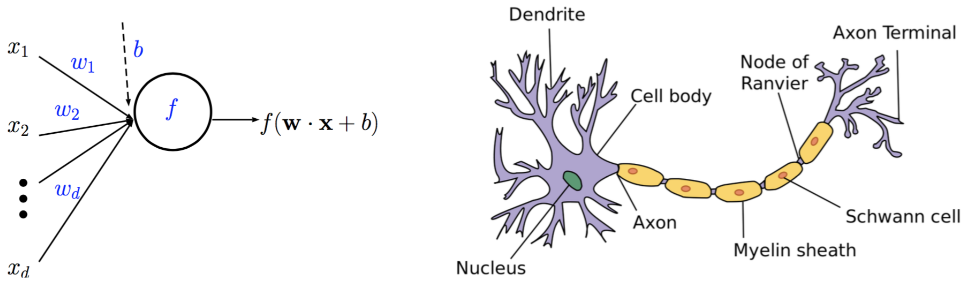


Figure 43: Artificial Neuron

Artificial neurons are mathematical functions from $R^d \to R$ defined by wights $(w_i)$, bias $(b)$ and $f$ (rule). These rules are called activation functions. Actually, ANN is a composition of many functions. It has been proved that every continuous function from input to output can be implemented with 1 hidden layer (containing enough hidden units) and proper nonlinear activation functions.
There are several popular activation functions:

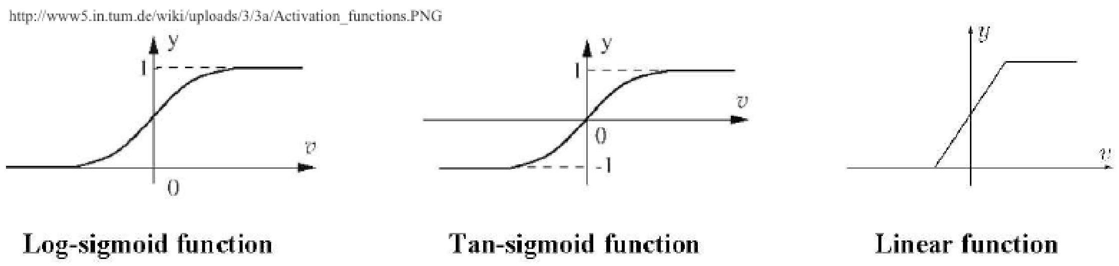


Figure 44: Activation Functions

As the picture shows, input data enter from left side and has some operations performed on first layer of neurons. The simplest ANN model just include input layer, one hidden layer and output layer, but most deep learning models has more hidden layers. The number of layers depends on your data set and your specific application.

### 7.4.4 How to train ANNs

In a real world application, if we want to solve more complex problem, one layer model is not enough. Multilayer Neural Network models are introduced. Multilayer perceptrons (MLP) is one of them.
First, we select an activation function for all neurons. Second, we initialize weights $w$ and bias $b$. Then, in order to get a good prediction, we need to tune the weights and bias. However, since each perceptron has discrete behavior, making it an effect latter layers hard to predict. As we mentioned before, there are several different activation functions. One of them is Sigmoid, which is the smoothed-out version of the perceptron.

$$f(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

This function can reduce the affect of the value of $w$ and $b$. In other words, when we tune these parameters, a “small” change can cause a slice difference in the output.

These are notations of sigmoid neuron network:

Matrix of all weights:

$$\mathbf{W}^l = (w^l_{jk})_{j,k}; \qquad (w^l_{jk} \text{ is layer } l,\ j \text{ back to } k \text{ weight.})$$

Vector of biases in layer $l$:

$$\mathbf{b}^l = b(b^l_j)_j; \qquad (b^l_j \text{ is layer } l, \text{ neuron } j \text{ bias.})$$

Vector of outputs from neurons in layer $l$

$$\mathbf{a}^l = (a^l_j)_j; \qquad (a^l_j \text{ is layer } l, \text{ neuron } j \text{ output})$$

Vector of weighted inputs to neuron in layer $l$:

$$\mathbf{z}^l = (z^l_j)_j; \qquad (z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j;\ \text{weighted input to neuron } j \text{ in layer } l)$$

From the above, we have

$$\text{Input layer is } \mathbf{a}^0 = \mathbf{x}; \qquad \text{and the output is } \mathbf{a}^L$$

For each $1 \leq l \leq L$:

$$\mathbf{a}^l = \sigma(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l) = \sigma \mathbf{z}^l$$

The square **loss function** depends on it weights and biases:

$$C(\{\mathbf{W}^l, \mathbf{b}^l\}_{1 \leq l \leq L}) = \frac{1}{2n} \sum_{i=1}^{n} ||\mathbf{a}^L(\mathbf{x}_i) - \mathbf{y}_i||^2$$

To simplify, we consider $n = 1$:

$$C(\{\mathbf{W}^l, \mathbf{b}^l\}_{1 \leq l \leq L}) = \frac{1}{2} \sum_{i=1}^{n} ||\mathbf{a}^L(\mathbf{x}_i) - \mathbf{y}_i||^2 = \sum_j (a^L_j - y_i(j))^2$$

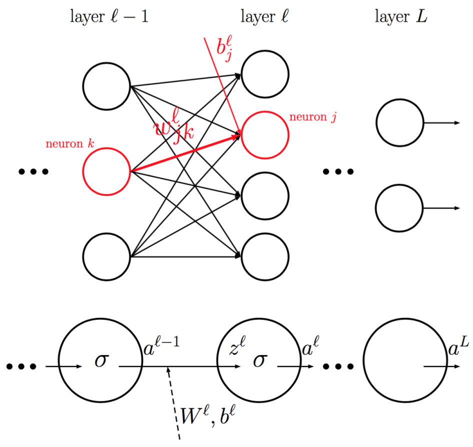


Figure 45: Sigmoid Neuron Network

Similar to the previous method, we still use gradient decent to tune the weight and bias. We need to compute $\frac{\partial C_i}{\partial w_{jk}^L}$ and $\frac{\partial C_i}{\partial b_j^L}$ first. By the chain rule we have:

$$\frac{\partial C_i}{\partial w_{jk}^L} = \frac{\partial C_i}{\partial a_j^L}\frac{\partial a_j^L}{\partial w_{jk}^L} = (a_j^L - y_i(j))\frac{\partial a_j^L}{\partial a_j^L}\frac{\partial z_j^L}{\partial w_{jk}^L} = (a_j^L - y_i(j))\sigma'(z_j^L)a_k^{L-1}$$

And we have

$$a_j^L = \sigma(\sum_{k'} w_{jk'}^L a_{k'}^{L-1} + b_j^L) = \sigma z_j^L$$

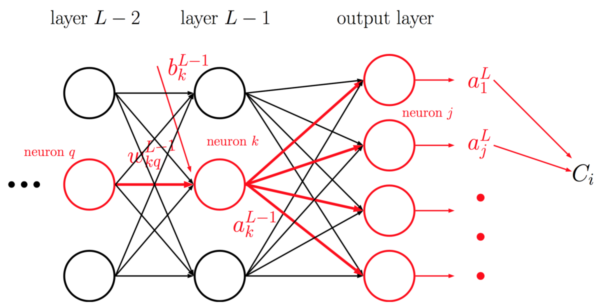


Figure 46: Computation for the output layer

Similarly, we obtain that

$$\frac{\partial C_i}{\partial b_j^L} = \frac{\partial C_i}{\partial a_j^L}\frac{\partial a_j^L}{\partial b_j^L} = (a_j^L - y_i(j))\sigma'(z_j^L)$$

The rate of change of $C_i$ with the respect to $w_{jk}^L$ depends on the following three factors:

$$a_j^L - y_i(j), \qquad \sigma'(z_j^L), \qquad a_k^{L-1}$$

After computing L layer, we continue with L -1 layer:

$$\frac{\partial C_i}{\partial w_{kq}^{L-1}} = \sum_j \frac{\partial C_i}{\partial a_j^L}\frac{\partial a_j^L}{\partial w_{kq}^{L-1}} = \sum_j \frac{\partial C_i}{\partial a_j^L}\frac{\partial a_j^L}{\partial a_k^{L-1}}\frac{\partial a_k^{L-1}}{\partial w_{kq}^{L-1}}$$

Where,

$\frac{\partial C_i}{\partial a_j^L}$ is known; $\frac{\partial a_j^L}{\partial a_k^{L-1}}$ is link between layers L and L-1; $\frac{a_k^{L-1}}{\partial w_{kq}^{L-1}}$ similarly as output layer

The same as L -2 and more further inside links between layers:

$$\frac{\partial C_i}{\partial w_{qr}^l} = \sum_{p,\dots,k,j} \frac{\partial a_q^l}{\partial w_{pq}^l} \frac{\partial a_p^{l+1}}{\partial a_q^l} \cdots \frac{a_j^L}{\partial a_k^{L-1}} \frac{\partial C_i}{\partial a_j^L}$$

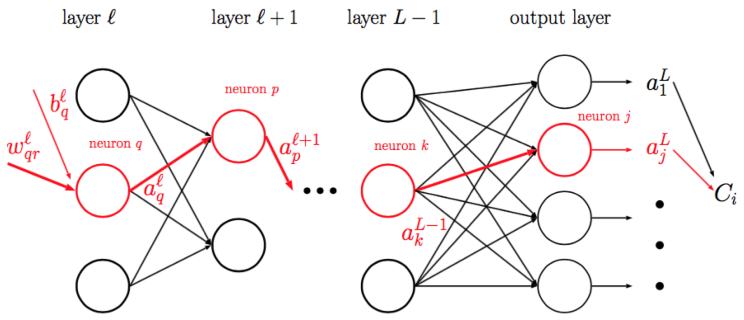


Figure 47: Computation of layer L-1

Then, using the backpropagation algorithm, we have:

$$\frac{\partial a_j^L}{\partial a_q^l} = \sum_{p,\dots,k} \frac{\partial a_p^{l+1}}{\partial a_q^l} \cdots \frac{\partial a_j^L}{\partial a_k^{L-1}} \qquad \text{for } l = L, \dots, 1$$

$$=> \frac{\partial C_i}{\partial w_{qr}^l} = \sum_j \frac{\partial a_q^l}{\partial w_{qr}^l} \cdot \frac{\partial a_j^L}{\partial a_q^l} \cdot \frac{\partial C_i}{\partial a_j^L} \qquad \frac{\partial C_i}{\partial w_{qr}^l} = \sum_j \frac{\partial a_q^l}{\partial b_q^l} \cdot \frac{\partial a_j^L}{\partial a_q^l} \cdot \frac{\partial C_i}{\partial a_j^L}$$

After initializing all the weights $w_{jk}^l$ and biased $b_j^l$, we use above formulas to compute $\frac{\partial C_i}{\partial w_{qr}^l}, \frac{\partial C_i}{\partial w_{qr}^l}$ of every layer and every neuron.
Now, we can update the weights and biases by:

$$w_{jk}^l \leftarrow w_{jk}^l - \eta \cdot \frac{\partial C_i}{\partial w_{qr}^l}, \qquad b_j^l \leftarrow b_j^l - \eta \cdot \frac{\partial C_i}{\partial w_{qr}^l}$$

These steps are repeated until convergence.
We can implement these functions in python; for more detail information, please check the reference link at the end of this report.

### 7.4.5 Our experiments and result

**The MNIST Digits Recognition Problem:** The following table shows parameters group 1 used and the error rate.

| | Cross-Val : Train Rate | Error Rate | Layers and Best Parameters |
|---|---|---|---|
| Set 1 | 1:3 | 1.29% | one layer<br>20x[500,250,100,50] |
| Set 2 | 1:4 | 1.4% | two layers<br>10x [100,50]<br>10x [300,150,50] |

**The Springleaf Marketing Response Problem:** We tried three Neural Network R Packages:

- **neuralnet:** Training of neural networks using backpropagation, resilient backpropagation with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993) or the modified globally convergent version by Anastasiadis et al. (2005). nnet
- **nnet:** Software for feed-forward neural networks with a single hidden layer, and for multinomial log-linear models.
- **RSNNS :** Neural Networks in R using the Stuttgart Neural Network Simulator (SNNS)

After comparing the three packages, only "neuralnet" worked for our project. We have tried our whole training data set as input data, but we did not get output after running the data set for 10 days. So we just use 4000 observations to train model and pick several two layers matrix. However, the result is not as good as we expected. Sometimes, it cannot predict target 1 and assign all the data as group 0. So we did not use these result.

A large fraction of this section is written based on Dr. Chen's slides at
*http://www.math.sjsu.edu/˜gchen/Math285S16/lec9neuralnets.pdf*
The reader is referred to the above url for further details.

## 8 MNIST Experiments

During the course of the project we experimented with different ways of combining dimension reduction and classification. In particular we explore a technique we call "local dimension reduction" where instead of a applying dimension reduction technique such as PCA to the entire dataset before training our classifiers, apply localized versions of PCA to different classes or pairs of classes and train classifiers in the reduced space.

We also explore ways to extend LDA using features corresponding to zero eigenvectors.

### 8.1 Local One-vs-Rest PCA

In this section we present a method of combined PCA with kernel SVM. Previous studies apply PCA on all training data and classify the data in the PCA space with kernel SVM. We will name this method as "global PCA" the dimension reduction is applied to the entire dataset. The procedure is straightforward. First apply PCA on the whole training data set and choose a certain number of principle components. Then we train our classifier in the reduced space. For classification, the test data needs to be projected into the reduced space and classified there.

We propose applying dimension reduction to each class separately. We first apply PCA to the points of a single class $i$. Then the rest of the training set is projected into the reduced space and a classifier $f_{i,\neg i}$ is trained. Classification of a new point $x$ is done by projecting it into each of the $N$ subspace where its image $y = W_i(x)$ is classified.

We give a pseudo-algorithm for the one-vs-rest dimension reduction procedure for the unsupervised case.

1. center $X$ about the $i$th mean $\mu_i$
2. find dimension reduction $W_i$
3. train $f_i$ on the set $Y_i = W_i(X)$

Where $f_{i,\neg i}$ binary classifier for class $i$ and not $i$. To classify a new point $x$, for each class $i$

1. $x_c = x - \mu_i$
2. $y = W_i(x_c)$
3. $c_i(x) = f_{i,\neg i}(y)$

classification of $x$ is done according to rule (2). For our purposes, the classifier used is a kernel SVM.

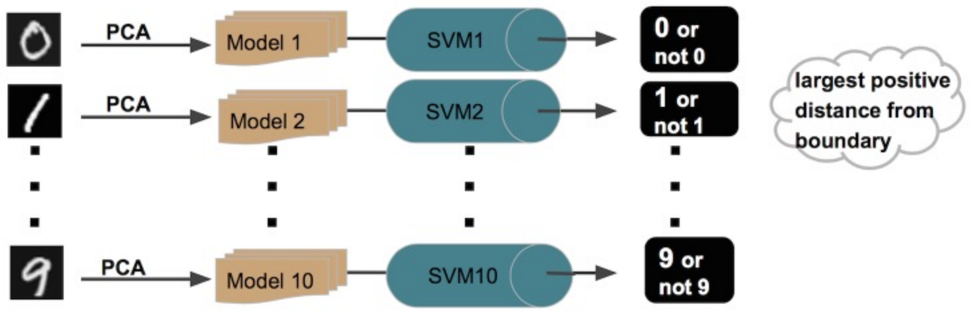


Figure 48: Local PCA with one-vs-rest SVM method

Below is a comparison between the one-vs-rest or "classwise" PCA to the "global" PCA. Overall the global application of PCA outperforms the classwise application in our preliminary experiments.

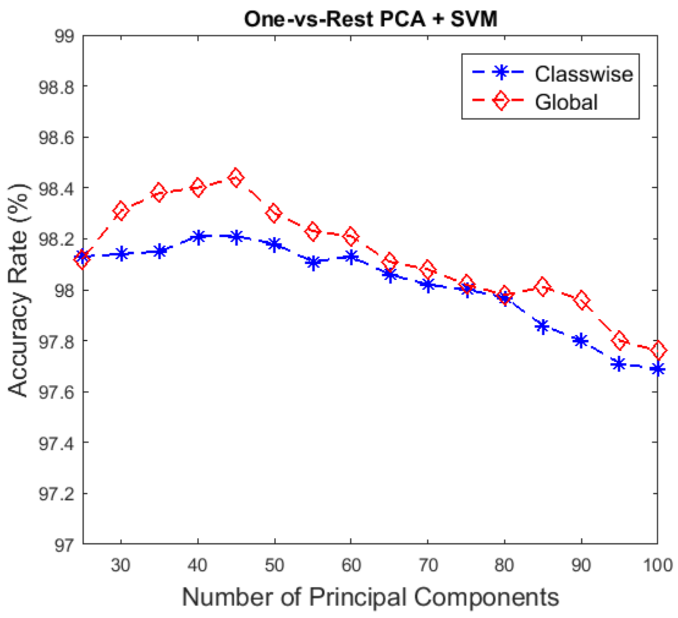


Figure 49: One-vs-rest SVM with Local PCA

## 8.2 Local Pairwise Dimension Reduction

To construct a pairwise classifier using this method, we first apply a dimension reduction technique to the subset of data containing only the points of classes $i$ and $j$. Then a classifier $f_{ij}$ is trained on the reduced subset. This is done for all pairs $i$,$j$. A new point is classified by projecting it into each of the $\binom{N}{2}$ subspaces and its images are classified by rule (2).

Below give a pseudo-algorithm for the pairwise dimension reduction procedure. Let $X_{ij}$ be the subset of $X$ corresponding to classes $i$ and $j$ and $\mu_{ij}$ its centroid. Then, for each pair $ij$,

1. center $X_{ij}$ about the mean $\mu_{ij}$
2. find dimension reduction $W_{ij}$
3. train $f_{ij}$ on the set $Y_{ij} = W_{ij}(X_{ij})$

Classification of new data is done by the following. Let $x$ be a new point. Then for each pair $i$ and $j$,

1. $x_c = x - \mu_{ij}$
2. $y = W_{ij}(x_c)$
3. $c_{ij}(x) = f_{ij}(y)$

classification of $x$ is done according to rule (2).

In this experiment we try pairwise SVM using NDA and LDA with a baseline case (i.e. $W_{ij} = I$). A kernel SVM was used with parameter $\gamma = \frac{1}{\sqrt{d}}$, which is the square root of the default parameter setting in LIBSVM. The data was first reduced to 300 dimensions using PCA. The pairwise transformations were then applied over a range of dimensions between 2 and 10. Results converge as $d$ grows larger and approaches $m$, the number of dimensions of the full dataset.

As can be seen from the plot, the two discriminant transformations worsen classification accuracy as compared to no pairwise transformation (i.e. $W_{ij} = I$ in 3.1.1). The baseline error rate exceeds the error rates of NDA and LDA for $d < 10$ and $d < 7$, respectively, where the error rate increases to over 50%. It is interesting to note that the behavior of the LDA error rate mirrors the behavior of the simple SVM curve, while the NDA error rate decreases linearly until $d < 10$. Also

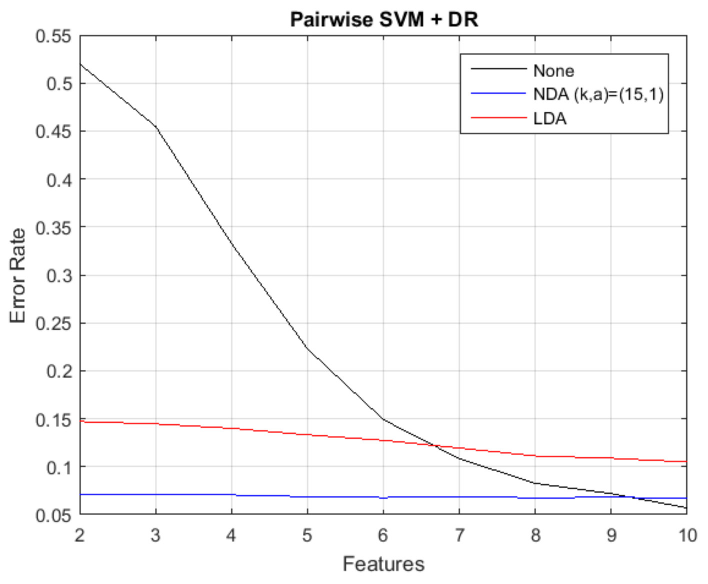


Figure 50: Pairwise Dimension Reduction

noteworthy is the noticeable increase in the error rate at low dimensions exhibited by LDA and the base case. NDA appears to preserve classification features at lower dimensions better than LDA.

Figure 51 compares pairwise PCA with global PCA. We can see that we gain a slight improvement in accuracy with the pairwise method using between 30 and 45 principle components.

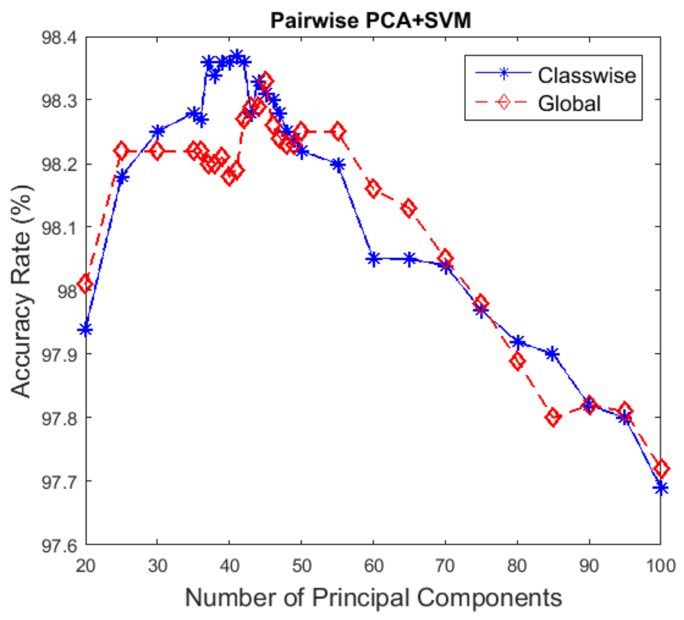


Figure 51: Pairwise Dimension Reduction

## 8.3 LDA Extension

As mentioned in Section 1.2, LDA has at most $c-1$ nonzero (real part) eigenvalues which can be used for feature extraction. The obvious question is whether information useful for classification is contained in the features corresponding to the zero eigenvalues. Another related issue is the possible existence of complex eigenvalues and how to handle them. In both cases the question of how to select among the features $y = w^T X$ where $w$ is a zero-eigenvector of $S_w^{-1} S_b$ must be answered.

We experiment sorting these features in order of ascending variance $Var(y)$.

First, we sort all features corresponding to eigenvalues with zero real parts by increasing variance. Next, we front-load the eigenvalues with nonzero complex parts followed by variance-sorted zero-eigenvalues. These are compared to the unsorted features as extracted from the original transformation.

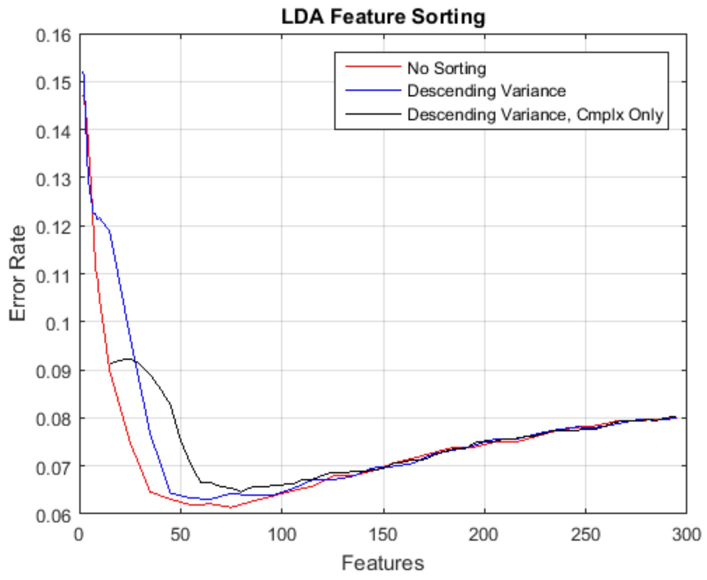


Figure 52: Pairwise SVM + LDA Extension

# 9 Summary of Results

## 9.1 Summary of Results

This section presents the results of all the methods as well as the combination of methods, that have been tried out in the MNIST digit recognition project. We will start with the instance based methods, namely, k Nearest Neighbors and k Means. This will be followed by support vector machine methods, neural networks and finally some ensemble methods.

### 9.1.1 k-Nearest Neighbors

| **Method Detail** | **Preprocessing Method** | **Best Parameters** | **Error Rate** |
|---|---|---|---|
| Weighted kNearest Neighbor with weighted sum rule | None | $k = 8$ | 2.76% |
| kNearest Neighbor by majority Voting | None | $k = 3$ | 2.95% |
| Matrix kNearest Neighbor with 2D LDA | 2D LDA | $k = 5$, L2 marix norm | 2.96% |

### 9.1.2 Local k Means

| **Method Detail** | **Preprocessing Method** | **Best Parameters** | **Error Rate** |
|---|---|---|---|
| local k Means with deskewing and oneVsAll PCA | Deskewing and oneVsAll PCA,dimension=150 | $k = 10$ | 1.14% |
| local k Means on deskewed data | Deskewing | $k = 10$ | 1.17% |
| local k Means after local PCA | local PCA, dimension=150 | $k = 10$ | 1.53% |
| local k Means | None | $k = 14$ | 1.75% |
| local k Means after PCA | Global PCA,dimension=50 | $k = 2$ | 2.19% |

### 9.1.3 Support Vector Machine

| Method Detail | Preprocessing Method | Best Parameters | Error Rate |
|---|---|---|---|
| Pairwise SVM after PCA on deskewed data | Deskewing + PCA, dimension=60 | $RBFkernel, \gamma = 0.034, c = 3$ | 0.94% |
| Pairwise SVM after PCA on deskewed data | Deskewing + PCA, dimension=45 | $RBFkernel, \gamma = 0.056, c = 2$ | 0.97% |
| OneVsAll SVM after OneVsAll PCA on deskewed data | Deskewing + OneVsAll PCA, dimension=55 | $RBFkernel, \gamma = 0.034, c = 5$ | 1.0% |
| OneVsAll SVM after OneVsAll PCA | OneVsAll PCA, dimension=55 | $RBFkernel, \gamma = 0.034, c = 5$ | 1.2% |
| Pairwise SVM after Global PCA | Global PCA, dimension=45 | $RBFkernel, \gamma = 0.036, c = 3$ | 1.23% |
| Pairwise SVM after Global PCA | Global PCA, dimension=57/58 | $RBFkernel, \gamma = 0.05, c = 2$ | 1.25% |
| Pairwise SVM after pairwise PCA | Pairwise PCA, dimension=50 | $RBFkernel, \gamma = 0.02, c = 2$ | 1.31% |
| SVM after Global PCA and pairwise NDA | Global PCA + NDA, d1=300, d2=45 | $RBFkernel, \gamma = 0.04, c = 5$ | 1.38% |
| Pairwise SVM | None | $RBFkernel, \gamma = 0.034, c = 5$ | 1.4% |
| SVM with global tSNE | Global tSNE | $d = 3, pcad = 30, perp = 30, theta = 0.5, dimension = 300$ | 3.0% |
| Global linear SVM with parametric tSNE | Parametric tSNE | $layer[600, 600, 1500], perp = 30, dimension = 300$ | 3.4% |

### 9.1.4 Neural Networks

| Method Detail | Preprocessing Method | Best Parameters | Error Rate |
|---|---|---|---|
| Neural Network Ensemble on deskewed data | Deskewing + Neural Network Ensemble | 20x[500,250,100,50], $crossentropy$, (cross-val, train)=(.25,.75) | 1.29% |
| Neural Network Ensemble on deskewed data | Deskewing + Neural Network Ensemble | 10x [100,50] and 10x [300,150,50], cross entropy, (cross-val, train) = (0.2,0.8) | 1.41% |

### 9.1.5 Combined Methods

In this section results of two classification methods have been combined. The data set has been deskewed and dimension of the data has been reduced to 60 using PCA. The pre-processed data set then has been used to predict the digits with SVM algorithm as well as local k means algorithm. Henceforth, using the Bayes Rule we obtain the method that maximizes the conditional probability of a correct decision. By this we mean, we consider the method with higher $P(X = i|\hat{X} = i), i = 0, 1...9$. Having chosen the method that maximizes the probability, the prediction is the digit that has highest probability of being the prediction by that method.
With this brief description, we tabulate the results of our combination method where we combined the two methods that have been most successful for our data set, namely, local k-means and SVM.

| Method Detail | Preprocessing Method | Best Parameters | Error Rate |
|---|---|---|---|
| SVM and local k means deskewed data after PCA | Deskewing + PCA, dimension=60 | k=10, RBF kernel, $\gamma = 0.034$, c=3 | 0.97% |

### 9.2 Springleaf Marketing Response

Springleaf is a specific data mining project in the real world. What we face is not the well designed and clean data in the textbook, but huge and noisy data with a lot of missing values and mixed type of features. We need to determine inaccurate, incomplete data and improve the data quality before data analysis. In feature engineering process, some feature encoding like adding date difference gives us an impressive promotion. As for dealing with the missing value, we tried three ways. It turns out that filling the missing value with median value has better performance than filling that with mean value or randomly replacing missing value according to the value distribution in the feature. After data pre-processing, we tried various classification techniques on the cleaned data. The results show classifiers which based on distance like SVM, KNN are not suitable for our case, however, tree model or logistic regression which can naturally handle variables of mixed type are preferred here. This may imply that categorical variables are dominant in our data set. To further improve the result, we applied stacking method to ensemble six base models with different classifiers and different subsets of data. The stacking method gives us additional 1% improvement on accuracy. We believe that a crucial point to the success of stacking is feature diversity. Figure 53 shows the result summary of all the classifiers. Overall, our most significant improvement came from model selection, parameter tuning and Stacking.

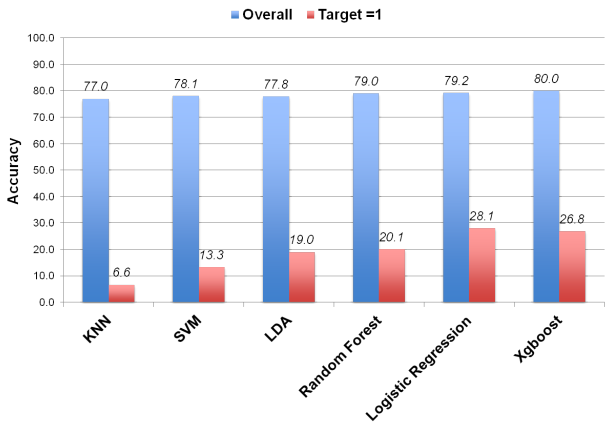


Figure 53: Methods comparison for Springleaf data

## 10 Future Work

Through the MNIST digit recognizer we have explored different dimension reduction and classification techniques and have combined them to produce more accurate results. Our experimentation lead to some possible avenues for further exploration including using an SVM parameter selection method based on kNN, incorporating a dimension reduction step into pairwise and one-vs-rest SVM, and extending discriminant analysis transformations into higher dimensions using features vectors in the kernel of the transformation, see (insert appropriate sections).

Using a pairwise or one-vs-rest kernel SVM classifier requires the training of multiple constituent kernel SVMs. Each kernel SVM has its own optimal set of parameters $C$ and $\sigma$ and different SVMs may have different optimal sets. Optimizing each constituent SVM separately multiplies the computation time needed to train the whole model. Therefore it is desirable to find a single set of parameters that optimize accuracy over the whole classifier and a method of finding it. We experimented with a method of selecting scale parameter $\sigma$ using kNN using the $\log(n)$ nearest neighbors.

Along with finding optimal parameters for kernal SVM, our use of a pairwise transformation before training each SVM slightly improved accuracy. It is possible the application of these local dimension reductions into the training of the constituent binary classifiers can improve accuracy general. Both require further study using the MNIST set and their application to different datasets to determine if they generalize to other kinds of data.

There are some topics that we covered near the end of the project or in passing that we would like to learn more about. Neural networks achieved the best results of all the classification methods tried here. We would like to understand more about neural networks especially convolutional neural networks and ways to increase training speed. In addition, model validation techniques such as cross validation are important topics we should continue to explore.

## 11 Acknowledgement

We would like to express our deep gratitude to the following people or organization: Prof. Chen for his guidance, valuable comments and suggestions in our project; Profs. Bremer and Simic for their proposal that made this project possible; Woodward Foundation for generous funding; and Prof. Simic for all the support.

## References

[1] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[2] J. C. Burges. Dimension reduction : A guided tour. *Foundations and Trends in Machine Learning*, 2(4):275–365, 2009.

[3] Hsu C. Lin C. A comparison of methods for multi-class support vector machines. *Neural Network*, 13(2):415–425, 2002.

[4] Vapnik V. Cortes C. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[5] Jerome H Friedman. Regularized discriminant analysis. *Journal of the American statistical association*, 84(405):165–175, 1989.

[6] K Fukunaga and JM Mantock. Nonparametric discriminant analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):671–678, 1983.

[7] Andrej Gisbrecht, Alexander Schulz, and Barbara Hammer. Parametric nonlinear dimensionality reduction using kernel t-sne. *Neurocomputing*, 147:71–82, 2015.

[8] Jianping Gou, Lan Du, Yuhong Zhang, Taisong Xiong, et al. A new distance-weighted k-nearest neighbor classifier. *J. Inf. Comput. Sci*, 9(6):1429–1436, 2012.

[9] Klaus Hechenbichler and Klaus Schliep. Weighted k-nearest-neighbor techniques and ordinal classification. Technical report, Discussion paper//Sonderforschungsbereich 386 der Ludwig-Maximilians-Universität München, 2004.

[10] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[11] Ian T Jolliffe. Principal component analysis and factor analysis. *Principal component analysis*, pages 150–166, 2002.

[12] Nandakishore Kambhatla and Todd K Leen. Dimension reduction by local principal component analysis. *Neural Computation*, 9(7):1493–1516, 1997.

[13] Effrosini Kokiopoulou, Jie Chen, and Yousef Saad. Trace optimization and eigenproblems in dimension reduction methods. *Numerical Linear Algebra with Applications*, 18(3):565–602, 2011.

[14] Ming Li and Baozong Yuan. 2d-lda: A statistical linear discriminant analysis for image matrix. *Pattern Recognition Letters*, 26(5):527–532, 2005.

[15] Tao Li, Shenghuo Zhu, and Mitsunori Ogihara. Using discriminant analysis for multi-class classification: an experimental investigation. *Knowledge and information systems*, 10(4):453–472, 2006.

[16] Zhifeng Li, Dahua Lin, and Xiaoou Tang. Nonparametric discriminant analysis for face recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(4):755–761, 2009.

[17] Tom M. Mitchell. *Machine Learning,Draft of Chapter 3*. McGraw Hill, 2015.

[18] Sunil Ray. 6 easy steps to learn naive bayes algorithm (with code in python), 2015.

[19] scikit learn. 1.9. naive bayes.

[20] Ben-David Shalev-Shwartz. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014.

[21] Santosh Srivastava, Maya R Gupta, and Béla A Frigyik. Bayesian quadratic discriminant analysis. *Journal of Machine Learning Research*, 8(6):1277–1305, 2007.

[22] stack overflow. A simple explanation of naive bayes classification.

[23] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[24] Antonio Carlos Gay Thome. *Svm classifiers-concepts and applications to character recognition.* INTECH Open Access Publisher, 2012.

[25] Boser B. Guyon I. Vapnik V. A training algorithm for optimal margin classiers. *In Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[26] Laurens van der Maaten. Learning a parametric embedding by preserving local structure. *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 5:384–391, 2009.

[27] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*, pages 91–109. Springer, 2003.

[28] Todd Will. Introduction to the singular value decomposition. *Davidson College*, 1999.

[29] Tao Xiong, Jieping Ye, Qi Li, Ravi Janardan, and Vladimir Cherkassky. Efficient kernel discriminant analysis via QR decomposition. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pages 1529–1536, 2004.

[30] Jieping Ye and Qi Li. A two-stage linear discriminant analysis via qr-decomposition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(6):929–941, 2005.

[31] Jieping Ye, Qi Li, Hui Xiong, Haesun Park, Ravi Janardan, and Vipin Kumar. Idr/qr: an incremental dimension reduction algorithm via qr decomposition. *Knowledge and Data Engineering, IEEE Transactions on*, 17(9):1208–1222, 2005.

## A Sample codes (MNIST)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% CAMCOS   Function: LocalKmeansAnalysis
%%% Student: Wilson A. Florero-Salinas          date: 9/17/15
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LocalKmeansAnalysis(k)
% k = number of neighbors in the local search; pred = predictions ;
% Assumes that data sets are in correct format (each observation is a row)
% Uses a local kmeans search for classification. Algorithm: (1) Given a test
% observation, find kNN for each set of digits. (2) Compute average of the kNN.
% (3) Compare observation with each (local) kNN average.
% (4) Assign to class that has closest trained center.

function [pred, Error] = LocalKmeansAnalysis(Xtr,ytr,Xtst,ytst, k)

[Ntst, N_vars]  = size(Xtst);
Ntr = length(ytr);
position = 1:Ntr;
LocalCenters = zeros(10,N_vars);
pred = zeros(Ntst,1); %vector that will stores the predictions.

for i = 1:Ntst
  for j = 0:9
    labels = ytr == j;
    [indx,~] = knnsearch(Xtr(labels,:), Xtst(i,:), 'k',k);
    x = position(labels); indx = x(indx); % preserve locations from
                                          % original data set.
    LocalCenters((j+1),:) = mean(Xtr(indx',:),1);
  end
  [idx,~] = knnsearch(LocalCenters, Xtst(i,:), 'k',1);
  pred(i,1) = idx - 1;
end

%Calculating Classification Error
Error = 1 - sum(pred == ytst)/Ntst;
fprintf('Misclassification Error = %.2f%%, using k = %i \n', Error*100, k)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%   k-Nearest Neighbors   %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---input---
%k: neighborhood size
%wmethod: weighting scheme- 'weighted','gaussian','dual','none'
%Xtr: dxn training data
%ytr: nx1 training labels
%---output---
%class: predicted class

function[class]=WkNNclassify(k,newData,wmethod, Xtr,ytr)

%calculates euclidean distance using column vectors as data points
n=size(Xtr);
temp=(Xtr-repmat(newData,1,n(2))).^2;  %subtract newData accross columns of ↩
    trainingData
temp=sqrt(sum(temp,1));                %square root of column-sums (euclidean ↩
    distance)
[D,I]=sort(temp);

if strcmp(wmethod, 'gaussian')  %gaussian kernel
    dmax=D(k+1);
    D=D(1:k); I=I(1:k);
    dweights=(1./dmax).*D;
    weights=(1./sqrt(2*pi))*exp(-0.5*(dweights.^2));
    W=[ytr(I), weights.'];
    Candidates=unique(ytr(I));
elseif strcmp(wmethod,'weighed') %weighted
    D=D(1:k); I=I(1:k);
    weights=(max(D)-D)./(max(D)-min(D));
    W=[ytr(I), weights.'];
    Candidates=unique(ytr(I));
elseif strcmp(wmethod, 'dual')  %dual weighted
    D=D(1:k); I=I(1:k);
    weights=((max(D)-D)./(max(D)-min(D))).*((max(D)+min(D))./(max(D)-D));
    W=[ytr(I), weights.'];
    Candidates=unique(ytr(I));
elseif strcmp(wmethod, 'none')  %no weighted
    D=D(1:k); I=I(1:k);
    weights=ones(length(D),1);
    W=[ytr(I), weights];
    Candidates=unique(ytr(I));
else
    disp('Error')
end

m=length(Candidates);
temp_sum=zeros(m);
for i=1:m
    temp_sum(i)=sum(W((find(W(:,1) == Candidates(i))),2));
end
[~, I2]=sort(temp_sum, 'descend');
class=Candidates(I2(1));
end
```

```
%%%%%%%%%%%%%%%%%%%%%%
%%%%   NDA Code   %%%%
%%%%%%%%%%%%%%%%%%%%%%

%%%  NDAprojected()  %%%
%Xtr: new data, observations in columns
%ytr: class labels for observations in data
%d: number of features to extract
%k: size of neighborhoods for local centers in Sb
%a: weight exponent
%b: scalar in S + bI
%cent: 'center' to center data
function[transf]=NDAproject(Xtr,ytr,k,a,b, cent)

isCent=false;
if strcmp(cent, 'center')
    [~,N]=size(Xtr);
    Xtr=Xtr-repmat(mean(Xtr,2),1,N); %center data
    isCent=true;
end

[SB, SW]=NPscatterMatrix(Xtr, ytr, k,a);
SW=SW+eye(size(SW)).*b;
E=inv(SW)*SB;
[W,D] = eig(E);
[D,I] = sort(diag(D),'descend');
W = W(:,I);

transf=struct('mat',W,'eigs',D,'method','NDA','k',num2str(k),'a',num2str(a),'bump',↩
    num2str(b),'centered',isCent);
end


%%%  withinScatter()  %%%
%creates Swi matrix
%Sw=sum(Swi)
%---input---
%Xtr=dxn matrix
%index for subsetting data by class
%---output---
%within scatter matrix for ith class

function[SWi]=withinScatter(Xtr, index) %obs is in columns
temp=Xtr(:,index);                      %subset columns by index
n=size(temp);

SWi=zeros();
mu=mean(temp,2);
for i=1:n(2)
    temp2=(temp(:,i)-mu);
    SWi=SWi+temp2*(temp2.');
end

end

%%%  NPbetweenScatter()  %%%
%creates Sb matrix
%---input---
%k: neighborhood size
%a: exponent on distance weights
%---output---
%Sb matrix
```

```
function[Sb]=NPbetweenScatter(Xtr,ytr,k,a)
classes=unique(ytr);
Sb=zeros();

%outer: by class
%inner: by each point
%inter-inter: rest of classes

for j=1:length(classes)         %for each class
    subSet=Xtr(:,logical(ytr==classes(j)));   %subset data by class
    Nj=size(subSet,2);
    for l=1:Nj                      %for each point in class
        %apply neighborData() to lth point in jth class
        [w,Mj]=neighborData(subSet(:,l),classes(j),Xtr,ytr,k,a);
        for i=1:size(Mj,2)          %for classes in Mj vector
           y=subSet(:,l)-Mj(:,i);
           Sb=w(i).*(y*y')+Sb;      %iterative sum to Sb matrix
        end
    end
end

end


%%%   NDAproject()   %%%
%---input---
%newPoint
%c: class of newPoint
%Xtr: dxn data matrix
%ytr: nx1 vector of labels
%k: neighborhood size
%a: exponent on distances used in weights
%---output---
%weights: (c-1)x1 vector of weights
%centers: nx(c-1) matrix of local centroids of newPoint, excluding c

function[weights, centers]=neighborData(newPoint, c, Xtr, ytr, k, a)

classSet=unique(ytr);

m=length(classSet);   %num class
n=size(Xtr);

%euclidean distance of newPoint from each training point
dist=sum(Xtr-repmat(newPoint,1,n(2))).^2;
dist=sqrt(sum(dist,1));

centers=zeros(n(1), m);                      %initialize storage for local centers

parfor j=1:m                                 %iterate over number of classes
    index=find(ytr == classSet(j));
    tempdat=Xtr(:,index);                    %subset by class
    tempdist=dist(index);                    %subsets distances
    [~,I]=sort(tempdist);                    %sort distances
    nbhd=I(2:(k+1));                         %k-nearest points from class, excluding ↩
       newPoint
    centers(:,j)=mean(tempdat(:,nbhd), 2);   %calculate class center
end

%calculates euclidean distance of newPoint from centers
dist2=(centers-repmat(newPoint,1,m)).^2;  %subtract newData accross columns of ↩
   trainingData
dist2=sqrt(sum(dist2,1));                 %distances between class centers
```

```
%distance from local center of its own class
dist_i=dist2(logical(classSet==c)==1);
%distances from local centers of other classes
dist_j=dist2(logical(classSet==c)<1);

weights=zeros(length(dist_j),1);

%calculates weights
for j=1:length(dist_j)
    if dist_i<=dist_j(j)
        weights(j,1)=dist_i.^a/(dist_i.^a + dist_j(j).^a);
    else
        weights(j,1)=dist_j(j).^a/(dist_i.^a + dist_j(j).^a);
    end
end

centers=centers(:,logical(classSet==c)<1); %remove newPoints's class from local ↩
    centers
end
```

```
%%% TwoDScatterMatrix() %%%
%create Sw and Sb matrices for use in inv(Sw)*Sb
%---input---
%data: dxdxn data matrix
%labels: nx1 vector of labels
%---output---
%Sw and Sb matrices
function[Sw, Sb]=TwoDscatterMatrix(data, labels)
classes=unique(labels);
L=length(classes);
Sb=TwoDbetweenScatter(data, labels);  %apply TwoDbetweenScatter()

Sw=zeros();
%iterate TwoDwithinScatter() over classes to create Sw matrices
parfor k=1:L
    tempIndex=find(labels==(k-1));
    Sw=Sw+TwoDwithinScatter(data, tempIndex);
end
end

%%% TwoDwithinScatter() %%%
%---input---
%data: dxdxn data matrix
%index: index of class
%---output---
%Swi matrix of the class
function[SWi]=TwoDwithinScatter(data, index)
temp=data(:,:,index);    %subset by third dimension
MU=mean(temp,3);         %take mean along 3rd dimension
n=size(temp);
SWi=zeros();
temp2=zeros();

parfor i=1:n(3)       %iterate over 3rd dimension
    temp2=(temp(:,:,i)-MU);  %center image
    SWi=SWi+(temp2')*temp2;
end
end

%%% TwoDbetweenScatter() %%%
%---input---
%data: dxdxn data matrix
```

```
%labels: nx1 label vector
%---output---
%Sb matrix
function[Sb]=TwoDbetweenScatter(data, labels)
classes=unique(labels);
L=length(classes);
Sb=zeros();
grandMU=mean(data, 3);                %grand mean along 3rd dimension

for i=1:L                             %iterate over number of classes
    tempIndex=find(labels==(i-1));    %find indices corresponding to ith class
    temp=data(:,:,tempIndex);         %subset by indices
    N=size(temp,3);
    tempMU=mean(temp,3);              %mean of subset over 3rd dimension
    temp2=tempMU-grandMU;             %mean minus grand mean
    Sb=Sb+N.*(temp2')*temp2;
end
end

%%% TwoDkNNclassify() %%%
%kNN using matrix norms
%---input---
%k: neighborhood size
%newPoint:
%Xtr: dxn training data
%ytr: training labels
%metric: 'fro'=L2 norm, 'col'=feature vector norm, 1=L1 norm
%---output---
%class: predicted class

function[class]=TwoDkNNclassify(k,newPoint,Xtr, ytr,metric)
n=size(Xtr);
temp=zeros(n(3),1);
%find way to apply columnDistance to array

if strcmp(metric,'fro')
    for i=1:n(3)
        temp(i)=norm(Xtr(:,:,i)-newPoint,'fro');
    end
elseif strcmp(metric,'col')
    for i=1:n(3)
        temp(i)=sum(sqrt(sum((Xtr(:,:,i)-newPoint).^2)));
    end
elseif metric==1
    for i=1:n(3)
        temp(i)=norm(Xtr(:,:,i)-newPoint,metric);
    end
end

[~,I]=sort(temp);
nbhd=I(1:k);
nbhd=ytr(nbhd);
class=mode(nbhd);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% CAMCOS  globalPCALIBSVMpairs   Version: 1.0   date: 10/23/15
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; close all; clc
load('mnist60k10datadskw','ytr','ytst')     %load labels deskewed dataset.
%load deskewed dataset for up to 300 principal compoents.
load('globalPCA300skw','XtrPCA','XtstPCA')
N_test = length(ytst); M = nchoosek(0:9,2);
numModel = size(M,1);   %number of models (45 in total)
c = 3;         % cost parameter in options.
gamma = [0.01:0.002:0.05]; gVals = length(gamma);
C = 300;  % up to how many components to test? Only multiples of 10.
for N_vars = 10:5:60;
    Errors = zeros(gVals,1);
    fprintf('Beginning global PCA + LIBSVM Pairs Analysis with c fixed\n')
    for i = 1:gVals
     options = ['-s 0 -c 4 -t 2 -g ', num2str(gamma(i)), ' -q'];
     [~, Errors(i)] = LIBSVMPairMethod(XtrPCA(:,1:N_vars), ytr, XtstPCA(:,1:N_vars), ↩
         ytst, N_test, M, numModel, options);
    end
    figure;
    plot(gamma, Errors*100,'-b.', 'Markersize', 14);
    title(strcat('Error plot for global PCA + LIBSVM Pairs using c = ', num2str(c), '↩
         and PCA = ', num2str(N_vars)));
    set(gca,'xtick', gamma); xlabel('gamma values'); grid on;
    ylabel('Percent Error');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% CAMCOS   Function: LIBSVMPairMethod  Version: 1.1 date: 10/11/15
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Assumes that data set is in correct format i.e., each
% row is an observation. globalPCASVMPair builds pair n(n+1)/2 models.
% For example, 0 vs 1,...,0 vs 9,..., 8 vs 9. Each model is applied to each
% new observation. For that observation the class that was predicted most
% often is assigned as the predicted class.
% c = matrix of all possible combinations (45 total) so that numModel = 45.

  function [pred, Error] = LIBSVMPairMethod(Xtr, ytr, Xtst, ytst, N_test, M, numModel↩
      , options)

  PRED = zeros(N_test, numModel); %each model's predition is stored

   % Model training and prediction.
    fprintf('Begin model training. Ignore the following 45 outputs \n')
  for i = 1:numModel
    l1 = ytr == M(i,1); l2 = ytr == M(i,2);
    X = [Xtr(l1,:); Xtr(l2,:)];
    y = [ytr(l1); ytr(l2)];
    model = svmtrain(y, X , options);
    [PRED(:,i)] = svmpredict(ytst, Xtst, model);
  end
   fprintf('Ending model training. \n')

%Calculating Classification Error
 pred = mode(PRED,2);
 Error = 1 - sum(pred == ytst)/N_test;
 fprintf('Misclassification Error for SVM Pair method = %.2f%% \n', Error*100)
  end
```

```
################################################################################
################## METHOD: local pca + 1 VS rest SVM  #######################
```

```
# local pca: perform pca on each digit(0,1,...,9),
# which leads to 10 reduced subspaces based on 10 different classes.
# We project the whole training data set and the test data set on the 10 subspaces
# and train 10 1VSrest SVM. There are 10 scores for each test data in total,
# and we predict the test data set into the class which has the largest score.
#################################################################################

__author__ = 'Dan Li'

import numpy as np
import random
from sklearn import cross_validation, svm, datasets
from sklearn import metrics
import time, sys
import scipy.io
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


# load MNIST data from .mat file
Xtr = train_data['trainImages'].T
ytr = train_data['trainLabels'].ravel()
Xtst = test_data['testImages'].T
ytst = test_data['testLabels'].ravel()
num_train = ytr.shape[0]
print Xtr.shape, ytr.shape, num_train
num_test = ytst.shape[0]
print Xtst.shape, ytst.shape, num_test
#exit(0)


# define the pca function
def pca_from_svd(data, n_comp):
    """
    :param data:
    :param n_comp:
    :return subspace of top n_comp PCA components
    """
    data = data - np.mean(data,axis=0)
    U,S,Vt = np.linalg.svd(data,full_matrices=False)
    # sort the PCs by descending order of the singular values (i.e. by the
    # proportion of total variance they explain)
    ind = np.argsort(S)[::-1]
    V = Vt.T
    V = V[:, ind]
    return V[:,0:n_comp]


# perform PCA on 0, 1, 2...9 respectively
# and obtain the subspace with ncomp top pca components
ncomp_list = [25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100]
for ncomp in ncomp_list:
    num_class = 10
    groups = {}
    sub_spaces = {}
    labels = {}
    for i in range(num_class):
        labels[str(i)] = np.where(ytr==float(i),1.,0.)
        groups[str(i)] = Xtr[labels[str(i)]==1.]
        sub_spaces[str(i)] = pca_from_svd(groups[str(i)],ncomp)

    start = time.time()
    # initialize the model (5, 0.034)
    para_C = 1
    p=1.0/ncomp
    para_gamma = [p,p,p,p,p,p,p,p,p,p]

    # for each class (0,1,2...9), project all the data to the corresponding subspace
    # and construct 10 one-vs-rest binary SVC
    clf = {}
    for i in np.arange(num_class):
        print 'parameter: %s' %str(para_gamma[i])
        clf[str(i)] = svm.SVC(kernel='rbf', C=para_C, gamma=para_gamma[i])
        project_X = np.dot(Xtr-np.mean(groups[str(i)],axis=0),sub_spaces[str(i)])
        clf[str(i)].fit(project_X, labels[str(i)])

    # pass the projected test data to the 10 SVC.
    # The SVC that predict 1 will be nominated  as the predicted class.
    # if multiple SVC predict 1,
    # then pick the smallest index of SVC as the predicted class
    bin_predicted_y = np.zeros([num_class, ytst.shape[0]])
    for i in np.arange(num_class):
        project_test_X = np.dot(Xtst-np.mean(groups[str(i)],axis=0), sub_spaces[str(i↩
           )])
        bin_predicted_y[i,:] = clf[str(i)].decision_function(project_test_X)

    predicted_y = np.argmax(bin_predicted_y,axis=0)
    print '--------report-----------\n'
    print '-----------parameter----------'
    print 'C = %s, gamma = %s, pca = %s' %(str(para_C), str(para_gamma), str(ncomp))
    print 'confusion matrix: \n %s' %metrics.confusion_matrix(ytst, predicted_y)
    print 'precision score: %s'
        %metrics.precision_score(ytst,predicted_y,average='weighted')

    print "---- %s sec-----" %(time.time()-start)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%   Local Pairwise Dimension Reduction + SVM  %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%% pairwiseTransformSVM() %%%%%%%%%%%%
%---input---
%d: number of dimensions
%cost: SVM cost parameter
%Xtr: observations in columns
%ytr: labels
%Vmats: C(n,2)x1 cell array containing pairwise transformation matrices
%params: parameters to be passed on to svmtrain(). If using local variance for ↩
   Gaussian parameter, is nx1
%matrix
%cent: 'center' to center data about the mean of the ith pair
%---output---
%C(n,2)x2 cell array with trained SVM model and pair mean

function[pairModel]=pairwiseTransformSVM(d,cost, Xtr, ytr, VMats, params, cent)

C=length(VMats);
pairModel=cell(C,1);
means=cell(C,1);


for i=1:C   %iterates over pairs
      sub=VMats(i).set.classes;          %get transformation

      %subset labels and data by class pair
      indx=ismember(ytr,sub);
      indx=find(indx==1);
      labsTemp=ytr(indx);
      trainTemp=Xtr(:,indx);

      %center data
      if strcmp(cent,'center')
          [~,N]=size(trainTemp);
          means{i}=mean(trainTemp,2);                %mean of paired data
          trainTemp=trainTemp-repmat(means{i},1,N); %center pair data
      else
          [N,~]=size(trainTemp);
          means{i}=zeros(N,1);   %mean of paired data
      end

      %apply VMat pair transformation to pair data
      Vtemp=VMats(i).set.mat;
      trainTemp=Vtemp(:,1:d)'*trainTemp;  %d=features

      %processes 'params'
      if size(params,1) > 1
          i1=sub(1)+1; i2=sub(2)+1;       %get indices of class pair
          params2=params(i1)*params(i2);  %calculates Gaussian shape parameter
      elseif (size(params,1)==1) || (isempty(params)==1)
          params2=params;  %passes user-defined scalar parameter
      else
          warning('Warning! Error in variable params')
      end

      %format parameters for libsvm
      params2=strcat('-t 2',{' '},'-g',{' '},num2str(params2),'-c',{' '},num2str(cost↩
          ));
      %train SVM model for transformed pair data
      pairModel{i}=svmtrain(labsTemp,trainTemp',params2);
end
pairModel=[pairModel,means];
end

%%%%%%%%%%%% pairtransformPredict() %%%%%%%%%%%%
%predicts new data using C(n,2) SVM models. For each new observation, is
%predicted using all models, then assigned to mode of the results
%---input---
%d: dimensions
%inData: data
%svmmodels: C(n,2)x2 cell array
%cent: 'center' to use center passed from pairwiseTransformSVM()
%Vmat:
%---output---
%nx1 results vector
function[results1]=pairtransformPredict(d, inData, svmmodels, Vmat, cent)

newData=inData;
C=length(svmmodels);
n=size(newData,2);
results=zeros(n,C);
dValues=results;

for k=1:C     %iterate over pairs
    meanTemp=svmmodels{k,2};           %get mean of kth train pair
    if strcmp(cent, 'center')
        temp=newData-repmat(meanTemp,1,n); %center test data around kth pair center
    elseif strcmp(cent, 'nocenter')
        temp=newData;
    end
    %temp=newData;
    %transform test data
    modelTemp=svmmodels{k,1};              %get kth svm classifier
    Vtemp=Vmat(k).set.mat;                 %get kth pair transformation matrix
    temp=Vtemp(:,1:d)'*temp;               %project test data into
    [testResults, ~, dVal]=svmpredict(zeros(n,1),temp',modelTemp);   %classification
    dValues(:,k)=dVal;           %decision value
    results(:,k)=testResults;    %results of kth trial in row
end

W=pairWeights(results,dValues,'none');
classes=unique(results);
results1=zeros(n,1);

%for j=1:n   %apply pairVote()
%    results1(j)=pairVote(results(j,:),W(j,:),classes);
%end
results1 = mode(results,2);
end
```

```

param1 = list("objective" = "multi:softmax","eval_metric" = "mlogloss","num_class" = ↩
    10,
              "eta" = 0.18, "colsample_bytree"=0.5,"subsample" = 3, "verbose"= 0)
bst.cv2 = xgb.cv(param = param1, data = Xtr, label = ytr, nfold = cv.nfold, nrounds =↩
     cv.nround)
plot(bst.cv2$test.mlogloss.mean, lty = 1)
nround2 = which(bst.cv2$test.mlogloss.mean == min(bst.cv2$test.mlogloss.mean))
nround2
# resul is nround = 280
############################################
# train the model
############################################
bst2 = xgboost(data=Xtr, label = ytr, param = param1, nrounds = nround2)

############################################
# predict the model  & give Error
############################################
ypred2 = predict(bst2, Xtst)

Error = sum(ypred2 != ytst)/N.test;
Error*100

############################################
# Results + parameters
############################################
#param1 = list("objective" = "multi:softmax","eval_metric" = "mlogloss",
#             "num_class" = 10, "eta" = 0.2, "colsample_bytree"=0.5,"subsample" = 1)
#nround2=317, Error = 1.86%
#param1 = list("objective" = "multi:softmax","eval_metric" = "mlogloss",
#"num_class" = 10, "eta" = 0.2, "colsample_bytree"=0.5,"subsample" = 3)
#nround = 267, Error = 1.83%
#param1 = list("objective" = "multi:softmax","eval_metric" = "mlogloss",
#"num_class" = 10, "eta" = 0.2, "colsample_bytree"=0.5,"subsample" = 6)
#nround = 260, Error = 1.86
#param1 = list("objective" = "multi:softmax","eval_metric" = "mlogloss",
#              "num_class" = 10, "eta" = 0.4, "colsample_bytree"=0.5,"subsample" = 3)
#nround = 169, Error = 2.00%
#param1 = list("objective" = "multi:softmax","eval_metric" = "mlogloss","num_class" =↩
     10,
#"eta" = 0.18, "colsample_bytree"=0.5,"subsample" = 3, "verbose"= 0)
#nround = 308, Error = 1.83%
```

```
library(xgboost)
load("mnistdata60k10kdskw.RData")
N.tst = length(ytst)

############################################
# cross-validation to choose the parameters
############################################
param = list("objective" = "multi:softmax","eval_metric" = "merror", "num_class" = ↩
    10,
             "eta" = 0.18, "colsample_bytree"= 0.45,"subsample" = 3, "verbose"= 0, "↩
                 seed" = 10)

cv.nround = 500
cv.nfold = 5

bst.cv = xgb.cv(param = param, data = Xtr, label = ytr, nfold = cv.nfold, nrounds = ↩
    cv.nround)
plot(bst.cv$test.merror.mean, lty = 1)
nround = which(bst.cv$test.merror.mean == min(bst.cv$test.merror.mean))
############################################
# train the model
############################################
nr = length(nround)
error = rep(0,nr)
ypred = matrix(rep(0,N.tst*nr),N.tst,nr)

for(k in 1:nr){
  bst = xgboost(data = Xtr, label = ytr, param = param, nrounds = nround[k])
  ypred[,k] = predict(bst, Xtst)
  error[k] = sum(ypred[,k] != ytst)/N.tst
}

indx = which( min(error) == error)
nround[indx]

############################################
# predict the model
############################################

#Error
Error = sum(ypred[,indx] != ytst)/N.test
cat('Misclassification Error for xgboost pair is Error =', Error*100, '%')

############################################
# prepare for output (save data)
############################################

write.csv(ypred, 'xgboostdskwpred.csv', quote = F, row.names = F)

############################################
# Results + parameters
############################################
#param = list("objective" = "multi:softmax","eval_metric" = "merror", "num_class" = ↩
    10,
#              "eta" = 0.18, "colsample_bytree"= 0.45,"subsample" = 3, "verbose"= 0, "↩
    seed" = 10)
# Error = 1.41%, nround = 498
```