Effects of Principal Component Analysis on K-Nearest Neighbor Classification

Stephen Xie

Abstract

In this project, Principal Component Analysis (PCA) is used to mitigate the complications of high-dimensional and noisy data. PCA is a dimensionality reduction, or more specifically, a feature extraction technique that yields features that capture maximum variance of the original features. The K-Nearest Neighbors (KNN) classifier is used on classification datasets to provide a control set. PCA is applied to the data and then reclassified by KNN. It is found that while retaining almost all the variance, the classification accuracy is unaffected even though the dimensionality may be reduced significantly. PCA works the best on highly correlated data.

Introduction

Classification is an instance of supervised learning. It is the problem of predicting a predefined class of a collection of data. A well-known and benchmarked classification problem is: Given an image of a single handwritten digit, predict the number it is depicting. The finite set of classes in this example is $\{0,1,3\ldots9\}$. The United States Postal Service uses this sort of recognition technology to automate the processing of mail, which is much faster than a human reading and sorting them. Real world data can be high dimensional and noisy. High dimensional datasets have high computational complexity, and can require a significant amount time and memory to process because of sheer size. In image classification, each pixel is a variable. A modest 800 x 600 pixel resolution digital photo has 800*600 = 480,000 dimensions. Noise is random error or variance in a measured variable [1]. Noise will exist in most real-world data. In this project I will be exploring the effects of dimensionality reduction, more specifically feature extraction on mitigating these two complications. The motivation for this project was my lack of understanding for *the merits and drawbacks of dimensionality reduction, and when it is appropriate*.

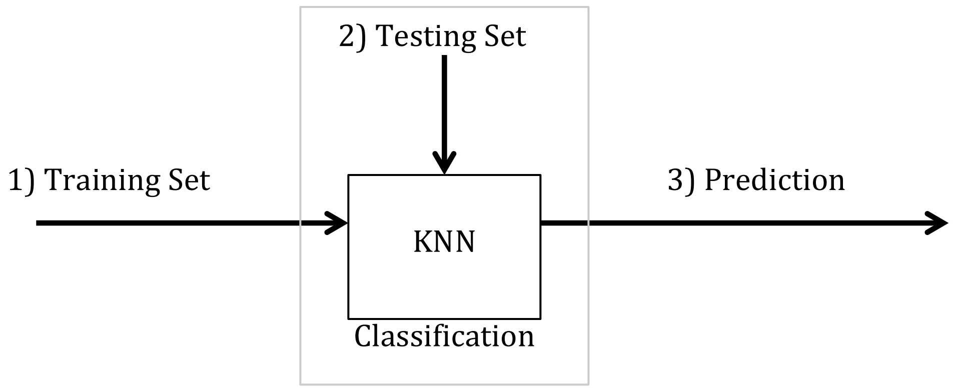
I will be using the K-Nearest Neighbors (KNN) [1] classification algorithm in this project. KNN is a non-parametric, deterministic model. It operates in a feature space where each data set is a feature vector. The dimension of the space corresponds to the number of variables in the data. KNN uses Euclidean distance as a metric to calculate the proximity of its neighbors. The use of this metric extends to clustering algorithms as well such as K-means clustering.

Our intuition of data and proximity in lower dimensions does not always generalize to higher dimensions. There are data visualization methods for higher dimensions but in general it is difficult to intuitively visualize beyond three dimensions. The distribution of high dimensional data is not obvious. The *Curse of Dimensionality* is a phenomenon proposed by Richard E. Bellman when he was solving problems in the field of dynamic optimization [5]. This phenomenon is commonly used to explain various complications of proximity in high dimensions. The volume of a feature space increases exponentially with the addition of extra dimensions. The volume increases too fast for the available amount of data and the data becomes sparse [5]. Under general conditions, as the number of dimensions in a space increases, all points converge to the same distance from a query point. Thus, the concept of using a distance metric to discriminate becomes meaningless [3]. The ratio of the distance of a point's nearest neighbor over the point's farthest neighbor approaches 1 as the number of dimensions increase [2]. Using a proximity metric to discriminate under this effect is no longer meaningful, as all points are approximately equidistant from each other.

Dimensionality Reduction is the process of reducing the number of random variables or attributes under consideration [1] Dimensionality reduction can be broadly divided into two approaches: Feature Extraction and Feature Selection. Feature Selection is selecting a subset of the existing variables without a transformation. Feature Extraction is transforming the existing variables into new variables where the number of new variables is lower than the original. Principal Component Analysis (PCA) is a feature extraction method. PCA will be used for this project. PCA is based on the sample covariance, which characterizes the scatter of the entire data set, irrespective of class membership. The projection axes chosen by PCA might not provide good discrimination power [6]. PCA is not always beneficial and can even be detrimental. When you map to lower dimensions, you are losing data, which can be both good and bad. Too much loss of data can leave you with a small sample that is not representative enough for accurate results. Or the data you lose might be noise that is unwanted in the first place.

Methodology

1. **Use the K-Nearest Neighbors classifier on a dataset partitioned into training and testing sets. I will be using the 1-Nearest Neighbor. Compare the predictions to the correct labels to get an accuracy of this model.**



2. **Perform Principal Component Analysis (PCA) on both the training set and testing set to produce a new, transformed training and testing set.**

The purpose of PCA is to linearly transform the original set of variables $\{x_1, x_2, \ldots , x_n\}$, into a set of new variables $\{v_1, v_2, \ldots , v_n\}$, called *Principle Components* (PCs). To reduce the number of dimensions from *n* to *p*, we project the original data onto $[v_1\ v_2\ \ldots\ v_p]$. PCs are axes of the new space so they are orthogonal. Each PC represents a direction of maximum variance of the original data. They are ordered so that the first PC is the direction of maximum variance, the second PC is the direction of maximum variance orthogonal to the first PC, the third PC is the direction of maximum variance orthogonal to the first and second PC, and so on. PCA can be implemented by *Singular Value Decomposition* (SVD) [4], which is also how R's *prcomp* function is implemented. SVD is a matrix factorization technique that returns the eigenvalues and eigenvectors of a matrix **X**:

$$[U\ \Sigma\ V] = svd(X)$$

$$X = U\Sigma V^T$$ [4]

Where:

- The columns of U are the eigenvectors of XX^T
- The columns of V are the eigenvectors of X^TX, corresponding to the ordered singular values of Σ
- The diagonal matrix Σ contains the singular values of V, ordered from largest to smallest

To project the data onto *k* dimensions, we calculate

$$X * V[1{:}k]$$

The percentage of variance captured by a PC can be calculated by dividing its corresponding singular value by the sum of all the singular values. I want to retain at least 99% variance in my PCs.

$$\text{Let } \Sigma = \begin{bmatrix} s1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & sn \end{bmatrix}$$

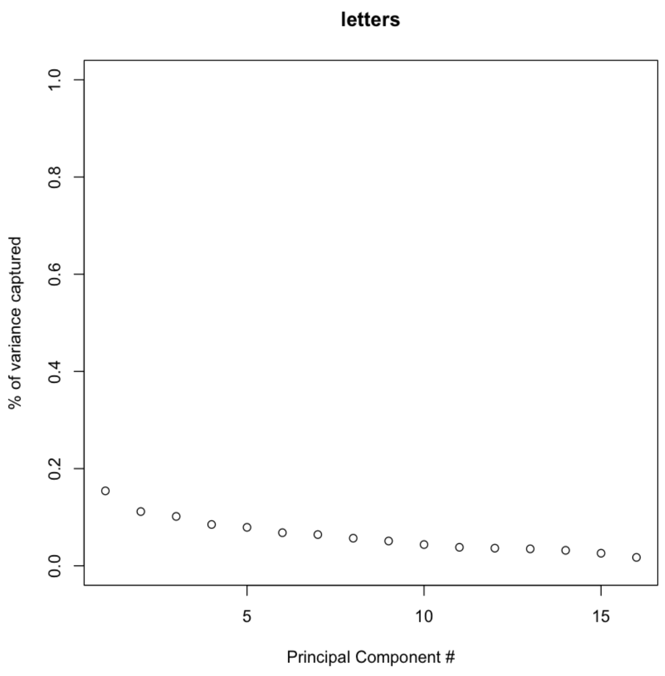
Here we find the smallest possible *k* for which, and *k* will be the number of dimensions we reduce to

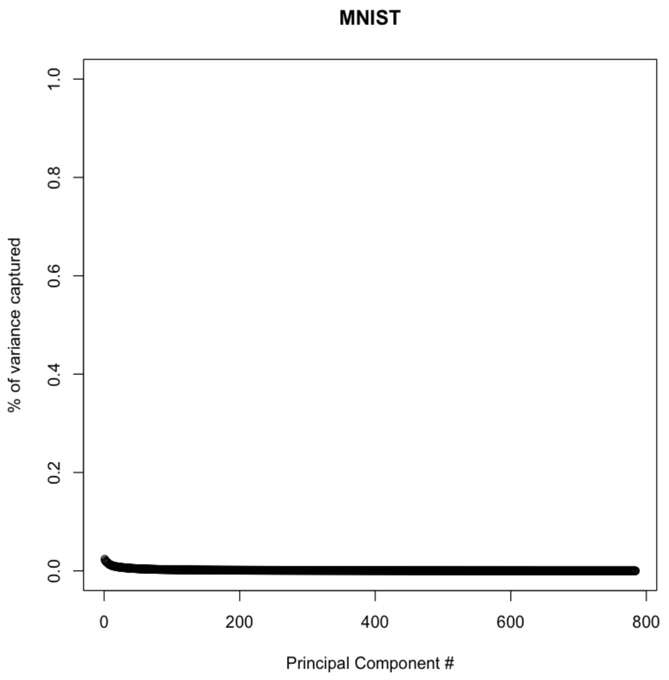
$$\sum_{i=1}^{k} si \,/\, \sum_{i=1}^{n} si \ \geq 0.99$$

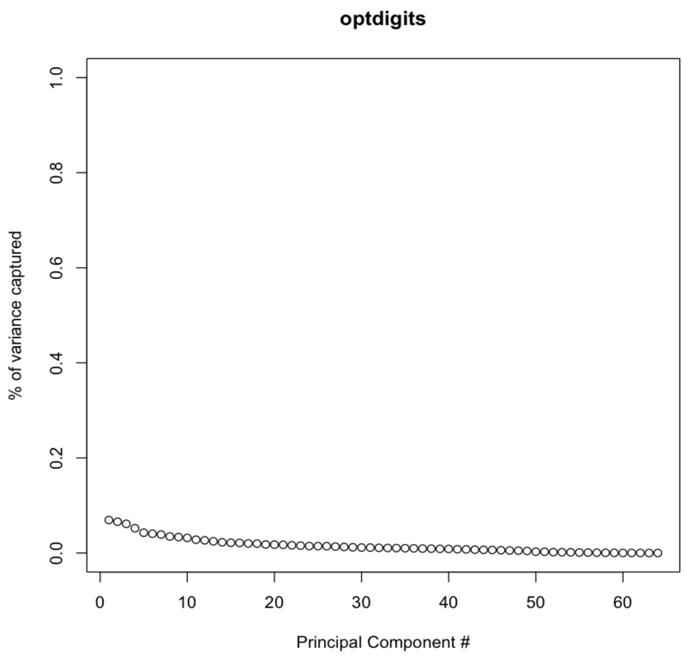
3. **Repeat step 1 on the *transformed* training and testing sets. Compare the classification results for step 1 and step 3.**

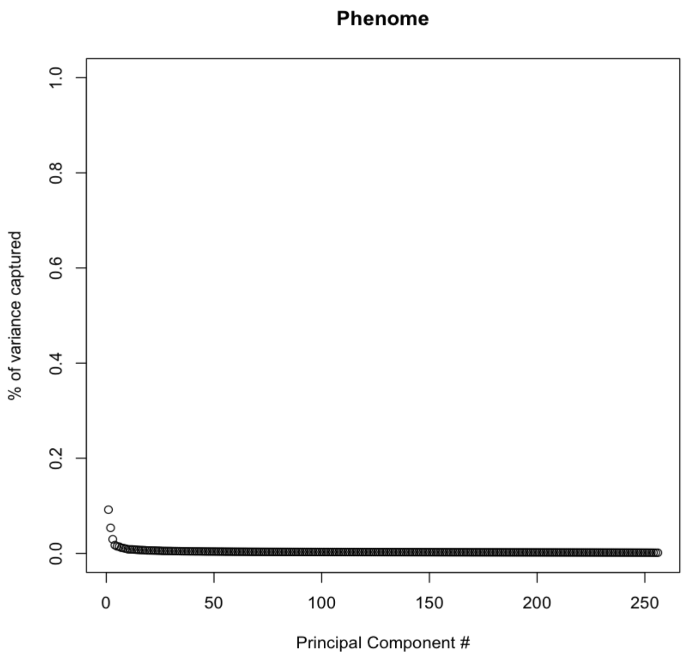
Results

Below are graphs of the proportion of variance of each PC of each dataset.









Dataset	Variables	# of PCs to capture 99%+ variance	Training Examples	Testing Examples
letters	16	16	14000	6000
optdigits	64	51	3822	1796
MNIST	784	556	29400	12600
phoneme	256	250	3156	1353

This chart displays the original number of variables, the minimum number of PCs it required to retain at least 99% of the variance, and how many examples were in the training and testing sets.

As we can see from these four graphs, none of the datasets had a single PC that captured a majority of the variance. The highest percentage of variance captured by a single PC was ~15%. From that we can say the variables of each respective dataset were not highly correlated. I expected the higher dimensional data would be able to significantly lower their dimension while retaining 99% variance. My intuition for this thought was that the more variables there are, the more correlation there must be by pure coincidence, but that is not necessarily true. The *MNIST* data set was able to reduce by a significant 228 variables from 784 to 556. But the *phoneme* dataset only reduced by 6 variables from 256 to 250. In accordance to that, I expected low dimensional datasets to require almost all their dimensions to retain all its variance. The *letters* dataset of only 16 variables required all 16 of its PCs to retain 99% variance. The *optdigits* dataset is also a handwritten digit recognition dataset like *MNIST* but with much fewer variables. It reduced its variables from 64 to 51, which was on par with my expectations. So, the only dataset with unexpected results was *phoneme.* Looking back at our graphs, you can notice that *phoneme* had the least significant leading PCs, with barely any upward trend at the beginning of the graph. We can conclude that *phoneme* was the least correlated dataset, which is why we could only reduce it by a mere 6 dimensions out of 256.

Dataset	KNN	KNN + PCA	%
letters	0.9553	0.9555	+0.0020
optdigits	0.9800	0.9800	0
MNIST	0.9663	0.9662	-0.0010
phoneme	0.8795	0.8810	0.0015

This chart displays the accuracy of the KNN classifier for before and after dimensionality reduction.

KNN performed well and all the accuracies were very high. The differences in accuracy between the original data and the reduced data are negligible. I expected both improvements and decreases, but absolutely did not expect to see almost identical accuracies. Because PCA is a linear transformation onto the same axes, and 99% of the variance was retained, the proportionality of the data must have remained the same. From these results, I can conclude that if able to retain almost all the variance, any lost data from reduction is negligible when using KNN for classification.

Discussion

I have found that using PCA for feature extraction can be a very powerful technique. It is to be used on data that is highly correlated for the largest reduction of dimensions and most discriminating principal components. It can also be rather useless, on lowly correlated data. If you need to capture enough variance that needs every single principal component, there may not be a need for PCA. You will not be reducing any dimensions in that case.

An improvement for my methodology would be to consider retaining lower amounts of variance, and then to compare them. I suspect there would be a bigger range of outcomes for that. For future research, I would like to explore more complex characteristics of the data to use as indicators of PCA's effects. There are still many statistical methods I have not yet learned or am not too familiar with (E.g. non-linear dimensionality reduction) so I was limited to just what I know. I did not want to delve into anything I was too unfamiliar with. I used only numerical datasets for this experiment but in the future I would also like to work with categorical or text data.

References

[1] Data Mining Concepts and Techniques, Jiawei Han, Micheline Kambar, Jian Pei

[2] On The Surprising Behavior of Distance Metrics in High Dimensional Space, Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim

[3] When Is "Nearest Neighbor" Meaningful? Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft

[4] UWaterloo Stat341 Lecture Notes, Ali Ghodsi
http://www.wikicoursenote.com/wiki/Stat341f11

[5] Dynamic Programming, Richard E. Bellman 1957

[6] Proceedings of International Conference on Internet Computing and Information Communications: ICICIC Global 2012, Swamidoss Sathiakumar, Lalit Kumar Awasthi, Roberts Masillamani, S S Sridhar

Appendix

Dataset Descriptions

- *Letters* is a numerical handwritten letter recognition dataset with 16 attributes
- *Optdigits* is a numerical handwritten digits dataset with 64 attributes
- *MNIST* is a numerical handwritten digit dataset, with 784 attributes
- *Phoneme* is a numerical digitalized speech dataset with 256 attributes

R code

On the next two pages

```
# this was code for one of my datasets but they are essentially the same
except for loading different datasets

# import extra libraries. class contains the knn classifier
library(class)

# load data
phoneme = read.csv("phoneme.data.csv")
letters_data_train = phoneme[1:3156,2:257]
letters_data_train_labels = phoneme[1:3156,258]
letters_data_test = phoneme[3157:4509,2:257]
letters_data_test_labels = phoneme[3157:4509,258]

# KNN, use k = 1
set.seed(1234) # set seed for consistent randomization of tie breakers
in KNN
letters_predictions = knn(train=letters_data_train,
test=letters_data_test, cl=letters_data_train_labels, k = 1)

# calculate number of correct vs incorrect predictions
accuracy = numeric(length(letters_predictions))
for (i in 1:length(letters_predictions)) {
  if (letters_predictions[i] == letters_data_test_labels[i]) {
    accuracy[i] = 1
  } else {
    accuracy[i] = 0
  }
}

# output accuracy
print(paste("total correct:" , sum(accuracy) , ", out of:" ,
length(letters_predictions)))
print(paste("accuracy:" , mean(accuracy)))


# Principal Component Analysis

# choose to center data, but not scale
pc = prcomp(letters_data_train, center = TRUE, scale. = FALSE)

# convert singular values to proportions
pcpercentage = apply(X = as.matrix(pc$sdev), MARGIN =2,  FUN =
function(x) x/sum(pc$sdev) )


# find smallest k that captures 99% of variance
minimumk <- function(eigenvalues, var) {
  for (i in 1:length(pcpercentage)) {
    vartotal = 0
    for (n in 1:i) {
      vartotal = vartotal + eigenvalues[n]
      if (vartotal/sum(eigenvalues) >= var) { return(n) }
    }
  }
}

k = minimumk(pcpercentage, var = 0.99)
print(paste("k:", k))
plot(pcpercentage, ylim=c(0,1), main = 'Phenome', xlab = 'Principal
Component #', ylab = '% of variance captured')


# project original data onto first k principal components, reducing to k
dimensions
new_train = as.matrix(letters_data_train) %*%
as.matrix(pc$rotation[,1:k])
new_test = as.matrix(letters_data_test) %*% as.matrix(pc$rotation[,1:k])

# KNN again after PCA
set.seed(1234)
letters_predictions = knn(new_train, new_test,
cl=letters_data_train_labels, k = 1, prob=TRUE)

# record number of correct vs incorrect predictions
accuracy = numeric(length(letters_predictions))
for (i in 1:length(letters_predictions)) {
  if (letters_predictions[i] == letters_data_test_labels[i]) {
    accuracy[i] = 1
  } else {
    accuracy[i] = 0
  }
}
# output accuracy
print(paste("total correct:" , sum(accuracy) , ", out of:" ,
length(letters_predictions)))
print(paste("accuracy: " , mean(accuracy)))
```