LECTURE ON DIMENSIONALITY REDUCTION

PEARL ARMSTRONG and ANGYALKA VALCSICS
INTRODUCTION TO STATISTICAL DATA MINING
STA 588

November 11, 2020

DIMENSIONALITY

Across a large variety of fields data is being collected every day. Data mining, or knowledge discovery of data, is the process of finding patterns, anomalies and correlations within the data to make a prediction. The results from data mining depend on the quality and quantity of the available data. When discussing data, the **dimensionality** of the data set is the number of features or attributes. The **curse of dimensionality** refers to the difficulties encountered when processing high dimensional data, these difficulties are not an issue when processing low dimensional data. In order to avoid the curse, **preprocessing** of the data and dimensionality reduction is a necessary step. To reduce the number of features, we can use feature selection, selecting a subset of relevant features and discarding the others. Or feature extraction, which produces a reduced set of features summarizing the majority of information contained in the original set. With fewer attributes, the interpretability of the model will increase, while the complexity and training time decreases.

The **curse of dimensionality** is the phenomenon in data analysis describing significant increase in difficulties encountered as the number of dimensions grows. The complexity of many algorithmic approaches of data mining in $\mathbb{R}^n$ space increases exponentially as n becomes large. As the number of attributes grows, the data becomes increasingly sparse in the volume of space the data occupies and the distance between two data points increases resulting in less similarity. Sparsity of data is problematic for methods requiring statistical significance since the amount of data needed to accurately and reliably build statistical models grows exponentially with the number of dimensions. When random sampling, the sampled data may not be representative of all possible outcomes due to the lack of data points. The differences in data density and the increased distance between points results in many clustering and classification algorithms performing poorly.

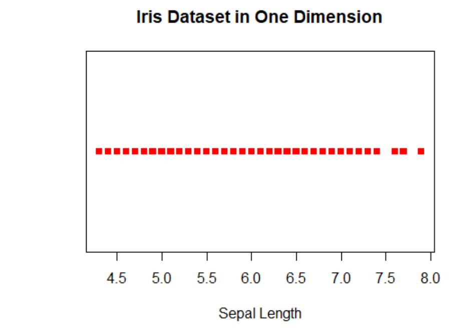
A **feature vector** is an n-dimensional vector of numerical features that represent some object. The vector space associated with these vectors is often called the **feature space**. If you are not familiar with vector spaces, a vector space is a collection of objects called vectors, which may be added together and multiplied by some scalar value. The operations of vector addition and scalar multiplication must satisfy certain requirements, or axioms, in order for the collection of vectors to be considered a vector space. Possibly the most important of these requirements is closure under addition and multiplication, meaning that the vector which results from adding two vectors together or multiplying a vector by a scalar will also belong to our space.

Given some data, a feature space is just the set of all possible values for a chosen set of features from that data. To reduce the dimensionality of the feature space, a number of dimensionality reduction techniques can be employed. Perhaps the most important take away is that the feature space can be interpreted as a real space. One of the many reasons that this spatial representation of our data is useful is that it allows us to introduce an idea of distance and therefore similarity and dissimilarity into our data.

Let's visualize this phenomenon by working with the Iris data set in R. The data set has 4 features or dimensions and 1 class variable. First we will look at just one dimension. Sepal.Length, has a range of 4.3 to 7.9. The data density is found by dividing the number of observations by the range, 150/3.6. So each interval has 41.67 samples. To visualize the one-dimensional data, we use a stripchart.

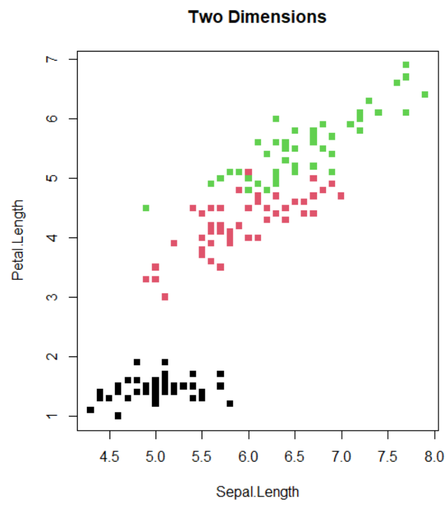
```
dim(iris)
[1] 150   5
r = range(Sepal.Length)[2] - range(Sepal.Length)[1]
#density number of observations/ feature space
density1D = 150/r
[1] 41.66667

stripchart(Sepal.Length ~ Species,
data = iris,
main = 'Sepal Length by Species',
xlab = 'Species',
ylab = 'Sepal Length',
col = c("black", "green", "red"),
pch = 16)
```

Note that in the image produced by this code, our data in this single dimension spans nearly every value in it's range.



Adding a 2nd dimension, Petal.Length, with a range of 5.9 units, the feature space increases to (5.9×3.6) or 21.24 units. The density of the data is 150/21.24 or 7.06 samples per interval. To visualize the two dimensions, we use the plot function.

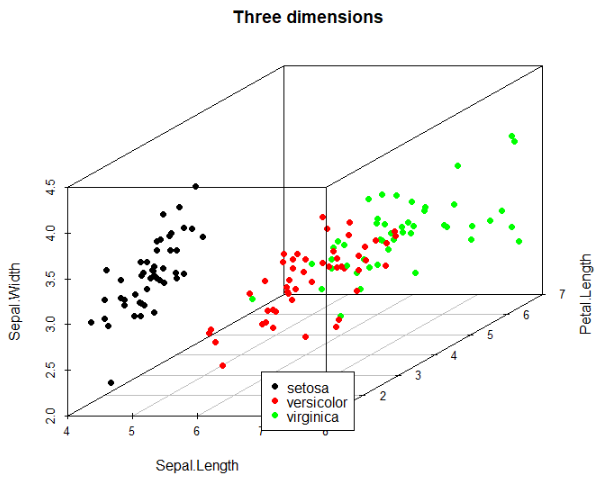
```
r2 = range(Petal.Length)[2] - range(Petal.Length)[1]
#density number of observations/ feature space
density2D = 150/(r*r2)
[1] 6.25
plot(Sepal.Length,Petal.Length,col = Species, pch = 15,
main = "Two Dimensions")
```



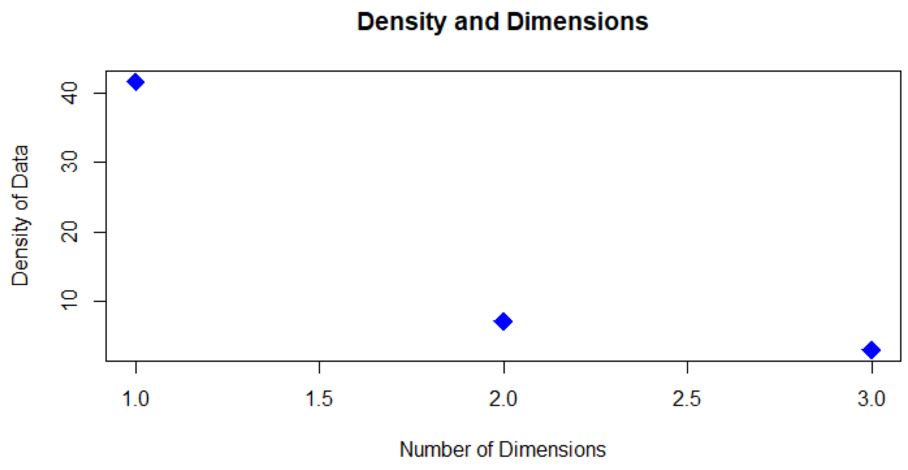
In this new space generated by our features, with Sepal.Length as the x-axis and Petal.Length as the y-axis, we can already see that this data takes up considerably less of the available two-dimensional space.

In three dimensions, the samples are further spread apart. By adding Sepal Width (the z-axis of our space), the range of which is 2.4, the feature space becomes $3.6 \times 5.9 \times 2.4 = 50.976$ units. As the data points become more spread out, the data density decreases to 2.083 samples per interval.

```
r3 = range(Sepal.Width)[2] - range(Sepal.Width)[1]

density3D = 150/(r*r2*r3)
[1] 2.942561

cols <- c("Black", "Red", "Green")

scatterplot3d(Sepal.Length, Petal.Length, Sepal.Width,
pch = 16, color=cols[as.numeric(Species)],
main = "Three dimensions")

legend("bottom", legend = levels(iris$Species),
col = c("Black", "Red", "Green"), pch = 16)
```



If we plot data density and number of dimensions, we can see that data density in the space decreases as the number of dimensions increase.



Linear separability is one benefit to working with high-dimensional data, and is referred to as the **blessing of dimensionality**. If we have two sets of points A, $B \in \mathbb{R}^n$ then, graphically, A and B are linearly separable if there exists a line, plane, or hyperplane that separates them (depending on the number of dimensions n of our data). Note that data can also be non-linearly separable.

Sometimes too much information can have a detrimental effect, resulting in a reduction in the effectiveness of data mining techniques. Some data attributes may not be contributing meaningful information to the model. While other features may have a negative effect on the quality and accuracy of the model. To minimize the effects of noise, correlation, and high dimensionality, some form of dimensionality reduction is necessary. Feature selection and feature extraction are two techniques to reduce the complexity of the data.

Feature selection refers to the process of selecting a subset of relevant features for use in model construction. The central premise when using a feature selection technique is that the data contains some features that are either redundant or irrelevant, that is, they can be removed without incurring much loss of information.

Redundant features duplicate some or all of the information contained in one or more other attributes. For example, the purchase price of a product and the amount of sales tax paid contain mostly the same information.

Irrelevant features contain almost no useful information and do not contribute to the predictive accuracy for the data mining task we are trying to accomplish. For example, a student's ID number is irrelevant for predicting their GPA.

Feature extraction creates new features from functions of the original features. This new reduced set of features should then be able to summarize most of the information contained in the original set of features. In this way, a summarized version of the original features can be created from a combination of the original set.

The difference between Feature Selection and Feature Extraction is that feature selection aims to rank the importance of the existing features in the data set and discard less important ones whereas feature extraction aims to create new features based on the originals.

FEATURE SELECTION

The three strategies for feature selection are: the Embedded approaches, Filter approaches, and Wrapper approaches.

Embedded approach. The feature selection occurs as part of the data mining algorithm. That is, feature selection is embedded into the process and the algorithm decides which attributes to use. Algorithms for building decision tree classifiers typically use an embedded approach. The embedded approach is dependent on the type of algorithm being run.

Filter approach. In this approach, feature selection happens before the data mining algorithm is run. Typically, in this approach features are selected by adhering to some set of criteria such as having low pairwise correlation to the other attributes or removed for adhering to some set of criteria such as having low variance. Filter methods use statistical methods for evaluation of a subset of features. Occasionally, filter methods will fail to return the best subset of features.

Wrapper approach. These methods allow the mining algorithm to act as a black box, extracting what it believes to be the best attributes. Wrapper methods can be subdivided into exhaustive search, heuristic search, and random search.

We will now cover a variety of Feature Selection techniques from the above approaches. First, let's discuss feature selection with a familiar approach–linear regression.

In the "mtcars" data, there are 12 features (x, mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb) and we want to predict the mpg (miles per gallon) then mpg becomes our response variable. Let's randomly select any of the predictor variables and try to fit the model for predicting mpg.

Starting with wt (weight):

```
data("mtcars")
mod1 <- lm(mpg ~ wt, data = mtcars)
summary(mod1)

The output:
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
Multiple R-squared:  0.7528
```

Our p-value for wt is $p = 1.29e - 10 \approx 0$. Then ith a significance level of $\alpha = 0.05$, we can reject the null hypothesis which allows us to conclude that there is evidence of a linear relationship between mpg and weight. Now let's fit the model with two variables wt and hp (horsepower).

```
mod2 <- lm(mpg ~ wt + hp, data = mtcars)
summary(mod2)
The output:
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
wt          -3.87783    0.63273  -6.129 1.12e-06 ***
hp          -0.03177    0.00903  -3.519  0.00145 **

Multiple R-squared:  0.8268
```

Our p-value for hp is $p = 0.00145$. Then with a significance level of $\alpha = 0.05$, we can reject the null hypothesis which allows us to conclude that there is evidence of a linear relationship between mpg and hp. Also note that, the $R^2 = 0.8268$, is an increase from the previous model with just weight. Hence this new model is more significant than the previous model. Recall that R-squared (R^2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. So, in our model above the R^2 is 0.8268, then approximately 82.68 percent of the observed variation can be explained by the model's inputs.

Usually, in fitting a linear regression model, we can simply consider all of the features at once and then decide which are most important for predicting our response by using hypothesis testing. This is ultimately a form of feature selection. What do when we have too many features to consider and the noise from the unimportant features causes us to not be able to find the most important features?

Consider the full model below.

```
mod.full <- lm(mpg ~ ., data = mtcars)
summary(mod.full)

The output:
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 12.30337   18.71788   0.657   0.5181
cyl         -0.11144    1.04502  -0.107   0.9161
disp         0.01334    0.01786   0.747   0.4635
hp          -0.02148    0.02177  -0.987   0.3350
drat         0.78711    1.63537   0.481   0.6353
wt          -3.71530    1.89441  -1.961   0.0633 .
qsec         0.82104    0.73084   1.123   0.2739
vs           0.31776    2.10451   0.151   0.8814
am           2.52023    2.05665   1.225   0.2340
gear         0.65541    1.49326   0.439   0.6652
carb        -0.19942    0.82875  -0.241   0.8122
```

Notice that in the full regression model, there are no significant predictors. It would be too time consuming to attempt to try all combinations of variables separately in order to see which are significant.

In **Stepwise Regression**, we start fitting the model with each individual predictor and see which one has the lowest p-value. The process starts with zero features and adds the one feature with the lowest significant p-value as described above. Then, it searches for and finds the second feature with the lowest significant p-value. On the third iteration, it will look for the next feature with the lowest significant p-value, and it will also remove any features that were previously added that now have an insignificant p-value.

In the MASS package, stepAIC() chooses the best model by the Akaike Information Criterion (AIC). AIC is a goodness of fit measure that favors smaller residual error in the model, but penalizes for including further predictors and helps avoid over fitting. Thus the best choice of model will balance fit with model size. It has an option named direction, which can take the following values: "both" (for Stepwise Regression, both Forward and Backward Selection); "backward" (for Backward Selection) and "forward" (for Forward Selection). It returns the best final model. We won't go too far into the complexities of AIC here but instead encourage the reader to look into it.

```
step.model <- stepAIC(mod.full, direction = "both",
                      trace = FALSE)
summary(step.model)

The output:
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   9.6178     6.9596   1.382 0.177915
wt           -3.9165     0.7112  -5.507 6.95e-06 ***
qsec          1.2259     0.2887   4.247 0.000216 ***
am            2.9358     1.4109   2.081 0.046716 *

Multiple R-squared:  0.8497
Residual standard error: 2.459
```

Notice that our R^2 value has increased to 0.8497. Thus, this is the model which explains the greatest proportion of the variance for the dependent variable that's explained by the independent variables. We should then accept this final model produced by Stepwise Regression, an easy-to-use method for finding the most important features in a linear regression model.

Moving forward, we have the option of using the package FSinR ("Feature Selection in R"), which contains functions to perform the feature selection process. More specifically, it contains a large number of widely used filter and wrapper methods that are combined with search algorithms in order to obtain an optimal subset of features.

```
install.packages("FSinR")
library("FSinR")
```

The feature selection process is done with the featureSelection function. This is the main function of the package and its parameters are:

1. Data: a matrix or data.frame with the data set.
2. Class: the name of the dependent variable.
3. Searcher: the search algorithm.
4. Evaluator: the evaluation method (filter or wrapper method).

The result of this function mainly returns the best subset of features found and the measure value of that subset. In addition, it returns another group of variables such as the execution time, etc.

The package FSinR also contains a function, searchAlgorithm, which allows you to select the search algorithm to be used in the feature selection process. The function consists of the following parameters:

1. Searcher: the name of the search algorithm.
2. Params: a list of specific parameters for each algorithm.

The result of the call to this function is another function to be used in the main function as a search algorithm.

Determination Coefficient method using FSinR
Recall that in statistics, the coefficient of determination R^2 is the proportion of the variance in the dependent variable that is predictable from the independent variables.

```
library(datasets)
library(caret)
# Load mtcars data
data("mtcars")
# Our evaluator  will be the
# 'determinationCoefficient' filter method
evaluator <- filterEvaluator('determinationCoefficient')
# Use the 'selectKBest' direct search algorithm
#  'k' features with the greatest evaluations are returned
directSearcher <- directSearchAlgorithm('selectKBest',
                                        list(k=3))
# mpg as the response
results <- directFeatureSelection(mtcars,
  'mpg',
  directSearcher,
  evaluator)
results
The output:
$featuresSelected
[1] "wt"   "cyl"  "disp"

$valuePerFeature
[1] 0.7528328 0.7261800 0.7183433
```

Note that this feature selection method finds the R^2 value for each feature in the data set when it is fitted with the response value, then ranks these values to choose the best features.

For example, the R^2 associated with the model $mod1 \leftarrow lm(mpg \sim wt, data = mtcars)$ is 0.7528, the R^2 associated with the model $mod2 \leftarrow lm(mpg \sim cyl, data =$

$mtcars$) is 0.7262, and the R^2 associated with the model $mod3 \leftarrow lm(mpg \sim disp, data = mtcars)$ is 0.7183.

```
results$bestFeatures
The output:
     cyl disp hp drat wt qsec vs am gear carb
[1,]   1    1  0    0  1    0  0  0    0    0

# Compare
mod4 <- lm(mpg ~ wt + disp + cyl, data = mtcars)
summary(mod4)

The output:
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 41.107678   2.842426  14.462 1.62e-14 ***
wt          -3.635677   1.040138  -3.495  0.00160 **
disp         0.007473   0.011845   0.631  0.53322
cyl         -1.784944   0.607110  -2.940  0.00651 **
Multiple R-squared:  0.8326
Residual standard error: 2.595
```

Hence, the function found that the best $k = 3$ features for predicting the response variable 'mpg' are the features 'cyl', 'disp', and 'wt'.

Why did the two feature selection methods we used on the 'mtcars' data return different results? We must note here that linear regression assumes that the independent variables are, as described, independent of each other. Using just the coefficient of determination filter method does not account for collinearity of our independent variables. That is why, when we create a linear model with the best features returned by the coefficient of determination filter method, the variable 'disp' is no longer significant–even though the R^2 value for this model is still quite high. In the third step of the Stepwise Regression algorithm, we removed any features that were previously added that now have an insignificant p-value. In this feature selection method, we did not account for this.

LASSO Regression.
The Least Absolute Shrinkage and Selection Operator (LASSO) method is a technique which performs two main tasks: regularization and feature selection. The method shrinks the coefficients of the regression model as part of penalization. For feature selection, the variables which are left after the shrinkage process are used in the model. The goal then is to minimize the prediction error. LASSO reduces the dimensionality of our data by using an embedded feature selection technique, recall that embedded methods perform feature selection as part of the model construction process.

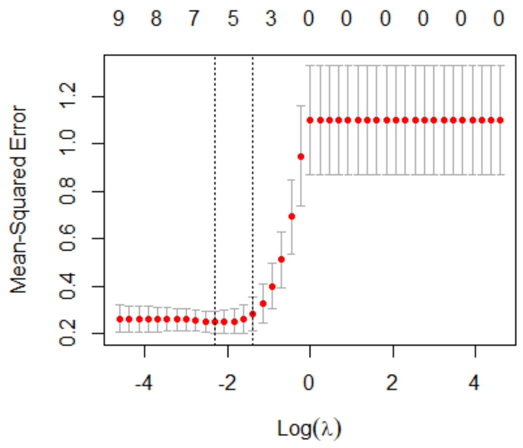
The parameter λ controls the strength of the penalty. When λ is large then coefficients are forced to be exactly equal to zero, this way dimensionality can be reduced. The larger the parameter λ is, the more coefficients are shrunken down to zero. On the other hand if $\lambda = 0$ then we just have typical linear regression with no penalty. LASSO helps to increase the model interpretability by eliminating irrelevant features that are not associated with the response variable, this way the chance of over-fitting is small.

We need to identify the optimal λ value and then use that value to train the model. To achieve this, we can use the *glmnet()* function with alpha = 1 for LASSO regression. When we pass alpha = 0, *glmnet()* runs a Ridge regression, and if we let alpha = 0.5, then *glmnet()* runs the Elastic Net model which is a combination of Ridge and LASSO regression.

```
library(glmnet)

mtcars <- scale(mtcars)

# split into training and test sets
n = length(mtcars[,1])
# number of training examples
n.train = round(n*0.8)
# indices of training samples
train.ind = sample(n,n.train)
train.cars = mtcars[train.ind,]
test.cars = mtcars[-train.ind,]
```

```
# create some sequence of lambdas to test
lambda_seq <- 10^seq(2, -2, by = -.1)

# Note that cv.glmnet requires our predictors as
# a matrix rather than a data frame
cv_output <- cv.glmnet(as.matrix(train.cars[ ,-1]),
   train.cars[,1],
                       alpha = 1, lambda = lambda_seq,
                       nfolds = 5)

# identifying best lamda
best_lam <- cv_output$lambda.min
best_lam
The output:
[1] 0.1
```

We can then plot the cross-validation curve (red dotted line), and upper and lower standard deviation curves along the λ sequence (error bars). Two selected λ values are indicated by the vertical dotted lines. Note that $log(0.1) = -1$.



Now, we can use this optimal lambda to re-train our model and find which variables were most important. That is, which variables did not get shrunken to zero or close to zero. Note that the "·" means that the coefficient of the variable is zero.

```
# Rebuilding the model with best lamda value identified
lasso_best <- glmnet(as.matrix(train.cars[ ,-1]),
 train.cars[,1], alpha = 1, lambda = best_lam)

coef(lasso_best)
The output:
11 x 1 sparse Matrix of class "dgCMatrix"
                     s0
(Intercept)  5.43026517
cyl           .
disp          .
hp          -0.01088949
drat         1.41428195
wt          -2.78712747
qsec         1.09732634
vs            .
am           1.54482405
gear         0.15498375
carb        -0.15202964
```

Hence, picking the features whose coefficients have the greatest magnitude compared to the others, the model picked the features "wt", "am", "drat", and "qsec" as the most important predictors. LASSO has reduced the dimensionality of our data from 10 to 4 of the most important features. Please note that this result is very similar to our result with the Stepwise Regression method.

Sequential Feature Selection.
Sequential Forward Selection (SFS), a special case of sequential feature selection, is a greedy search algorithm that attempts to find the optimal feature subset by iteratively selecting features based on the classifier performance. We start with an empty feature subset and add one feature at a time in each round; that is, one feature is selected from the pool of all features that are not in our feature subset, and it is the feature that when added results in the best classifier or prediction performance. SFS is a wrapper method which uses a heuristic search for feature selection.

Let's find the most important features in the wine cultivar classification data set we used earlier in this course. We will use K-Nearest Neighbors as the wrapper evaluator and Sequential Forward Selection as the search algorithm.

```
wine <- read.csv(
  "C:/wine.csv", header =TRUE
)
evaluator <- wrapperEvaluator("knn")
searcher <- searchAlgorithm('sequentialForwardSelection')
results <- featureSelection(wine, 'Cultivar',
directSearcher, evaluator)
results$bestFeatures

The output:
     Alcohol MalicAcid Ash Alcalinity Magnesium
[1,]       0         0   0          0         0
     TotalPhenols Flavanoids
[1,]            1          1
     NonflavanoidPhenols Proanthocyanins
[1,]                   0               0
     ColorIntensity Hue OD280OD315 Proline
[1,]              0   0          1       0
```

The best classifiers for the wine cultivar are then the variables with 1 below the feature name. Thus, this wrapper method reduced the number of features in our model from 13 to 3 of the most important features.

Feature extraction, as stated before, is the process of linearly transforming the data to create a new set of features which summarize the critical information contained within the original set. Two such techniques are Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). PCA is an unsupervised technique, a technique that finds the directions of maximal variance. In contrast, LDA is a supervised method that attempts to find a feature subspace that maximizes class separability.

FEATURE EXTRACTION

Principal Component Analysis.

Principal Component Analysis (PCA) is a technique for reducing the dimension of a $n \times p$ data matrix X where the data is centered on the means of each variable. PCA allows us to summarize a large set of correlated variables with a smaller number of representative variables explaining the majority of the variability in the original set. An unsupervised approach, PCA uses the feature set $X_1, X_2, \cdots, X_p$ without an associated response vector Y. That is, the PCA procedure is "non-dependent". PCA is most suitable when variables have a linear relationship among them.

PCA is based on a singular value decomposition (SVD) of the data matrix X into two matrices V and U where $X = U\Sigma V^T$. The two matrices V and U are orthogonal, or uncorrelated. The matrix V is usually called the loadings matrix, and the matrix U is called the scores matrix. The loadings can be understood as the weights for each original variable when calculating the principal component. The matrix U contains the original data in a rotated coordinate system. The matrix Σ is the diagonal matrix of singular values of X, where singular values are the square roots of non-negative eigenvalues.

We will illustrate PCA on the Iris data set. As a reminder, the data set contains 3 classes with 50 observations each, the class variable refers to a species of irises, and 4 feature variables of measurements from each sample: the length and the width of the sepals and petals, in centimeters.

```
# a function which centers and scales our data matrix
center_scale <- function(x) {
  x <- scale(x, scale = TRUE)                                   }
# x is our data matrix
x <- iris[,1:4]
# center and scale the data matrix
x <- center_scale(x)
# response vector
y <- as.matrix(as.numeric(iris[,5]))
```

```
# Obtain the SVD
S <- svd(x)

# d is the vector of diagonal entries of the matrix Sigma
# (the non-negative singular values of x)
d <- S$d
> d
[1] 20.853205 11.670070  4.676192  1.756847

# see that the sqrt of the eigenvalues
# are equal to the singular values
eigen.vals <- eigen(t(x)%*%x)$values
sqrt(eigen.vals)
> sqrt(eigen.vals)
[1] 20.853205 11.670070  4.676192  1.756847
```

In the context of PCA, an eigenvector represents a direction or axis. Each eigenvector has a corresponding eigenvalue which represents variance along that eigenvector. The greater the eigenvalue, the greater the variance along that eigenvector will be. Eigenvalues are constrained to decrease monotonically–notice this trend above. The first two eigenvalues are much greater than the other two–that is, they explain more of the variance. These eigenvalues are commonly plotted on a scree plot, which we will demonstrate, to show the decreasing rate at which variance is explained by additional principal components.

PCA seeks a linear combination of variables such that the maximum variance is extracted from the variables. It then removes this variance and seeks a second linear combination which explains the maximum proportion of the remaining variance, and so on. This is called the principal axis method and results in the orthogonal (uncorrelated) components described above. The principal components are then linear combinations of the original variables weighted by their contribution to explaining the variance in a particular orthogonal dimension. The goal is dimension reduction and there is no guarantee that the dimensions are interpretable.

The first principal component is equal to the following normalized linear combination of the features

$$Y_1 = \phi_{11}X_1 + \phi_{12}X_2 + \cdots + \phi_{1p}X_p$$

The first component is dominant and has the largest amount of variance. Why? We prevent the first principal component from explaining too much of the variance by requiring that the sum of the squares of the weights is equal to one. That is,

$$\sum_{i=1}^{p} \phi_{1i}^2 = 1$$

We calculate the second principal component in the same way, with the condition that it is uncorrelated with the first principal component and that it accounts for the next highest variance. This continues until a total of p principal components are calculated.

We will first perform PCA by using the function *prcomp()*. By default the *prcomp()* function centers variables to have a mean of 0. Using scale = T, each variable is scaled to have a variance of 1.

```
iris.pc <- prcomp(iris[,-5],center = T,scale. = T)
summary(iris.pc)
The output:
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7084 0.9560 0.38309 0.14393
Proportion of Variance 0.7296 0.2285 0.03669 0.00518
Cumulative Proportion  0.7296 0.9581 0.99482 1.00000
```

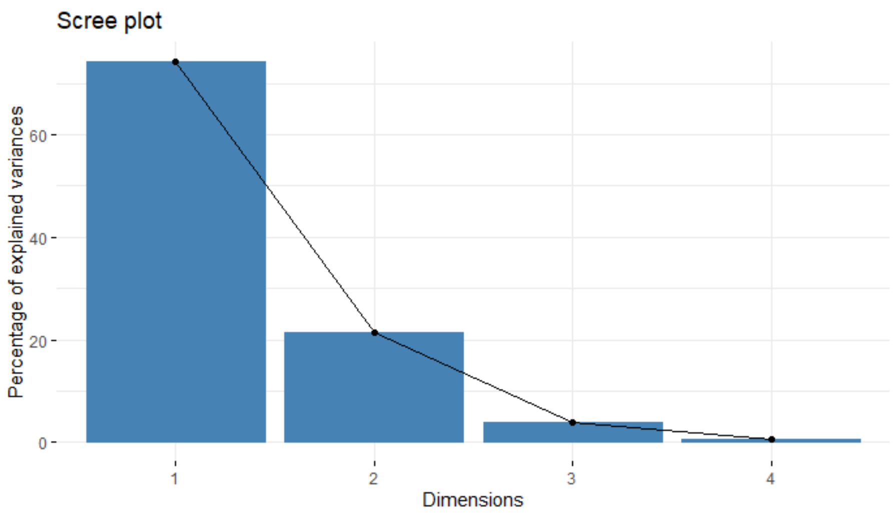
The rotation matrix gives the principal components loadings, each column contains the corresponding principal component loading vector.

```
> iris.pc$rotation
                    PC1         PC2        PC3        PC4
Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

We can easily create a visualization of the variance explained by each principal component by using the “factoextra” package.

```
install.packages("factoextra")
library(factoextra)
fviz_eig(iris.pc)
```

From the scree plot on the following page, we can conclude that the highest proportion of variance is explained by the first two principal components. Note that the variance explained decreases as the number of dimensions increase. Additionally, we should employ the “elbow” method for deciding the optimal number of principal components.



The optimal number of principal components is decided by where the elbow of the curve above is. In this case, the optimal number of principal components is two. Thus, we have successfully reduced the dimensionality of our model from 4 to 2 features which are a linear combination of our original features from the data set.

Now the new features, components, are given by

$$p_1 = 0.521X_1 - 0.269X_2 + 0.58X_3 + 0.565X_4$$

and

$$p_2 = -0.377X_1 - 0.923X_2 - 0.024X_3 - 0.067X_4$$

where p_1 is the first component and p_2 is the second.

The *prcomp()* function will output the standard deviation of each principal component. The variance explained by each component is found by squaring the these values. To compute the proportion of variance explained by each component, we would divide the variance explained by the total variance explained. The first principal component explains 73% of the variance in the data.

```
varExp = iris.pc$sdev^2
[1] 2.93883571 0.88704012 0.15279249 0.02133168
pve = varExp/sum(varExp)
[1] 0.73470893 0.22176003 0.03819812 0.00533292
```

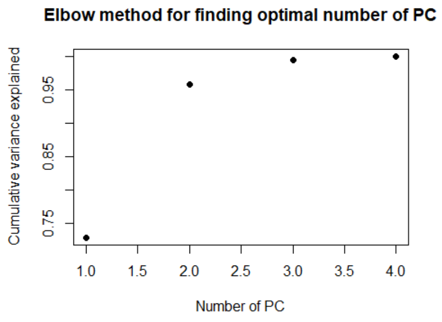
The sum of the variances of all of the principal components will equal the sum of the variances of all of the variables, that is, all of the original information has been explained or accounted for by these new variables, the principal components. The first two principal components explain over 95 percent of the variance.

Now, let us demonstrate using eigenvectors. Recall from before that in the context of PCA, an eigenvector represents a direction or axis. Each eigenvector has a corresponding eigenvalue which represents variance along that eigenvector. We saw before that the first two eigenvalues were the greatest. The principal components are eigenvectors of the data's covariance matrix. Thus, the principal components are often computed by eigendecomposition of the data covariance matrix or singular value decomposition of the data matrix.

```
# find orthonormal set of eigenvectors
pval <- svd(t(x)%*%x)$v
> pval
           [,1]        [,2]       [,3]       [,4]
[1,] -0.5210659 -0.37741762  0.7195664  0.2612863
[2,]  0.2693474 -0.92329566 -0.2443818 -0.1235096
[3,] -0.5804131 -0.02449161 -0.1421264 -0.8014492
[4,] -0.5648565 -0.06694199 -0.6342727  0.5235971
```

Note that the variable 'pval' is our rotation matrix from the *prcomp()* function, each column corresponds to the principal component loading vector. Furthermore, we could also compute the optimal number of principal components using matrix multiplication.

```
# data matrix obtained by using the transformed covariates
w <- x%*%pval
# column variances
cvars <- apply(w,2,var)
tot_var_exp <- sum(cvars)
# ratios of variance explained for each principal comp
ratio <- cvars/tot_var_exp
# cumulitive sums of the variance explained ratios
csums <- cumsum(ratio)
plot(1:length(csums), csums, pch = 19)
# get where the components start explaining >= 95% of variance
k <- which(csums >= 0.95)[1]
> k
[1] 2
```



Linear Discriminant Analysis.
Linear Discriminant Analysis, LDA, also referred to as Fisher linear discriminant, is a linear classifier and feature reduction technique using linear transformations. A supervised method, LDA finds a feature subspace maximizing class separability by making a distance comparison in the space spanned by the class means.

Both LDA and PCA are linear transformation techniques. PCA is a feature extraction method which finds the directions of maximal variance, while LDA attempts to find a feature subspace that maximizes class separability.

One assumption of LDA is the p-dimensional random variable X has a multivariate Gaussian distribution with E[X] = μ, and the Cov(X)= Σ is the $p \times p$ covariance matrix. When LDA is used as a classifier this requirement is necessary; although when using LDA for feature reduction, the normality assumption can be violated.

Similiar to PCA, LDA uses the eigendecomposition to derive coefficients of a score function for each class. These functions, after scaling the numeric predictor variables according to the class specific coefficients, outputs a score. The LDA model looks at the score from each function and selects the highest score of an observation to assign it to a class. These scoring functions are referred to as the discriminant functions.

We would classify an observation x to class j, if it has the highest log-likelihood among k classes if, for i = 1,...k:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

The above function is called the discriminant function. This tells us how likely it is that data x is from some class k. The decision boundary separating the two classes k and l, is the set of x where two discriminant functions have the same value. If a data point was to fall on this line, then the data is equally likely to come from both classes. In essence, this defines a decision boundary by maximizing the distance between the means of the projected data from each class while minimizing the within-class variance.

In situations where the Σ_k are not equal, a quadratic term remains in the equation and we must use Quadratic Discriminant Analysis, QDA.

We will illustrate dimensionality reduction using LDA with the lda function on the Iris data set. The lda function is in the MASS library. A call to lda contains a formula, data, and a prior. The prior argument sets the prior probabilities of class membership. These should be specified in order of the factor levels. When omitted from the argument, the class proportions from the training set are used.

After loading the library, the numeric data will need to scaled and centered before calling lda.

```
library(MASS)
iris = iris
attach(iris)
scale(iris[,-5], center = TRUE)
fit.lda.iris = lda(Species ~ ., data = iris,
prior = c(1,1,1)/3)
> fit.lda.iris
The output:
Call:
lda(Species ~ ., data = iris, prior = c(1, 1, 1)/3)
Prior probabilities of groups:
    setosa versicolor  virginica
 0.3333333  0.3333333  0.3333333
Group means:
           Sepal.Length Sepal.Width Petal.Length Petal.Width
setosa            5.006       3.428        1.462       0.246
versicolor        5.936       2.770        4.260       1.326
virginica         6.588       2.974        5.552       2.026
Coefficients of linear discriminants:
                    LD1         LD2
Sepal.Length  0.8293776  0.02410215
Sepal.Width   1.5344731  2.16452123
Petal.Length -2.2012117 -0.93192121
Petal.Width  -2.8104603  2.83918785
Proportion of trace:
   LD1    LD2
0.9912 0.0088
```

A call to lda returns the prior probability of each class, the counts for each class, the class-specific means for each covariate, the linear combination coefficients for each linear discriminant and the singular values, giving the ratio of the between-group and within-group standard deviations of the linear discriminant variables. The singular values can be used to compute the amount of the between-group variance that is explained by each linear discriminant. One thing to note in this model, the first linear discriminant, LD1, explains more than 99% of the between-group variance in the iris data set.

We would now like to demonstrate how to perform LDA using eigenvectors. We first need two functions, one which finds the between-class scatter matrix and one to find the within-class scatter matrix. To do this we let

$$S_B = \sum_{k=1}^{C} N_i (m_k - m)(m_k - m)^T$$

$$S_W = \sum_{k=1}^{C} \sum_{x \in X}^{n} (x - m_k)(x - m_k)^T$$

where C is the number of classes, N_k is the sample size of the respective class, m_k is the class mean vector, and m is the global mean.

```
withinSS <- function(y, X){
        # how many observations
        nrx = nrow(X)
        # how many variables
        ncx = ncol(X)
        # class levels and number of levels
        clevs = levels(y)
        ng = nlevels(y)
        # within cov matrix
        Within = matrix(0, ncx, ncx)
        for (k in 1:ng){
                # select obs of k-th group
                tmp <- y == glevs[k]
                # mean k-th class
                mean_k = colMeans(X[tmp,])
                # center k-th class matrix
                Xk = scale(X[tmp,], center=mean_k,
                scale=FALSE)
                # sum the SS matrices
                Within = Within + t(Xk) %*% Xk
        }
        return(Within)
}
```

```
betweenSS <- function(y, X){
        ng = nlevels(y)
        # global means
        mean_all = colMeans(X)
        # matrix to store results
        Between = matrix(0, ncol(X), ncol(X))
        # calculate between Sum of squares
        for (k in 1:ng){
                tmp <- y == clevs[k]
                nk = sum(tmp)
                mean_k = colMeans(X[tmp,])
                dif_k = mean_k - mean_all
                between_k = nk * dif_k %*% t(dif_k)
                Between = Between + between_k
        }
        return(Between)
}
```

Now to compute the LDA projection, we will solve the generalized eigenvalue problem for the matrix $S_W^{-1}S_B$. The projection vector that has the highest eigenvalue provides higher discrimination power between classes.

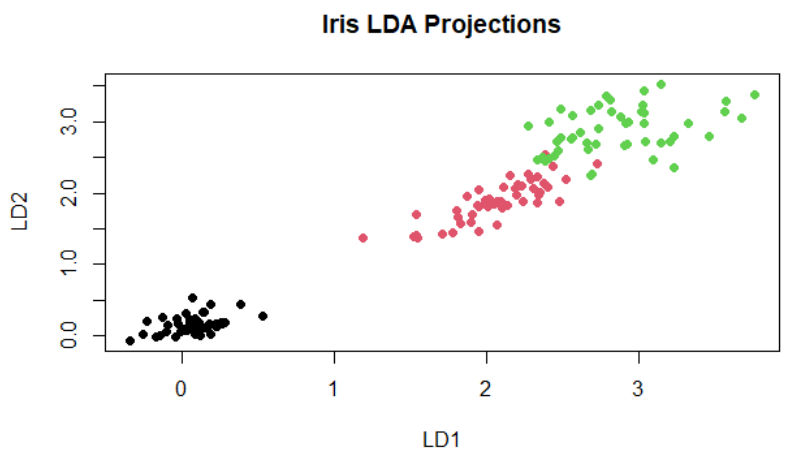
```
# within and between class scatter matrices
Sw = withinSS(iris.y, iris.x)
Sb = betweenSS(iris.y, iris.x)
# compute the LDA projection
invSw = solve(Sw)
invSwbySB = invSw*Sb
# find eigenvalues and corresponding eigenvectors
lda.iris = eigen(invSwbySB)
vecs = lda.iris$vectors
vals = lda.iris$values
vals
The output:
> vals
[1] 50.7211260 15.9898110  2.3559271  0.7856278
```

We choose the one with the highest value, i.e. the dominant eigenvector. These are our scaling factors. We will project onto the first two eigenvectors and transform the variables.

```
W = vecs[,1:2]
W
The output:
> W
            [,1]        [,2]
[1,] -0.21316400 -0.14988488
[2,] -0.03203418  0.05694058
[3,]  0.91159187  0.32486610
[4,] -0.35005026  0.93206991
```

Wow we use W to transform our samples onto the new subspace and plot our new feature subspace that we constructed via LDA.

```
iris = as.matrix(iris[,-5])
# transform our samples onto the new subspace
lda.iris = iris%*%W
plot(lda.iris, col = Species, pch = 19,
xlab = "LD1", ylab = "LD2",
main = "Iris LDA Projections")
```



Dimensionality, the number of features of a data set, can be difficult to work with as the number of attributes grows large. Without the use of a feature reduction or a feature selection technique as part of preprocessing, the curse of dimensionality will have a negative effect when data mining, leading to poor interpretability and inaccurate results. The use of feature selection algorithms such as LASSO regression, reduce the feature set to the important attributes and discards the rest. Feature extraction techniques, such as PCA, create a new set of features summarizing most of the information. In either case, the use of feature selection and feature extraction techniques will cause the interpretability of the model to increase, while the complexity and training time decreases.