Introduction to data science methods with derivation

Yifan Cheng

June 2023

Contents

1 Introduction

In today's society, with the rapid development of computer science and information technology, data science has become increasingly significant, especially in the field of finance. As an interdisciplinary field, data science integrates theories and methods from various areas such as statistics, machine learning, artificial intelligence, and computer science, providing us with effective tools to handle and comprehend vast amounts of data.

Particularly in the financial sector, the application of data science is continually deepening. Decision-makers in financial markets use data science methods to extract valuable information from massive financial data for essential tasks such as risk assessment, investment strategy formulation, and market trend forecasting. Furthermore, financial institutions, such as banks, investment firms, and insurance companies, increasingly rely on data science to optimize their business processes, enhance customer experience, and reduce operational costs.

Machine learning technologies have been widely used in various fields, and we can find related algorithm packages in many places and use different algorithms to complete tasks. Using machine learning functions in different programming languages (such as R, Python) is not a very complicated thing, because there are too many tutorials online. However, this article will delve into the derivation, motivation, and mathematical logic behind different algorithms, allowing readers to understand the mathematical background of different machine learning algorithms and their applications in practice.

2 Prediction Methods in Finance

In simple terms, profiting from trading, or "making money," is the goal of every financial practitioner. As I'm not a student of finance, I can't systematically elaborate on practical trading strategies. However, in my view, the essence of profiting from trading is arbitrage (buying low and selling high), making the prediction of financial market price trends a crucial aspect of quantitative trading. In this section, I will briefly introduce commonly used statistical models for prediction in practice.

2.1 Linear Regression Models

Linear regression and the closely related linear prediction theory are widely used statistical tools in empirical finance. We use linear regression models to reveal the statistical relation between response variable (usually denoted as Y) and predict variable(s) (usually denoted as X_p).

2.2 Methods of Least Square

A linear regression model relates the output (or response) y_t to p input (or predictor) variables x_{t1},...,x_{tp}, which are also called regressors, via $y_t = \beta_0 + \beta_1 x_{t1} + ... + \beta_p x_{tp} + \epsilon_t(1.1)$, note that all of α, β_{t1},$\ldots$,β_{tp} are unknown and need to be estimated.

To fit the linear regression model (1.1), the method of least squares choose parameter which minimize the residual sum of squares, also know as RSS,

$$\text{RSS} = \sum_{t=1}^{n} y_t - (\beta_0 + \beta_1 x_{t1} + ... + \beta_q x_{tq})^2$$

In other words, we can treat RSS as a criterion to determine the goodness of fit, and in general, lower RSS represents the regression line is more closed to the true value (it is equivalent to "low bias"). In addition, RSS can be one of the model selection criterions, but sometimes we may have overfitted problem when RSS is too low, and other criterions like R^2, R^2_{adj}, AIC, BIC may be better (discuss later).

Deriving the least square estimators is a simple process with optimization, we can use the method of taking derivative to find the extreme value.

Simple linear regression can be treated as one special case of multiple linear regression because in simple linear regression we only have one predictor and two estimators, so we can use one-dimensional calculus to find the value of β_0 and β_1

Simple linear regression:

$$y_t = \beta_0 + \beta_1 x_{t1} + \epsilon_t$$
$$\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x}))(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

For estimators above, we can also use the maximum likelihood estimation to derive the estimator and eventually we will have the same result.

$$\begin{gathered} L(\beta_0, \beta_1, \sigma^2) = \prod_{i=1}^{n} (\frac{1}{2\pi\sigma^2})^{\frac{1}{2}} exp(-\frac{(y_i - E(y_i))^2}{2\sigma^2}) \\ = (\frac{1}{2\pi\sigma^2})^{\frac{n}{2}} exp(-\frac{\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2}{2\sigma^2}) \\ log(L) = -\frac{n}{2} log(2\pi) - \frac{n}{2} log(\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \end{gathered}$$

Solving for $\hat{\beta_0}$:

$$\begin{gathered} \frac{\partial l}{\partial \beta_0} = \frac{\partial \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2}{\partial \beta_0} = 0 \\ \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) = 0 \\ \sum_{i=1}^{n} y_i - n\beta_0 - \sum_{i=1}^{n} \beta_1 x_i = 0 \\ \hat{\beta_0} = \bar{y} - \hat{\beta_1} \bar{x} \end{gathered}$$

Solving for $\hat{\beta_1}$:

$$\begin{gathered} \frac{\partial l}{\partial \beta_1} = \frac{\partial (\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2)}{\partial \beta_0} = 0 \\ \frac{\partial l}{\partial \beta_1} = \sum_{i=1}^{n} -2 \frac{1}{2\sigma^2} x_i (y_i - \beta_0 - \beta_1 x_i) = 0 \\ \sum_{i=1}^{n} x_i y_i - \hat{\beta_0} \sum_{i=1}^{n} x_i - \hat{\beta_1} \sum_{i=1}^{n} x_i^2 = 0 \\ \sum_{i=1}^{n} x_i y_i - (\bar{y} - \hat{\beta_1} \bar{x}) \sum_{i=1}^{n} x_i - \hat{\beta_1} \sum_{i=1}^{n} x_i^2 = 0 \\ \hat{\beta_1} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \end{gathered}$$

Now let's consider higher dimensions, $y_t = \alpha + \beta_1 x_{t1} + ... + \beta_p x_{tp} + \epsilon_t$, in matrix form, $Y = X\beta + E$, and the estimator of β is $\hat{\beta} = (X'X)^{-1}X'Y$. Note that $Y = X\beta + E$ shows the true relation between Y and X, for most of time we are not able to find the exact true relation, so we can use an estimator of Y: $\hat{Y} = X\hat{\beta}$ (unbiased), therefore we can write $\hat{Y} = X(X'X)^{-1}X'Y$, set $H = X(X'X)^{-}1X'$,we get $\hat{Y} = HY$. The derivation of estimator for multiple linear regression is also trivial, just required some algebra knowledge.

3 Metric of Fit

In our previous discussions, we've mentioned that predicting the market is a critical task when analyzing financial markets, making the construction of predictive models a significant endeavor in our analysis. Now, we're confronted with an obvious question: how do we evaluate the performance of the models we've built?

Supervised learning is about to estimate f under the generating mechanism: $Y = f(X) + \epsilon$, for example, the linear regression we have introduced in previous is a type of supervised learning algorithm, also we will introduce other unsupervised learning algorithms later but now we will focus on supervised learning. A distinctive feature of supervised learning is the provision of training and testing data sets (which is why it's termed "supervised"). Simply put, for a complete data set D, we can divide it into two parts: D^{train} and D^{test}. We use D^{train} to train our model, fit it, and then use the fitted model to compare with D_{test} to evaluate the performance of our model. Of course, not all steps in machine learning involve splitting the data into training and testing sets. We will also introduce other methods, such as cross-validation, later on.

Start with the regression problems where Y is quantitative, we have $Y = f(X) + \epsilon$, we estimate Y by Y hat where $\hat{Y} = f(\hat{X})$. For most of time, we are likely to see our prediction is closed to the true value. Ideally, we want to evaluate f hat by the expected mean squared error (MSE): $E[(Y - f(\hat{x}))^2]$, but be careful we focus more on the test MSE instead of training MSE.

3.1 Bias-Variance Trade-off

Bias and variance are two properties of the statistical model, and we use those two terms to discuss the performance of the fitted model. Variance means how much f hat would change if we estimated it using a different training data set; bias means the difference between fitted model and the true relation.

We can use the term flexibility or complexity to explain "variance", if f hat is less complex (like simple linear regression), the variance of $\hat{f}$ is usually small (figure 1); on the other hand, a complex model (like non-parametric model) usually has high variance (figure 2).

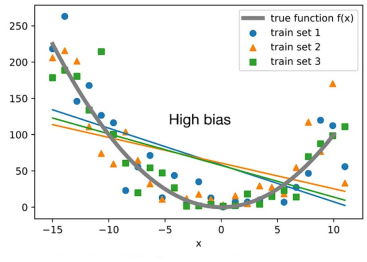


figure 1

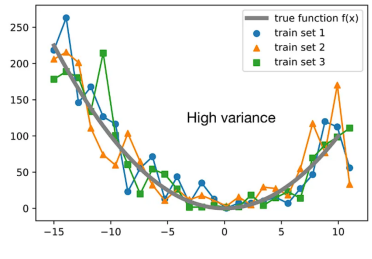


figure 2

Generally speaking, simpler models have lower complexity (like SLR), requiring less computational cost, meaning our computers can model different training data in a shorter amount of time. In contrast, complex models demand higher computational costs, thus requiring a longer modeling process or a better-performing graphics card. However, we should note a clear advantage of choosing complex models: they often can enhance the accuracy of predictions (reduce MSE).

Assume that we have the regression model $Y = f(x) + \epsilon$, where ϵ is independent of X and $E(\epsilon) = 0, E(\epsilon^2) = \sigma^2$. Assume that the training data (x1,y1),. . . ,(xn,yn) are used tot construct an estimate of f, denoted as $\hat{f}$. We can show that $E[(f(x) - \hat{f}(x))^2|X = x] = Var(\hat{f}(x)) + [E\hat{f}(x)–f(x)]^2$ and $E[(Y–\hat{f}(x))^2|X = x] = Var(\hat{f}(x)) + [E\hat{f}(x)–f(x)]^2 + \sigma^2$ (attached later see appendix 314 hw).
Here is the proof of the properties above:

$E[(f(X) - \hat{f}(X))^2|X = x]$
$= E[([f(x) - E(\hat{f}(x))] + [E(\hat{f}(x)) - \hat{f}(x)])^2]$
$= E[(f(x)-E(\hat{f}(x)))^2+(E(\hat{f}(x))-\hat{f}(x))^2+2(f(x)-E(\hat{f}(x)))(E(\hat{f}(x))-\hat{f}(x))]$
$= E[(E(\hat{f}(x)) - f(x))^2] + E[(E(\hat{f}(x)) - \hat{f}(x))^2]$
(because $E[E(\hat{f}(x)) - \hat{f}(x)] = 0$)

Here we have:

1) $E[(E(\hat{f}(x)) - f(x))^2] = E[(\hat{f}(x) - E(\hat{f}(x)))^2] = Var(\hat{f}(x))$ by definition.
2) consider $E[(E(\hat{f}(x)) - \hat{f}(x))^2]$, f(x) is a constant since our expectation is based on the given X=x, and $E(\hat{f}(x))$ is also a constant. Therefore we have $E[(f(x) - \hat{f}(x))^2|X = x] = [E(\hat{f}(x)) - f(x)]^2 + Var(\hat{f}(x))$

Now let's consider the Y instead of $f(X) + \epsilon$.

$E[(Y - \hat{f}(x))^2|X = x]$
$= E[(Y - f(x) + f(x) - \hat{f}(x))^2]$
$= E[(Y - f(x))^2 + (f(x) - \hat{f}(x))^2 + 2(f(x) - \hat{f}(x))(Y - f(x))]$
$= E[(Y - f(x))^2] + E[(f(x) - \hat{f}(x))^2] + 2E[(f(x) - \hat{f}(x))(Y - f(x))]$

Consider above separately:

1) $E[(Y - f(x))^2]$
$= E[(f(x) - f(x) + \epsilon)^2] = E[\epsilon^2] = \sigma^2$
2) $E[(f(x) - \hat{f}(x))^2]$
$= E[(f(X) - \hat{f}(X))^2|X = x] = [E(\hat{f}(x)) - f(x)]^2 + Var(\hat{f}(x))$
3) $E[(f(x) - \hat{f}(x))(Y - f(x))]$
$= E[f(x)Y - f(x)^2 - \hat{f}(x)Y + \hat{f}(x)f(x)]$
$= f(x)^2 - f(x)^2 - \hat{f}f(x) + f(x)\hat{f}(x) = 0$

Therefore we obtain that $E[(Y - \hat{f}(x))|X = x] = \sigma^2 + [E(\hat{f}(x)) - f(x)]^2 + Var(\hat{f}(x))$. In other word, we can conclude that the expected MSE = Irreducible error + squared bias + variance.

3.2 Model Selection

After constructing various models, we need to select the best one for further analysis. This raises a question: how do we judge the quality of a model? Ideally, the best criterion would be to calculate the prediction error of the model. However, this process isn't applicable in all cases: for example, when we don't have separated test and training data, or when the cost of calculating the error is too high. Therefore, we will introduce a series of selection criteria to aid us in model selection.

3.2.1 R^2

In linear regression, the first metric we learn to evaluate the quality of the model is R^2. R^2 represents the proportion of the variation in the outcome (Y) that can be explained by the predictors (X).

If we have a fitted model $\hat{y_i} = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \ldots + \hat{\beta}_p x_{ip}$, its residual is defined as $e_i = y_i – \hat{y_i}$, and residual sum of squares is $SS_{res} = \sum_{i=1}^{n} (y_i – \hat{y_i})^2$, the total sum of squares is $TSS = \sum i = 1^n (y_i – \bar{t})^2$, and we calculate $R^2 = (TSS – SS_{res})/TSS$. On the other hand, we can explain the R^2 by the source of variance: SS_{reg} means regression sum of squares and $SSreg = \sum i = 1^n (\hat{y} – \bar{y})^2$, which is the variation will be explained by our fitted line, SS_{res} means the residual sum of squares, or the unexplained variation, therefore, the $(TSS – SS_{res})/TSS$ represents how much variation can be explained by our model.

But in general, most of statistician will not treat R^2 as a good criterion: if we put more regressors into our fitted model, the R^2 is non-decreasing, so a large R^2 does not imply that the model fits the data well, and too many regressors may cause the overfitting problem. Here is the derivation about why R^2 is non-decreasing when new variable is added (appendix https://math.stackexchange.com/questions/1976747/prove-that-r2-cannot-decrease-when-adding-a-variable).

Suppose the original estimate by OLS is

$$y = X\beta + u \tag{1}$$

Now add a new variable X_0 and perform a new OLS estimate

$$y = X_0\hat{\beta}_0 + X\hat{\beta} + v \tag{2}$$

Combine (1) and (2), we get:

$$X\beta + u = X_0\hat{\beta}_0 + u^T v \tag{3}$$

Multiply both sides of (3) by u^T:

$$u^T u = u^T X_0 \hat{\beta}_0 + u^T v$$

Similarly, multiply both sides of (3) by v^T (see next page):

$$v^T u = v^T v$$

This tells us that

$$u^T X_0 \hat{\beta}_0 = u^T u - v^T v \tag{4}$$

Finally,

$$\begin{aligned}
v^T v &= (y - X_0\hat{\beta}_0 - X\hat{\beta})^T (y - X_0\hat{\beta}_0 - X\hat{\beta}) \\
&= (X\beta + u - X_0\hat{\beta}_0 - X\hat{\beta})^T (X\beta + u - X_0\hat{\beta}_0 - X\hat{\beta}) \qquad \text{from (1)} \\
&= (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0 + u)^T (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0 + u) \\
&= (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0)^T (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0) + 2u^T (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0) + u^T u \\
&= (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0)^T (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0) - 2u^T X_0\hat{\beta}_0 + u^T u \\
&= (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0)^T (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0) - 2(u^T u - v^T v) + u^T u \qquad \text{using (4)} \\
&= (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0)^T (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0) - u^T u + 2v^T v \\
u^T u &= (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0)^T (X(\beta - \hat{\beta}) - X_0\hat{\beta}_0) + v^T v \\
&\geq v^T v.
\end{aligned}$$

This says that the new SSE is less than the original SSE. It follows that R^2 must increase (weakly).

3.2.2 R^2_{adj}

As we continuously add more regressors, R^2 keeps increasing, a phenomenon that can easily lead to overfitting (usually, an overfit model will make precise predictions on training data but fail to accurately predict the test data). The essence of preventing overfitting lies in adding a penalty, and our new criterion is called R^2 adjusted (R^2_{adj}), which does not necessarily keep increasing as we add more regressors (thus increasing model complexity).
We have R^2_adj = 1-$\frac{[RSS/(n-p-1)]}{[TSS/(n-1)]}$, the advantage of using R^2_{adj} is that the R^2_{adj} may decrease when new parameters added.

3.2.3 AIC

The Akaike Information Criterion (AIC = -2 log L +2p) is a model selection metric used when testing on a standard set isn't feasible, such as with small data sets or time series. AIC evaluates a model's fit on training data, incorporating a penalty term for model complexity. The goal is to find the model with the lowest AIC, indicating the best balance of fit and generalizability. The formula for AIC involves the model's maximum log-likelihood (a measure of data fit) and the number of parameters (representing complexity).

Lower AIC values indicate higher log-likelihoods and better data fit, but a complexity penalty is added to discourage overfitting. Therefore, AIC assists in

identifying models that not only fit the training data well but are also likely to generalize effectively to new data.

3.2.4 BIC

The Bayesian Information Criterion, also known as BIC, is an essential component of the subjective Bayesian induction theory based on Bayesian decision-making. The formula for BIC is: BIC = ln(n)k - 2ln(L). It operates under incomplete information, using subjective probability estimates for partially unknown states.

The Bayesian formula is then applied to adjust the occurrence probabilities, and ultimately the expected values and adjusted probabilities are used to make the optimal decision. BIC is similar to AIC in practical perspective: the lower BIC means the model fits better, and the penalty of BIC is ln(n)k (larger than the penalty of AIC). When the sample size is too large, it can effectively prevent excessive model complexity caused by high model accuracy.

3.3 Data-splitting Techniques

As we mentioned before, when the dataset does not distinguish between training data and testing data, we can choose some metrics for model evaluation to judge the quality of the model. However, it should be noted that R^2, $R_a^2 dj$, AIC, BIC are evaluation criteria for regression analysis. In fact, the best and most direct model evaluation method is to calculate the MSE (Mean Squared Error), which is applicable to any algorithm or model, and we want the Test MSE to be as small as possible. When we only have a dataset, we can manually split it into a training set and a test set.

3.3.1 Validation Set Approach

In the Validation Set Approach, we randomly split the data into a training set and a test set, use the training set data for model building, and the test set data for calculating MSE. This method sounds like a good choice, but there's one thing we need to note: we are "randomly" splitting the data into a training set and a test set, and there is no set ratio for choosing the test set (or the training set), which can make our MSE very unstable: because MSE depends on the different datasets we divide.

3.3.2 Leave One Out Cross Validation

The instability of MSE in the validation set approach is due to our random selection of training data, which is very likely to result in different data used for training each time. An effective solution to this problem is to control the training data to be roughly the same each time: in our dataset, we select a data point as test data each time, and use all the remaining data as training data. We call this method LOOCV (leave one out cross validation), as the name suggests, that is, we repeat the selection of different test data points, and train the model with the remaining data. This ensures that the training sets in each repeat are basically the same, and it also ensures that the MSE is less likely to be unstable in different repeats.

However, we should note that LOOCV often faces the problem of high computational cost: when our data volume is too large (N data points), LOOCV requires us to perform N-1 calculations (sometimes this can be expensive). In this case, we can choose a method that balances computational cost and the stability of MSE: k-fold Cross Validation. K-fold CV uses a portion of available data to fit the model and then tests with different portions of data. We divide the data into K roughly equal parts; for instance, when K=5, the situation looks like this: for every five data points, we choose one data point as test data and use the remaining four data points for training.

4 Beyond Linear Regression

So far, almost all of our content has been centered around explaining linear regression in detail. We need to acknowledge that linear regression is easy to interpret and widely used, but we must also note that real-world data is complex and often multi-dimensional, implying that the true relationships may not necessarily be linear. Moreover, linear regression has its limitations; in other words, while linear models are simple, they also imply that they may not be able to accomplish the tasks we need them to. Now we will introduce some other methods which beyond the linear regression.

4.1 Regularized Linear Regression

Denote the loss function of linear regression as L(w) = $\sum_{i=1}^{N} ||w^T x_i - y_i||^2$, according to the OLS, we have the $\hat{w} = (X^T X)^{-1} X^T Y$. Here we have a process with inverse, but we need to be careful that some matrices may not exist inverse. For example, assume we have matrix X ($N \times P$), if P is extremely large (P > N), $X^T X$ does not have inverse and we can't get the analytical solution for w hat (that is the reason why we need to have the regularization).

On the other hand, mathematically we call it "inverse does not exist", and in practical machine learning, we have a term "overfitting" to describe (to demonstrate it, we can consider an extreme case: if we only have 1 data point, there will be infinite ways to fit it). To deal with overfitting problem, regularized method is one possible solution, and we will go through solving overfitting problem in future concepts.
The frame of regularization is $\arg\min_w$[L(w) + λ P(w)], where L is the loss function and P is the penalty.

4.1.1 Ridge Regression

Ridge regression is one of the regularized linear regression will be introduced, and we have $p(w) = ||w||_2^2 = w^T w$. Consider

$$\begin{aligned} J(w) &= \sum_{i=1}^{N} ||w^T x_i - y_i||^2 + \lambda w^T w \\ &= (w^T x^T - Y^T)(xw - Y) + \lambda w^t w \end{aligned}$$

by some algebra manipulations.
Continue:

$$\begin{aligned} J(w) &= w^t x^t x w - 2 w^t x^t Y + Y^T Y + \lambda w^t \\ w &= w^T (x^T x + \lambda I) w - 2 w^T x^T Y + Y^T Y \end{aligned}$$

we need to find $\hat{w} = \arg\min_w J(w)$, by taking derivative, we solve
$\hat{w} = (x^T x + \lambda I)^{-1} x^T Y$, note that here the matrix is non-singular.

On the other side, we can interpret ridge regression as Bayesian perspective.

We assume that w $N(0, \sigma_0^2)$ as a prior, and $p(w|y) = \frac{p(y|w)p(w)}{p(y)}$ which p(y) is a constant by Bayes's theorem. Based on the MAP (maximum posterior), we have $\hat{w} = \arg\min_w p(w|y) = \arg\min_w p(y|w)p(w)$. Since we have $f(w) = w^T x$, $y = f(w) + epsilon = w^T x + \epsilon$ which ϵ $N(0, \sigma^2)$, so the distribution of $y|xw$ $N(w^T x, \sigma^2)$, then we will have $p(y|w) = \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{-\frac{(y-w^T x)^2}{2\sigma_0^2}}$, and $p(w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. Continue on solving $\hat{w}_{MAP}$,

$$\begin{aligned}
\hat{w} &= \arg\max_w log[p(y|w)p(w)] \\
&= \arg\max_w \frac{-(y-w^T x)^2}{2\sigma^2} - \frac{||w||^2}{2\sigma_0^2} \\
&= \arg\min_w \frac{(y-w^t x)^2}{2\sigma^2} + \frac{||w||^2}{2\sigma_0^2} \\
&= \arg\min_w (y - w^T x)^2 + \frac{\sigma^2}{\sigma_0^2}||w||_2^2
\end{aligned}$$

Therefore we obtain the $\hat{w}_{MAP} = \arg\max_W \sum_{i=1}^N (y_i – w^T x_i)^2 + \frac{\sigma^2}{\sigma_0^2}||w||_2^2$. In fact, it is same as the definition of estimator of ridge regression: loss function + constant * penalty. In conclusion, we will get the same estimator no matter based on least square estimator or maximum posterior, Regularized LSE <=> MAP (noise follows Gaussian distribution)

4.1.2 Another Regularization: Lasso

We call both ridge and lasso regression are shrinkage methods, and the shrinkage some estimates towards 0 by the penalty term or also called the shrinkage term. By choosing different penalty, we will have different shrinkage method. For example, we choose L_2 norm in Ridge regression, we can also consider the L1 norm, and we call it Lasso. Similar to ridge regression, the lasso shrinks the coefficient estimates towards 0, but in the case of lasso, the L_1 penalty has the effect of forcing some of coefficient estimates to be exact zero (note that in Ridge regression, we just shrinkage estimates towards 0 not exact 0), so we can perform the variables selection via lasso. To explain why lasso can get exact 0, we need some tedious math to find w hat which similar to the derivation of Ridge regression in previous. After taking the derivative, we will have $\hat{w} = (2x^T x)^{-1}(2x^T y – \lambda I)$ by lasso penalty. Compare to ridge regression: in ridge regression, the w hat will be 0 only if lambda $-> \infty$ and therefore the value of β will be as low as possible but will not become zero; but in lasso, the w hat will be 0 when subtracting some value of lambda (i.e. hyperparameter).

4.2 Optimization in Machine Learning Problems

In all the previous content, we introduced some basic machine learning algorithms from a statistical perspective, as well as some parameter solving methods. At the same time, we can simply summarize the problem of parameter solving as a mathematical problem: finding the extreme value. For example, in linear regression, our parameters are solved by ordinary least squares, which is equivalent to finding the minimum value of the loss function. In this way, we have transformed the problem of finding beta in machine learning models into an optimization problem.

An important type of optimization problem in operations research is convex optimization, which consists of a convex set (constraints), a convex function (objective function), and an objective function with a zero first derivative. The solutions to the objective function in machine learning mainly use convex optimization methods: for example, linear regression is linear programming, which is a type of convex optimization; for example, common support vector machines (SVM) and logistic regression are non-linear programming where both the objective function and condition functions are convex, which also belong to convex optimization. There are two main types of convex optimization algorithms commonly used in machine learning: Gradient Descent and Newton Methods.

4.2.1 Gradient Descent

Imagine you are on a mountain peak, without considering other factors, how should you walk to get to the foot of the mountain as quickly as possible?

Of course, you choose the steepest place. This is also the core idea of the gradient descent method: it gradually approaches the minimum value of the function by taking a step forward in the current gradient direction (the steepest direction) each time.

Assume that in the n_{th} iteration, we have $\theta_n = \theta_{n-1} + \triangle\theta$, then $L(\theta_n) = L(\theta_{n-1} + \triangle\theta) \approx L(\theta_{n-1}) + L'(\theta_{n-1})\triangle\theta$ based on the Taylor expansion, then $\theta_n = \theta_{n-1} - \alpha L'(\theta_{n-1})$, note that we use L represents the loss function, but sometimes we use J(w) as the function we want to optimize which is not exactly same as the loss function.

The advantages and disadvantages of gradient descent are quite clear:

Advantages: It requires the calculation of the first derivative, the operation is simple, and the computation is small. When the loss function is a convex function, it can ensure convergence to a relatively good global optimum.

Disadvantages: α is a fixed value, the quality of convergence is directly influenced by it. If it's too small, the convergence will be too slow; if it's too

large, it will cause oscillations and fail to converge to the optimal solution. For non-convex problems, it can only converge to a local optimum and has no ability to escape from local optima because once the gradient is zero, there will be no further changes.

Here is an example of using gradient descent to find the parameter for Ridge regression: https://stackoverflow.com/questions/65909753/gradient-descent-for-ridge-regression

4.2.2 Coordinate Descent Method

As we discussed earlier, the objective function usually incorporates L_1 regularization and L_2 regularization to prevent overfitting or constrain parameters. Now, let's focus on L1 regularization, also known as Lasso regularization, because Lasso can perform feature selection and dimensionality reduction. For the objective function, we first determine whether it's Smooth or Non-Smooth. If it's a Smooth function, it is differentiable and we can use gradient descent or Adam optimization to find the optimal value. If it's a Non-Smooth function, it's non-differentiable and it's not as straightforward to find the optimal value. The Lasso regularization term is the sum of the absolute values of the parameters, which makes the objective function Non-Smooth. In other words, optimization algorithms like gradient descent don't work with it.

So, how can we find the optimal solution for the parameters? Let's introduce a new method for finding the extremal values: Coordinate Descent Method.

For the optimization problem in Lasso, the task is: $\min_\beta \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1$ and we set the dimensions such as $y \in R^n, X \in R^{n\times p}, \beta \in R^p$.

Now make the loss function as $l(\beta) = l(\beta_1,\ ...,\ \beta_p)$, such as:

$$\begin{aligned} f(\beta) &= \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1 \\ &= \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| \end{aligned}$$

We need to find the optimized β_k such as $\min_{\beta_k} f\left(\beta_1, \cdots, \beta_{k-1}, \beta_k, \beta_{k+1}, \cdots, \beta_p\right)$ and we just find the partial derivative (next page):

$$\begin{aligned} \frac{\partial f(\beta)}{\partial \beta_k} &= -\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)x_{ik} + \frac{\partial\left(\lambda|\beta_k|\right)}{\partial\beta_k} \\ &= -\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} + \sum_{i=1}^{n} x_{ik}^2\beta_k + \frac{\partial\left(\lambda|\beta_k|\right)}{\partial\beta_k} \end{aligned}$$

For the term $\frac{\partial(\lambda|\beta_k|)}{\partial\beta_k}$, we need to introduce a little about subgradient (Subderivtive):

$$\frac{\partial(\lambda|\beta_k|)}{\partial\beta_k} = \begin{cases} \lambda & \beta_k > 0 \\ [-\lambda, \lambda] & \beta_k = 0 \\ -\lambda & \beta_k < 0 \end{cases}$$

Using the definition of subgradient above, we can solve the partial derivative of $f(\beta)$:

$$\frac{\partial f(\beta)}{\partial\beta_k} =$$
$$\begin{cases} -\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)x_{ik} + \sum_{i=1}^{n} x_{ik}^2\beta_k + \lambda & \beta_k > 0 \\ \left[-\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} - \lambda, -\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} + \lambda\right] & \beta_k = 0 \\ -\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} + \sum_{i=1}^{n} x_{ik}^2\beta_k - \lambda & \beta_k < 0 \end{cases}$$

Therefore, we will find the extreme value β_k^* by setting $\frac{\partial f(\beta)}{\partial\beta_k} = 0$, but need to realize that $\frac{\partial f(\beta)}{\partial\beta_k}$ is a piecewise function.

$r_k = \sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik}, z_k = \sum_{i=1}^{n} x_{ik}^2$
When $\beta_k > 0$:

$$-\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} + \sum_{i=1}^{n} x_{ik}^2\beta_k^* + \lambda = 0$$
$$\beta_k^* = \frac{\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} - \lambda}{\sum_{i=1}^{n} x_{ik}^2}$$
$$\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} > \lambda \text{ since } \beta_k > 0$$

When $\beta_k < 0$:

$$-\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} + \sum_{i=1}^{n} x_{ik}^2\beta_k - \lambda = 0$$
$$\beta_k^* = \frac{\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} + \lambda}{\sum_{i=1}^{n} x_{ik}^2}$$
$$\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} < -\lambda \text{ since } \beta_k < 0$$

When $\beta_k = 0$:

$$0 \in \left[-\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} - \lambda, -\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} + \lambda\right]$$
$$-\lambda \leq \sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} \leq \lambda$$

In conclusion, we find β_k^*:

$$\beta_k^* = \begin{cases} \frac{\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} - \lambda}{\sum_{i=1}^{n} x_{ik}^2} & \sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} > \lambda \\ 0 & -\lambda \leq \sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} \leq \lambda \\ \frac{\sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} + \lambda}{\sum_{i=1}^{n} x_{ik}^2} & \sum_{i=1}^{n}\left(y_i - \sum_{j\neq k} x_{ij}\beta_j\right)x_{ik} < -\lambda \end{cases}$$

4.2.3 Stochastic Gradient Descent

In the previous content, we introduced the method of gradient descent. When the scale of the data is relatively small, we can calculate the gradient on all the data and update iteratively. However, when the data scale is large, the overhead of calculating the gradient on all the data each time will be very huge. Therefore, stochastic gradient descent, which can greatly reduce the computation overhead, is often used in the optimization of large-scale data.

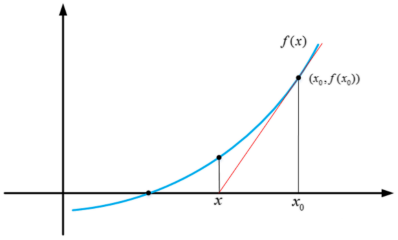
Consider an optimization problem: minimize $\frac{1}{m}\sum_{i=1}^{m} f_i(x)$. For gradient descent method, the algorithm will find the minimum in each iteration like $x^{(k)} = x^{(k-1)} - t_k \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x_{(k-1)})$, k = 1, 2, 3, ...

We can also consider the example: objective function $J(x) = \frac{1}{n}\sum_{i=1}^{n} J(x_i)$, $J(x_i)$ is the i^{th} objective function, then the objective function has the gradient at x is $\nabla J(x) = \frac{1}{n}\nabla\sum_{i=1}^{n} J(x_i)$. If we use the gradient descent method, then we will calculate the gradient n times per iteration. The stochastic gradient descent method only uses i_{th} sample $J(x_i)$ to update the parameter, therefore the stochastic gradient will reduce the computation costs from O(n) to O(1).

4.2.4 Newton Raphson Method

Essentially, Newton's method converges in a second-order manner, while gradient descent converges in a first-order manner, which makes Newton's method faster. To put it more colloquially, if you want to find the shortest path to the bottom of a basin, gradient descent takes a step in the direction of the steepest slope from your current position each time, while Newton's method not only considers the steepness of the slope when choosing a direction, but also whether the slope will become steeper after taking a step. (figure newton)

Figure Newton



Based on the figure above, we want to find the root of the function: $f(x) = 0$. First we randomly pick x_0, and take its tangent, so the tangent will intersects with x-axis at x_1, then repeat taking tangents and find its intersections of x-axis until it converges to the root.

Formally,

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1}$$
$$(x_0 - x_1)f'(x_0) = f(x_0)$$

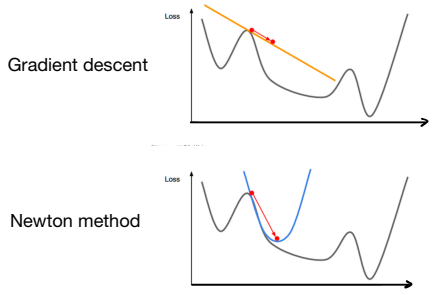
$$x_0 - x_1 = \frac{f(x_0)}{f'(x_0)}$$
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$
$$\dots$$
$$x_t + 1 = x_t - \frac{f(x_t)}{f'(x_t)}$$

Be more formal and rigours, we need to use Hessian matrix and Taylor expansion.

Gradient descent uses the first derivative of the function, and our candidate point moves in the negative gradient direction. Through an iterative process, we can find the extremum of the function. Newton's method is a method that uses the second derivative and Taylor expansion, which converges faster than gradient descent. First, we need to understand whether the direction of Newton's method is a descent direction. Here we only consider the simple Newton method so we do not consider the impact of step size (assume step size =1).

Mathematical derivation

The gradient descent algorithm approximates the function at position x_n with a first-order function, which is a straight line. It computes the gradient to determine the optimization direction for the next step, which is the opposite direction of the gradient. On the other hand, Newton's method approximates the function at position x_n with a second-order function, which is a quadratic curve. It computes the gradient and the second derivative to determine the optimization direction for the next step. The illustrations of first-order optimization and second-order optimization are as shown below (figure newton vs gd).



4.3 Bayesian Linear Regression

Recall the linear regression: we have the basic model as $f(x) = w^T x$, $y = f(x) + \epsilon = w^T x + \epsilon$.

Based on the frequentist perspective, w is an unknown constant, which we convert the linear regression to an optimization problem (like the argmin and argmax in previous).

In least square estimation, we use MLE for solving parameter s.t.

$W_{MLE} = \arg\max_w P(Data|w)$.

To deal with the overfitting problem, we need to consider the regularized LSE (includes Ridge and Lasso). For regularized LSE, we need to maximize its posterior such as $W_{MAP} = \arg\max_w P(w|Data)$. Both LSE and regularized LSE are point estimator (optimization problem).

4.3.1 Bayesian Methods

Here we do not continue on the frequntist perspective: w is not an unknown constant and it turns to be a random variable. Thereofre, we need to find the posterior as $P(w|Data)$ and this process is not point estimator (not an optimization problem).

Before we get started, let's clarify some notations.
Data: $\{(x_i, y_i)\}_{i=1}^N$, $x_i \in R^p$, $y_i \in R$.

$$X = (x_1\ x_2\ ...\ x_N)^T = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Np} \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Our model is quite simple: $f(x) = w^T x$, $y = f(x) + \epsilon$, and $x,\ y,\ \epsilon$ are random variables, $\epsilon \sim N(0, \sigma^2)$.

Basically, the Bayesian method contains two parts: inference (find the posterior, w is not a constant so we need to find its distribution) and prediction.

4.3.2 Inference:$P(w|Data)$

$P(w|Data) = P(w|X, Y) = \frac{P(X,Y|X)}{P(Y|X)} = \frac{P(Y|w,X)P(w)}{\int P(Y|w,X)P(w)dw}$, notice that $P(w, Y|X) = P(Y|w, x)P(w|X)$, and $P(w|X) = p(w)$ because X has no impact on w.

Since $\epsilon \sim N(0, \sigma^2)$, we can conclude $P(y|x, w) = N(w^T x, \sigma^2)$, then $P(Y|w, X) = \prod_{i=1}^N P(y_i|w, x_i) = \prod_{i=1}^N N(y_i|w^T x_i, \sigma^2)$.

Consider the prior, we assume $P(w) = N(0, \Sigma_p)$ (conjugate prior has some convenience in convergence, we can also choose other prior such as uniform distribution).

Therefore, $P(w|Data) \propto P(Y|w, X)P(w) \propto \prod_{i=1}^N N(y_i|w^T x_i, \sigma^2) \cdot N(0, \Sigma_p)$. Since $P(Y|w, X)$ and $P(w)$ both follows Gaussian distribution, so we can guarantee that $P(w|Data)$ is also Gaussian distributed, denote as $P(w|Data) = N(\mu_w, \Sigma_w)$.

Consider the likelihood:

$$P(Y|X,w) = \prod_{i=1}^{N} \frac{1}{(2\pi)^{\frac{1}{2}}\sigma} exp(-\frac{1}{2\sigma^2}(y_i - w^T x_i)^2)$$
$$= \frac{1}{(2\pi)^{\frac{N}{2}}\sigma^N} exp(-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - w^T x_i)^2)$$

Convert the inside summation expression to matrix:

$$= \frac{1}{(2\pi)^{\frac{N}{2}}\sigma^N} exp(-\frac{1}{2}(Y - Xw)^T\sigma^{-2} \cdot I(Y - Xw))$$
$$= N(Xw, \sigma^2 I) \text{ by Multivariate Gaussian distribution.}$$

Therefore $P(w|Data) \propto N(Xw, \sigma^2 I) \cdot N(0, \Sigma_p)$ and the next step of the inference is to solve μ_w and Σ_w.

$P(w|Data) \propto exp(-\frac{1}{2}(Y - Xw)^T\sigma^{-2} \cdot I(Y - Xw)) \cdot exp(-\frac{1}{2}w^T\Sigma_p^{-1}w)$ (we only consider the term related to w)

$$= exp(-\frac{1}{2\sigma^2}(Y^T - w^T X^T)(Y - Xw) - \frac{1}{2}w^T\Sigma_p^{-1}w)$$
$$= exp(-\frac{1}{2\sigma^2}(Y^T Y - 2Y^T Xw + w^T X^T Xw)) - \frac{1}{2}w^T\Sigma_p^{-1}w)$$

To solve the parameters we liked, one method is using complete square.

Brief introduction to complete square:
If we have the Gaussian distribution like $P(x) = N(\mu, \Sigma)$ and consider the exponential part: $exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$, which is equal to $exp(-\frac{1}{2}x^T\Sigma^{-1}x - 2\mu^T\Sigma^{-1}x + C)$ C is the constant we do not need to concern. $x^T\Sigma^{-1}x$ is the 2^{nd} order term and $-2\mu^T\Sigma^{-1}x$ is the 1^{st} order term.

Now look back $P(w|Data) = exp(-\frac{1}{2\sigma^2}(Y^T Y - 2Y^T Xw + w^T X^T Xw)) - \frac{1}{2}w^T\Sigma_p^{-1}w)$, the 2^{nd} order term of w is $-\frac{1}{2\sigma^2}w^T X^T xw - \frac{1}{2}w^T\Sigma_P^{-1}w = -\frac{1}{2}(w^T(\sigma^{-2}X^T X + \Sigma_p^{-1})w)$, and we can define $(\sigma^{-2}X^T X + \Sigma_p^{-1})$ as $\Sigma_w^{-1} = A$.

The 1^{st} order term is $-\frac{1}{2\sigma^2} \cdot (-2)Y^T Xw = \sigma^{-2}Y^T Xw$, so we define $\sigma^{-2}Y^T X$ as $\mu_w^T\Sigma_w^{-1}$ (notice that all procedures are corresponded to the completing square above). Since $\sigma^{-2}Y^T X = \mu_w^T\Sigma_w^{-1}$ and we defined $A = \Sigma_w^{-1}$:

$$A\mu_w = \sigma^{-2}X^T Y$$
(we take transpose both sides and A is symmetric)
$$\mu_w = \sigma^{-2}A^{-1}X^T Y$$

4.3.3 Prediction by Bayesian Linear Regression

Consider the new case: given the data x^* and y^*, and recall our model is $f(x) = w^T x$ with $y = f(x) + \epsilon$. Then, we can easily find $f(x^*)$:

$$f(x^*) = x^{*T} w$$

but we need to be careful that w is not a constant, and we can learn w as the posterior such as $P(w|Data) = N(\mu_w, \Sigma_w)$ as we discussed early. According to the linear combination of Gaussian distribution, we can write:

$$x^{*T} w \sim N(x^{*T} \mu_w, x^{*T} \Sigma_w x^*)$$

In other words, we have $P(f(x^*)|Data, x^*) = N(x^{*T} \mu_w, x^{*T} \Sigma_w x^*)$.

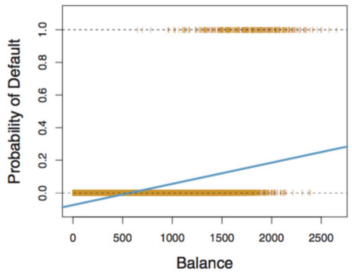
We can treat the prediction above as the noise free model because we only consider the x, so now let's look at y^*, and $y^* = f(x^*) + \epsilon$. Similarly, we still use the Gaussian distribution to find $P(y^*|Data, x^*)$ since $\epsilon \sim N(0, \sigma^2)$. Using the result we find about $f(x^*)$ in previous, we have $P(y^*|Data, x^*) = N(x^{*T} \mu_w, x^{*T} \Sigma_w x^* + \sigma^2)$.

5 Methods for Solving (linear) Classification Problems

In all of our previous discussions, we have actually only explored one important task in machine learning: regression, or prediction. One thing to note is that in real life, we not only have quantitative data, but also categorical data. For categorical data, we usually convert it into a "yes or no" question. For example, in real life, if we want to predict whether a company will pay dividends at the end of the year, we can collect historical data from other companies of similar size (or in the same field), record dividends as 1, and no dividends as 0. This way, we have transformed it into a classification problem.

The most important step in regression analysis is fitting, and the fitted model should capture as much variance as possible. In classification problems, we can also consider fitting models, but using linear models here is not suitable: in classification problems, we only consider values of 0 or 1, and other values are not within our consideration. For example, if we establish a linear model for the dataset that needs to be classified, we would end up with meaningless results (figure 3).

Figure 3



5.1 Logistic Regression

Logistic regression is a parametric approach that assumes parametric structure on

$$P(x) = P(y = 1|X) = E(y|x)$$
$$with\ P(x) = \frac{exp(\beta_0+\beta_1 x_1+\ldots+\beta_p x_p)}{1+exp(\beta_0+\beta_1 x_1+\ldots+\beta_p x_p)}$$
and β can be estimated by mle.

Here is an example of building logistic regression model. We can see the result looks good but there is the warning such as:

"Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred"

We will discuss why the warning is here later. (R code)

Visit the data set

Here I use R to build the logistic regression model for the clients' credit card data.

First I read the csv file and remove null value.

Second I split the data by training set and test set.

Then I fit the logistic model and show the confusion matrix.

One thing need to be aware: although the confusion matrix shows that our model is good, but the warning is not neglectable because maybe our data is not good for performing a reasonable logistic regression which I will discuss later about overfitting problem.

5.2 Why not Logistic Regression?

Logistic regression is good because it is easy to interpret, and it will provide accurate prediction or classification for most of cases. We need to be aware that logistic regression will perform well conditionally.

In classification problems, when the classes are well-separated, the parameter estimates for the logistic regression model are surprisingly unstable. Maybe this statement makes you confused, now let us see some obvious examples.

We know the shape of the logistic regression curve is like an S-shape curve between 0 and 1 (like figure 4). When the classes are well separated, the fitted line will no longer be S-shape which is unstable (figure 5).

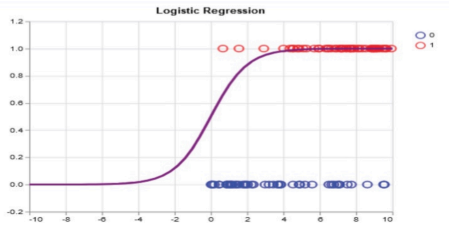
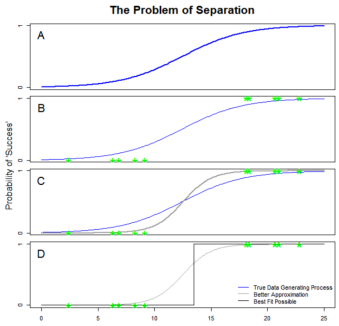


Figure 4

Figure 5



Well-separated classes is also known as complete separation, let us explain this term by the survivors from Titanic. Let Y be the dependent binary variable which indicates whether the passenger survived (0 indicates death and 1 indicates survival). Let X be the separating variable which indicates the class of passengers on Titanic. We say the complete separation is the situation where X predicts all values of Y, that would be the case if all first-class passengers on the Titanic had survived the wreckage, and none of the second-class passengers had survived (just an example).

Under complete separation, your logistic model is going to look for a logistic curve that assigns, for example, all probabilities of DV to 1 when SV=1, and all probabilities to DV to 0 when SV=0

This corresponds to the aforementioned situation where only and all first-class passengers of the Titanic survive, with SV=1 indicating first-class passenger membership. This is problematic because the logistic curve lies strictly between 0 and 1, which means that, to model the observed data, the maximisation is going to push some of its terms towards infinity. Since the logistic regression curve lies strictly between zero and one, this likelihood cannot be achieved, only approached asymptotically as the coefficients beta approaches infinity.

5.3 Discriminant Analysis

Discriminant analysis method provides the linear classification, the parametric assumption is assumed on P(X=x—Y=y), and the discriminant analysis will not suffer from the well separated classes like logistic regression and it will be ore suitable for multi-class classification.

5.3.1 Linear Discriminant Analysis

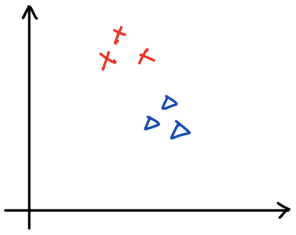


Figure 6

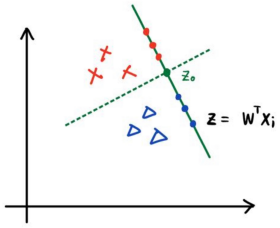


Figure 7

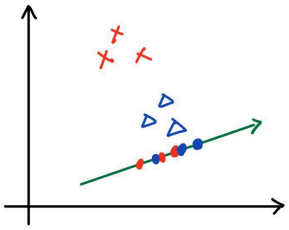


Figure 8

Begin with a simple classification question (figure 6), we define red and blue are two classes, and we are required to make a linear threshold for classification. Here we have two ways to project our data point into a plane (figure 7, figure 8). In figure 7, it is very easy to approximate a suitable linear threshold can separate our data points completely: let all data points project to plane w, for projection points are greater than z0, we classify it as red, otherwise, we classify it as blue. In figure 8, it is impossible to find a single threshold which can separate two classes. In other words, we expect to find a projection which can make the small intraclass distance and large interclass distance.

Be more formal, let X be N samples, $X = (x_1, x_2, \ldots, x_N)^T$ (N*P), $Y = (y_1, y_2, \ldots, y_N)$ (column vector) be N corresponding classes, $xi \in R^p$, and $y_i \in \{+1, -1\}$, here y=+1 represents it belongs to c_1, -1 represents it belongs to c_2. Furthermore define $x_{c1} = \{\text{xi—y=+1}\}$ and $x_{c2} = \{x_i | y = -1\}, |x_{c1}| = N_1, |x_{c2}| = N_2, N_1 + N_2 = N$.

Let X project to w (also set $||w|| = 1$), then we get $z_i = w^T x_i$ for each projection, and $\bar{z} = \frac{1}{N}\sum_{i=1}^{N} z_i = \frac{1}{N}\sum_{i=1}^{N} w^T x_i$. The covariance matrix of z (S_z) will be

$$\begin{aligned} &\tfrac{1}{N}\textstyle\sum_i^N (z_i - \bar{z})(z_i - \bar{z})^T \\ &= \tfrac{1}{N}\textstyle\sum_i^N (w^T x_i - \bar{z})(w^T x_i - \bar{z})^T. \end{aligned}$$

Consider the binary classification c_1 and c_2,
for c_1:

$$\begin{aligned} \bar{z_1} &= \tfrac{1}{N_1}\textstyle\sum_{i=1}^{N_1} w^T x_i \\ s_1 &= \tfrac{1}{N_1}\textstyle\sum_{i=1}^{N_1} (w^T x_i - \bar{z_1})(w^T x_i - \bar{z_1})^T; \end{aligned}$$

for c_2:

$$\bar{z_2} = \frac{1}{N_2}\sum_{i=1}^{N_2} w^T x_i,$$
$$s_2 = \frac{1}{N_2}\sum_{i=1}^{N_2}(w^T x_i - \bar{z_2})(w^T x_i - \bar{z_2})^T$$

As we claim before, the core of linear discriminant analysis is "small intraclass distance and large interclass distance", so here we define interclass distance as $s_1 + s_2$ and intraclass as $(\bar{z_1} - \bar{z_2})^2$.

Therefore we can define an objective function: $J(w) = \frac{(\bar{z_1} - \bar{z_2})^2}{s_1 + s_2}$ for linear discriminant analysis to achieve small intraclass distance and large interclass distance, $\hat{w} = \arg\max_w J(w)$.

Consider the numerator,

$$\begin{aligned}(\bar{z_1} - \bar{z_2})^2 &= (\frac{1}{N_1}\sum_i^{N_1} w^T x_i - \bar{z_2})^2 \\ &= \frac{1}{N2}\sum_{i=1}^{N_2} w^T x_i \\ &= (w^T(\frac{1}{N_1}\sum_{i=1}^{N_1} x_i - \frac{1}{N_2}\sum_{i=1} N_2 x_i))^2 \\ &= w^T(\bar{x_{c1}} - \bar{x_{c2}})(\bar{x_{c1}} - \bar{x_{c2}})^T w.\end{aligned}$$

Consider the denominator,

$$\begin{aligned}s_1 &= \frac{1}{N_1}\sum_{i=1}^{N_1}(w^T x_i - \frac{1}{N_1}\sum_{j=1}^{N_1} w^T x_j)(w^T x_i - \frac{1}{N_1}\sum_{j=1}^{N_1} w^T x_j)^T \\ &= \frac{1}{N1}\sum_{i=1}^{N_1} w^T(x_i - \bar{x_{c1}})(x_i - \bar{x_{c1}})^T\end{aligned}$$

Notice that

$$\begin{aligned}w &= w^T[\frac{1}{N_1}\sum_{i=1}^{N_1}(xi - \bar{x_{c1}})(x_i - \bar{x_{c1}})^T] \\ w &= w^T s_{c1} w\end{aligned}$$

Similar to s_1, we can define $s2 = w^T s_{c2} w$. Then the denominator is $w^T(s_{c1} + s_{c2})w$, and $J(w) = \frac{w^T(\bar{x_{c1}} - \bar{x_{c2}})(\bar{x_{c1}} - \bar{x_{c2}})^T w}{w^T(s_{c1} + s_{c2})w}$

5.3.2 Generative Model

In linear classification problems, our goal is to classify the input data into specified categories. Typically, there are two types of outputs for linear classification: hard classifiers and soft classifiers. A "hard classifier" provides a final category label as output, which is the model's decision on the category to which a data point belongs. For example, in a binary classification problem, the hard output could be either 0 or 1. A "soft classifier," on the other hand, provides the probabilities or scores of a data point belonging to each category. For instance,

in a binary classification problem, the soft output could be the probability of a data point belonging to category 1, like 0.7 (with the probability of belonging to category 0 being 0.3).

Now we will discuss the soft classifier, and soft classifier contains two subsections: probabilistic discriminant model (such as logistic regression), and probabilistic generative model (such as Gaussian Discriminant Analysis and Navie Bayes).

In general, probabilistic discriminant model directly find $P(y = 1|x)$ and $P(y = 0|x)$ (for binary case. In generative model, we use Bayes' theorem $P(y|x) = \frac{P(x|y)P(y)}{P(x)}$, and compare $P(y = 0|x)$ and $P(y = 1|x)$. We know $P(y|x)$ is proportional to $P(x|y)P(y) = P(x, y)$, therefore the main goal of generative model is to parameterize the $P(x, y)$, and define $P(y|x)$ as posterior, p(x—y) as likelihood, and p(y) as prior. In generative model, $\hat{y} = \arg\max_{y \in \{0,1\}} p(y|x) = \arg\max_{y} p(y)p(x|y)$.

5.3.3 Gaussian Discriminant Analysis

Assume we have the data $\{(xi, yi)\}_{i=1}^{N}$, we have $x_i \in R^p$ and $y_i \in \{0, 1\}$.

In other words, $y \sim Bernuolli(\phi)$, and $x|y = 1 \sim N(\mu_1, \sigma), x|y = 0 \sim N(\mu_2, \Sigma)$, note that here we assume different μ and same Σ. The reason is quite simple, we choose the same covariance matrix Σ solely because the covariance matrices of each Gaussian distribution are originally identical. Note the definition of the covariance matrix of an n-dimensional random variable X (as a natural generalization of covariance): $\Sigma = (\sigma_{ij})_{n \times n};\ \sigma_{ij} = Cov(X_i, X_j);\ i, j = 1, \cdots n.$

Therefore, the covariance matrix Σ depends only on all the features that have been selected in the model (these features are expressed as the valued vector random variable X). Since it contains all the selected features, it is naturally the same Σ.

To solve the $\hat{y}$ as we mentioned before for the generative model, we firstly consider the log likelihood:

$l(\theta) = log \prod_{i=1}^{N} P(x_i, y_i)$
$= \sum_{i=1}^{N} logP(x_i|y_i)p(y_i)$
$= \sum_{i=1}^{N} [logp(x_i|y_i) + logp(y_i)]$
$= \sum_{i=1}^{N} [logN(\mu_1, \Sigma)^{y_i} N(\mu_2, \Sigma)^{1-y_i} + log\phi^{y_i}(1 - \phi)^{1-y_i}]$
$= \sum_{i=1}^{N} [logN(\mu_1, \Sigma)^{y_i} + logN(\mu_2, \Sigma)^{1-y_i} + log\phi^{y_i}(1 - \phi)^{1-y_i}]$

Therefore, we use θ to perform all the needed parameters, $theta = \{\mu_1, \mu_2, \Sigma, \phi\}$, and $\hat{\theta} = \arg\max_{\theta} l(\theta)$. Now we are going to solve all four parameters.

Solving for ϕ:

$$\begin{aligned}\frac{\partial}{\partial\phi} l(\theta) &= \sum_{i=1}^{N} y_i \frac{1}{\phi} + (1-y_i)\frac{1}{(1-\phi)}(-1) \\ &= \sum_{i=1}^{N} y_i \frac{1}{\phi} - (1-y_i)\frac{1}{(1-\phi)} = 0\end{aligned}$$

Then we can solve $\hat{\phi} = \frac{1}{N}\sum_{i=1}^{N} y_i = \frac{N_1}{N}$ (N_1 is the number of y=1).

Solving for μ_1:

$$\begin{aligned}&\frac{\partial}{\partial\mu_1} l(\theta) = \frac{\partial}{\partial\mu_1} \sum_{i=1}^{N} logN(\mu_1, \sum)^{y_i} \\ &= \frac{d\partial}{\partial\mu_1} \sum_{i=1}^{N} y_i log \frac{1}{(2\pi)^{\frac{p}{2}} |\sum|^{\frac{1}{2}}} exp(-\frac{1}{2}(x_i-\mu_1)^T \sum^{-1}(x_i-\mu_1)).\end{aligned}$$

Therefore we obtain:

$$\mu_1 = \frac{\sum_{i=1}^{N} y_i x_i}{\sum_{i=1}^{N} y_i} = \frac{1}{N_1}\sum_{i=1}^{N_1} x_i$$

Solving for $\sum$:
First we need to know some important facts:

$$\begin{gathered}\frac{\partial tr(AB)}{\partial A} = B^T \\ \frac{\partial |A|}{\partial A} = |A|A^{-1} \\ tr(AB) = tr(BA) \\ tr(ABC) = tr(CAB) = tr(BCA)\end{gathered}$$

Look at the expression of $l(\theta)$, we can see there are only first two terms contain the Σ, for simplicity, we define:

$$\begin{aligned}\Delta &= \sum_{i=1}^{N} (\log N(\mu_1, \Sigma)^{y_i} + \log N(\mu_2, \Sigma)^{1-y_i}) \\ &= \sum_{i=1}^{N} (y_i \log N(\mu_1, \Sigma) + (1-y_i)\log N(\mu_2, \Sigma))\end{aligned}$$

Because y_i can only be 1 and so:

$$\begin{gathered}\sum_{i=1}^{N} y_i \log N(\mu_1, \Sigma) = \sum_{i=1}^{N_1} \log N(\mu_1, \Sigma) \\ \sum_{i=1}^{N} (1-y_i)\log N(\mu_2, \Sigma) = \sum_{i=1}^{N_2} \log N(\mu_2, \Sigma)\end{gathered}$$

Therefore we can write δ as $\Delta = \sum_{i=1}^{N_1} \log N(\mu_1, \Sigma) + \sum_{i=1}^{N_2} \log N(\mu_2, \Sigma))$
.

First we solve the $\sum_{i=1}^{N} \log N(\mu, \Sigma)$:

$$\sum_{i=1}^{N} \log N(\mu, \Sigma) = \sum_{i=1}^{N} \log(\frac{1}{(2\pi)^{\frac{p}{2}} |\Sigma|^{\frac{1}{2}}} \exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu))) \tag{5}$$

$$= \sum_{i=1}^{N} \log(\frac{1}{(2\pi)^{\frac{p}{2}}}) + \log|\Sigma|^{-\frac{1}{2}} - \frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu) \tag{6}$$

Remove the constant term: $\sum_{i=1}^{N} \log(\frac{1}{(2\pi)^{\frac{p}{2}}}) = \text{C}$.

Notice that $(x-\mu)^T\Sigma^{-1}(x-\mu)$ has dimensions: $1\times p \quad p\times p \quad p\times 1$ separately, so the result will be a real number.

Therefore it can be expressed as $tr((x-\mu)^T\Sigma^{-1}(x-\mu))$.

Next we use the fourth important fact:

$$tr((x-\mu)^T\Sigma^{-1}(x-\mu)) = tr((x-\mu)(x-\mu)^T\Sigma^{-1})$$

then

$$\sum_{i=1}^{N} tr((x-\mu)(x-\mu)^T\Sigma^{-1}) = tr(\sum_{i=1}^{N}(x-\mu)(x-\mu)^T\Sigma^{-1})$$
$$tr(\sum_{i=1}^{N}(x-\mu)(x-\mu)^T\Sigma^{-1}) = tr(\{\sum_{i=1}^{N}(x-\mu)(x-\mu)^T\}\Sigma^{-1})$$

and $\sum_{i=1}^{N}(x-\mu)(x-\mu)^T$ is equivalent to N times variance S.

So $\sum_{i=1}^{N}(x-\mu)^T\Sigma^{-1}(x-\mu) = tr(NS\Sigma^{-1}) = Ntr(S\Sigma^{-1})$

Therefore, $\sum_{i=1}^{N}\log N(\mu,\Sigma) = -\frac{1}{2}N\log|\Sigma| - \frac{1}{2}Ntr(S\Sigma^{-1}) + C$

Substitute the equation above to δ, we will have:

$$\Delta = -\frac{1}{2}N_1\log|\Sigma| - \frac{1}{2}N_1tr(S_1\Sigma^{-1}) - \frac{1}{2}N_2\log|\Sigma| - \frac{1}{2}N_2tr(S_2\Sigma^{-1}) + C \quad (7)$$
$$= -\frac{1}{2}N\log|\Sigma| - \frac{1}{2}N_1tr(S_1\Sigma^{-1}) - \frac{1}{2}N_2tr(S_2\Sigma^{-1}) + C \quad (8)$$
$$= -\frac{1}{2}(N\log|\Sigma| + N_1tr(S_1\Sigma^{-1}) + N_2tr(S_2\Sigma^{-1})) + C \quad (9)$$

$$\frac{\partial\Delta}{\partial\Sigma} = 0$$
$$-\frac{1}{2}(N\frac{1}{|\Sigma|}|\Sigma|\Sigma^{-1} + (-1)N_1S_1^T\Sigma^{-2} + (-1)N_2S_2^T\Sigma^{-2}) = 0$$
$$N\Sigma^{-1} - N_1S_1^T\Sigma^{-2} - N_2S_2^T\Sigma^{-2} = 0$$
$$N\Sigma - N_1S_1^T - N_2S_2^T = 0$$
$$\Sigma = \frac{1}{N}(N_1S_1^T + N_2S_2^T) \quad (10)$$

6 Methods for High Dimensional Data Analysis

When we introduced linear regression previously, we talked about regularization methods (the reason for the need for regularization is to prevent overfitting). The most straightforward method to address overfitting is to increase the amount of sample data (sometimes the cost of data collection can be quite high), but we can also consider regularization, because the core of regularization is to constrain the parameter space and impose restrictions on the function (such as ridge, lasso). Additionally, we can also consider dimensionality reduction.

6.1 Curse of Dimensionality

Why do we sometimes need to reduce dimensions? When we are performing tasks like linear regression or linear classification, high dimension can lead to the curse of dimensionality and we can consider the curse of dimensionality as the important reason causes the overfitting problem

Take a 16*16 image as an example, with each pixel having 3 potential choices (RGB), that would be 3^{256} choices. In reality, we can't possibly have this many data, not even 1% of it. As the number of features increases, so does the possibility (combination) of data, and using a fixed percentage of training data also increases geometrically, thus causing the curse of dimensionality.

Here is the geometric interpretation (figure curse of dimensionality):

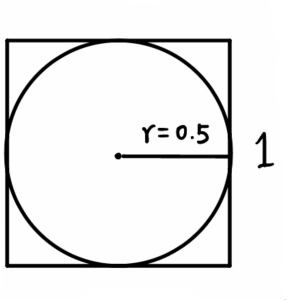


Figure: curse of dimensionality

Based on the figure above, we have a square with side length = 1, and inside the circle has the radius 0.5. When dimension=2 (which I draw on the paper), we have the area of the square = 1, volume = 1; for the circle, its area is $0.5^2\pi$ and its volume = $\frac{4}{3}\pi 0.5^3$. Now let's increase our dimension to D (D is a large positive integer), the volume of the hypercube is still 1, but the volume of the

hyperball is $k0.5^D$. As D goes to infinity, the volume of the hyperball will be 0. Therefore we face a problem: when dimension is small, our data can full fill the sample space like the circle and the square; when we are in the higher dimension, our data will be more and more sparse. (figure cofd)

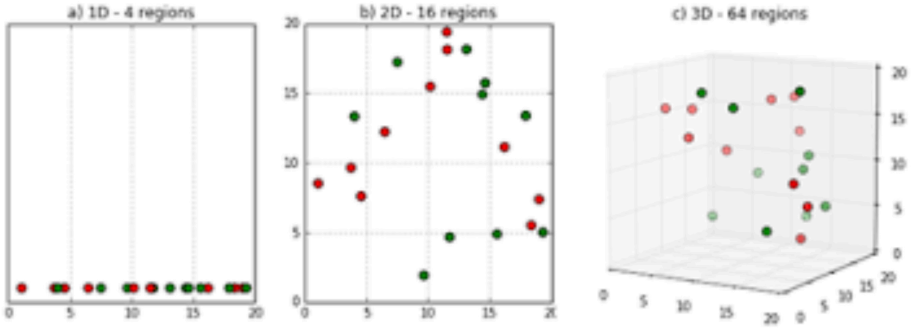


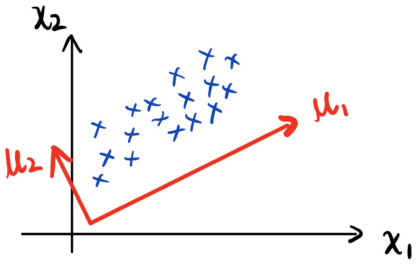
Figure cofd

6.2 PCA (Linear Dimension Reduction)

Before we get start, let me clarify the notation first:

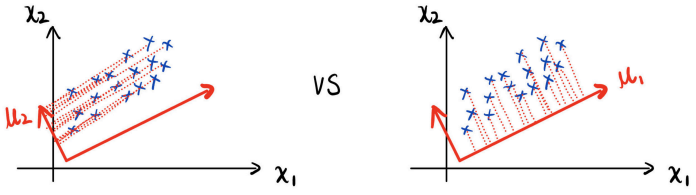
Data: $X = (x_1 x_2 \dots x_N)^T_{(N*P)}$,
Sample Mean: $\bar{X}_{(P*1)} = \frac{1}{N}\sum_{i=1}^{N} x_i$,
Sample Covariance: $S_{(p*p)} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$

We can simply understand PCA as the reconstruction of the features and transform the correlated features to uncorrelated (we can use orthogonal vectors in algebra to express the uncorrelated features). For example, when people want to apply for credit cards from the bank, the bank need to evaluate the applicant based on different aspects (features), such as name, gender, age, job, income, and salary, and we can see that the feature "income" is highly related to the feature "salary". Therefore our mission is to reconstruct features that make sure there are no correlations between each feature (dimension reduced), and we have two perspectives to achieve the goal: maximize projection variance and minimize reconstruction error (they are equivalent).
(figure pca)



As shown in the above figure, assume we have 2-D data in blue, and we have two projection direction u_1 and u_2. The variance of this set of data points is greater after projection in the u_1 direction, and the data is more dispersed, while the projection in the u_2 direction is very dense, so we call the u_1 direction the principal component.

Besides, the reconstruction error refers to the distance between the original data point and the data point after projection. Trivially, we can see that the reconstruction error for projection direction u_2 is much larger that the reconstruction error based on u_1.
(figure u1 vs u2)



6.2.1 Perspective 1: Maximize projection variance

Assume we have the sample x_i then centralize it $(x_i - \bar{x})$, and our projection direction is u_1 ($||u_1|| = 1$). Therefore the projection variance can be expressed as $J = \sum_{i=1}^{N}((x_i - \bar{x}^T)u_1)^2$ since we have the centralized data.
Continue:

$J = \frac{1}{N}\sum_{i=1}^{N}((x_i - \bar{x}^T)u_1)^2$
$J = \frac{1}{N}\sum_{i=1}^{N} u_1^T(x_i - \bar{x})(x_i - \bar{x})^T u_1$ since the inside of the summation is a real number.
$J = u_1^T(\frac{1}{N}\sum_{i+1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T)u + 1$ $J = u_1^T S u_1$

Here we need to find the maximized projection variance which can be considered as an optimization problem. $\hat{u_1} = \arg\max_{u_1} u_1^T S u_1$ s.t. $u_1^T u_1 = 1$, and this

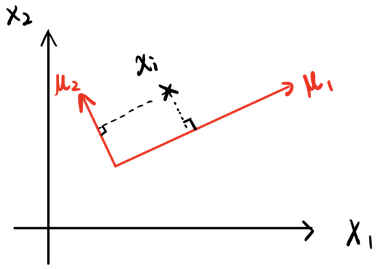
optimization problem with constraint can be solved by Lagrange Multiplier:

$$\begin{aligned} L(u_1, \lambda) &= u_1^T S u_1 + \lambda(1 - u_1^T u_1) \\ \frac{\partial L}{\partial u_1} &= 2Su_1 - \lambda 2u_1 = 0 \\ Su_1 &= \lambda u_1 \end{aligned}$$

Here the S is the covariance matrix and the λ is the eigenvalue, u_1 is the eigenvector.

6.2.2 Perspective 2: Minimize Reconstruction Error

(figure argmin error)



The figure above shows another interpretation of PCA via minimized reconstruction error. We have orthogonal vector u_1 and u_2 and a data point x_i, so we can treat u_1 and u_2 as the new coordinate. $x_i = (x_i^T u_1)u_1 + (x_i^T u_2)u_2$ In the example figure, we only consider the dimension = 2, now be more general:

First we need to centralize the x: $x_i^{'} = x_i - \bar{x}$
Second we reconstruct x to the vector space based on the projection vector:

$x_i^{''} = \sum_{k=1}^{p} (x_i^{'T} u_k) u_k$
Third, after the dimension reduction (assume we reduce to q-dim),
$\hat{x_i}^{''} = \sum_{k=1}^{q} (x_i^{'T} u_k) u_k$.

The minimized reconstruction error means we minimize the cost of reconstruct from $\hat{x_i}^{''}$ to $x_i^{''}$, so we can use the difference between $\hat{x_i}^{''}$ and $x_i^{''}$ to express the reconstruction error:

$$\begin{aligned} J &= \frac{1}{N} \sum_{i=1}^{N} ||x_i^{''} - \hat{x_i}^{''}||^2 \\ &= \frac{1}{N} \sum_{i=1}^{N} || \sum_{k=q+1}^{p} (x_i'^T u_k) u_k||^2 \\ &= \frac{1}{N} \sum_{i=1}^{N} \sum_{k=q+1}^{p} (x_i'^T u_k)^2 \\ &= \sum_{k=q+1}^{p} \frac{1}{N} \sum_{i=1}^{N} (x_i'^T u_k)^2 \\ &= \sum_{k=q+1}^{p} \frac{1}{N} \sum_{i=1}^{N} ((x_i - \bar{x})^T u_k)^2 \\ &= \sum_{k=q+1}^{p} u_k^T S u_k \end{aligned}$$

Now we minimize the cost J so:

$$u_k = argmin \sum_{k=q+1}^{p} u_k^T S u_k, \text{ s.t } u_k^T u_k = 1$$

Using Lagrange multiplier, we can easily solve that $Su_k = \lambda_k u_k$

6.3 Probabilistic PCA

Now we will use the probabilistic perspective to interpret PCA.

Assume our data $x \in R^p$,our feature space has dimension q ($z \in R^q$), let $q < p$. In other words, we use the lower dimensional features z to reconstruct our data x (dimension reduction), or we can treat x as the observed variable and treat z as the latent variable.

Furthermore, we have

$z \sim N(0_q, I_q)$, assume our prior z follows the Gaussian distribution

$x = w_z + \mu + \epsilon$, assume x follows the linear map with z

$\epsilon \sim N(0, \sigma^2 I_p)$
Based on our assumptions, the PPCA is similar to Gaussian mixture model but with Gaussian latent. (figure PPCA)

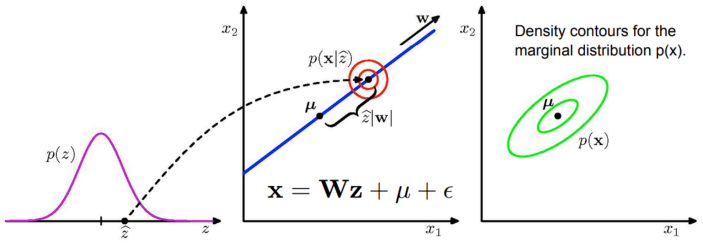


Figure: ppca

Interpretation of the figure above: first, we take latent variable z follows Gaussian distribution. Second, linear map on z, we have $x = w_z + \mu + \epsilon$. At the end, we obtain the distribution of x.

6.3.1 Inference of PPCA: $p(z|x)$

$z \sim N(0_q, I_q)$
$x = w_z + \mu + \epsilon$
$\epsilon \sim N(0, \sigma^2 I_p)$

Therefore $E(x|z) = E(wz+\mu+\epsilon) = wz+\mu$ (because the expectation of E(x—z) is the function z)

$Var(x|z) = Var(wz + \mu + \epsilon) = \sigma^2 I$ since wz and μ are both constants.

So we can conclude $x|z$ $N(wz + \mu, \sigma^2 I)$

Now consider p(x): $E(x) = E(wz + \mu + \epsilon)$
$= \mu$ (because here z is a random variable follows N(0,1)),
and
$Var(x) = Var(wz + \mu + \epsilon)$
$= Var(wz) + Var(\epsilon)$
$= wIw^T + \sigma^2 I$
$= ww^T + \sigma^2 I$
So we can conclude x $N(\mu, ww^T + \sigma^2)$

Now we need to solve the posterior of the z: p(z—x), before we work on the posterior, here is some proof about multivariate Gaussian.
First, this is an important property for multivariate Gaussian.
If we have:

$$x \sim N(\mu, \Sigma) \text{ and } y = Ax + B$$

Therefore:

$$y \sim N(A\mu + B, A\Sigma A^T)$$

This property can be proved easily:
$E[y] = E[Ax + B] = AE[x] + E[B] = A\mu + B$
$Var[y] = Var[Ax + B] = Var[Ax] + Var[B] = AVar[x]A^T = A\Sigma A^T$

Now assume:

$$x = \begin{pmatrix} x_a \\ x_b \end{pmatrix} \mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} \Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}, \text{ and } x \sim N(\mu, \Sigma).$$

For notation simplicity, we denote:

$$x_{b \cdot a} = x_b - \Sigma_{ba}\Sigma_{aa}^{-1} x_a = x_{b \cdot a} = \begin{pmatrix} -\Sigma_{ba}\Sigma_{aa}^{-1} & I \end{pmatrix} \begin{pmatrix} x_a \\ x_b \end{pmatrix}$$

$\mu_{b \cdot a} = \mu_b - \Sigma_{ba}\Sigma_{aa}^{-1}\mu_a$
$\Sigma_{bb \cdot a} = \Sigma_{bb} - \Sigma_{ba}\Sigma_{aa}^{-1}\Sigma_{ab}$

Here $\Sigma_{bb \cdot a}$ is Σ_{bb}'s Schur Complementary. $x_{b \cdot a}$ is the linear combination of x_b and x_a, so it follows Gaussian distribution.

According to the property of Gaussian, we can easily find the mean and variance of $x_{b \cdot a}$:

$$E[x_{b\cdot a}] = \begin{pmatrix} -\Sigma_{ba}\Sigma_{aa}^{-1} & I \end{pmatrix} \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} = \mu_b - \Sigma_{ba}\Sigma_{aa}^{-1}\mu_a = \mu_{b\cdot a}$$

$$\begin{aligned} Var[x_{b\cdot a}] &= \begin{pmatrix} -\Sigma_{ba}\Sigma_{aa}^{-1} & I \end{pmatrix} \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix} \begin{pmatrix} -\Sigma_{ba}\Sigma_{aa}^{-1} \\ I \end{pmatrix} \\ &= \begin{pmatrix} \Sigma_{ba} - \Sigma_{ba}\Sigma_{aa}^{-1}\Sigma_{aa} & \Sigma_{bb} - \Sigma_{ba}\Sigma_{aa}^{-1}\Sigma_{ab} \end{pmatrix} \begin{pmatrix} -\Sigma_{ba}\Sigma_{aa}^{-1} \\ I \end{pmatrix} \\ &= \begin{pmatrix} 0 & \Sigma_{bb} - \Sigma_{ba}\Sigma_{aa}^{-1}\Sigma_{ab} \end{pmatrix} \begin{pmatrix} -\Sigma_{ba}\Sigma_{aa}^{-1} \\ I \end{pmatrix} \\ &= \Sigma_{bb} - \Sigma_{ba}\Sigma_{aa}^{-1}\Sigma_{ab} \\ &= \Sigma_{bb\cdot a} \end{aligned} \tag{11}$$

therefore $x_{b\cdot a} \sim N(\mu_{b\cdot a}, \Sigma_{bb\cdot a})$ and we can also derive $x_b = x_{b\cdot a} + \Sigma_{ba}\Sigma_{aa}^{-1}x_a$

Finally we can find:

$E[x_b|x_a] = \mu_{b\cdot a} + \Sigma_{ba}\Sigma_{aa}^{-1}x_a = \mu_b + \Sigma_{ba}\Sigma_{aa}^{-1}(x_a - \mu_a)$
$Var[x_b|x_a] = \Sigma_{bb\cdot a}\mu_{b\cdot a} + \Sigma_{ba}\Sigma_{aa}^{-1}x_a$
So $x_b|x_a \sim N(\mu_{b\cdot a} + \Sigma_{ba}\Sigma_{aa}^{-1}x_a, \Sigma_{bb\cdot a})$

(Here, the expression of x_b is used directly to calculate $x_b|x_a$. The reason is that the meaning of conditional probability is to find the probability of x_b given that x_a is known. Therefore, it is assumed here that x_a is known and treated as a constant.)

Based on the result above, we make $\begin{pmatrix} x \\ z \end{pmatrix} \sim N\left(\begin{bmatrix} \mu \\ 0 \end{bmatrix}, \begin{bmatrix} ww^T + \sigma^2 I & \Sigma_{xz} \\ \Sigma_{zx} & I \end{bmatrix}\right)$ with $\Sigma_{xz} = \Sigma_{zx}^T$

Now solve Σ_{xz}

$$\begin{aligned} \Sigma_{xz} &= E[(x-\mu)(z-0)^T] \\ &= E[(x-\mu)z^T] \\ &= E[(wz+\varepsilon)z^T] \\ &= E[wzz^T] + E[\varepsilon z^T] \\ &= wE[zz^T] + E[\varepsilon]E[z^T] \\ &= w(Var[z] + E[z]^2) + 0 \\ &= wI \\ &= w \end{aligned} \tag{12}$$

Then $\begin{pmatrix} x \\ z \end{pmatrix} \sim N\left(\begin{bmatrix} \mu \\ 0 \end{bmatrix}, \begin{bmatrix} ww^T + \sigma^2 I & w \\ w^T & I \end{bmatrix}\right)$
$p(z|x) = N(w^T(ww^T + \sigma^2 I)^{-1}(x-\mu), I - w^T(ww^T + \sigma^2 I)^{-1}w)$

7 Application in Financial Markets

7.1 Linear Model in Stock Price

At the beginning, we introduced the linear regression model. Linear regression is a model with high interpretability, simplicity, and relatively low computational cost. However, a significant drawback of linear regression is its unstable accuracy. When the true relationship of the data is linear, linear regression can make reasonably good predictions. However, when the actual data relationship is nonlinear, the linear model will not produce accurate predictions.

We selected the performance of a stock from 2016 to 2021. We set the dependent variable as the closing price, and the independent variables as the opening price, the highest price of the day, the lowest price of the day, and the trading volume. The independent variables are respectively called x1, x2, x3, and x4, while the dependent variable is referred to as y.

```
### 从csv文件中读取股票数据
data <- read.csv("C:/Users/cheng/Desktop/600519.csv", encoding = "UTF-8",sep = ",")
data<- data[2:6]
data
```

Close <chr>	Begin <chr>	Highest <chr>	Lowest <chr>	Amount <chr>
1,932.99	1,950.00	1,959.95	1,919.02	2.63M
1,930.77	1,985.40	1,989.90	1,927.50	3.69M
1,985.00	1,960.00	1,990.00	1,951.08	4.24M
1,941.01	1,964.83	1,967.00	1,925.00	3.27M
1,956.03	1,953.00	1,986.20	1,946.25	3.98M
1,941.00	1,900.00	1,966.00	1,897.00	5.45M
1,896.43	1,852.00	1,917.00	1,852.00	4.58M
1,854.27	1,849.00	1,909.00	1,848.00	4.26M
1,828.41	1,794.19	1,849.00	1,788.20	3.12M
1,792.00	1,781.22	1,807.00	1,770.00	1.95M

1-10 of 1,217 rows Previous 1 2 3 4 5 6 ... 122 Next

(figure data)

Here we upload our data, and we can see there are 1217 observations during 5 years, and next we split data as training set and data set (figure training and test):

```
train.data <- data[1:600,]
train.data
```

	y <chr>	x1 <chr>	x2 <chr>	x3 <chr>	x4 <chr>
1	1,932.99	1,950.00	1,959.95	1,919.02	2.63M
2	1,930.77	1,985.40	1,989.90	1,927.50	3.69M
3	1,985.00	1,960.00	1,990.00	1,951.08	4.24M
4	1,941.01	1,964.83	1,967.00	1,925.00	3.27M
5	1,956.03	1,953.00	1,986.20	1,946.25	3.98M
6	1,941.00	1,900.00	1,966.00	1,897.00	5.45M
7	1,896.43	1,852.00	1,917.00	1,852.00	4.58M
8	1,854.27	1,849.00	1,909.00	1,848.00	4.26M
9	1,828.41	1,794.19	1,849.00	1,788.20	3.12M
10	1,792.00	1,781.22	1,807.00	1,770.00	1.95M

1-10 of 600 rows Previous 1 2 3 4 5 6 ... 60 Next

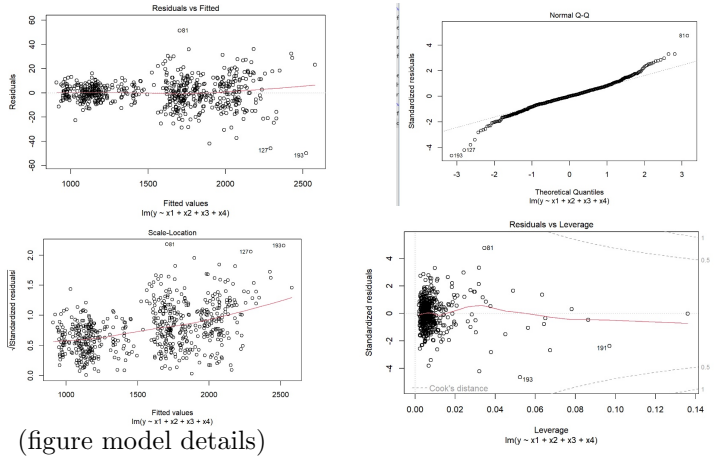
Next step we fit the linear model: try to use the opening price, highest price, lowest price and amount of trading to predict closing price.

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3 + x4, data = train.data)
##
## Coefficients:
## (Intercept)          x1          x2          x3          x4
##   2.347e+00   -6.118e-01    8.239e-01    7.863e-01   -1.892e-07
```

```
summary(lm.train)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3 + x4, data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -50.000  -6.014   0.027   6.057  51.480
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.347e+00  2.559e+00   0.917    0.359
## x1          -6.118e-01  2.491e-02 -24.558   <2e-16 ***
## x2           8.239e-01  3.155e-02  26.116   <2e-16 ***
## x3           7.863e-01  3.337e-02  23.562   <2e-16 ***
## x4          -1.892e-07  3.703e-07  -0.511    0.610
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.04 on 595 degrees of freedom
## Multiple R-squared:  0.9992, Adjusted R-squared:  0.9992
## F-statistic: 1.956e+05 on 4 and 595 DF,  p-value: < 2.2e-16
```

(figure model)
Now let's see the details of the fitted model:



(figure model details)
Based on the diagnostic plots provided:

Non-linearity and Heteroscedasticity: The residuals vs. fitted values plot exhibits patterns, notably a "fan-shape", suggesting non-linearity in the model and heteroscedasticity of residuals.

Normality Concerns: The Q-Q plot indicates that residuals might not be perfectly normally distributed.

High Leverage Points: The Cook's distance plot highlights several observations that might have undue influence on the model's estimates.

In summary, while the model offers an initial fit to the data, it might not be the optimal representation, then we can consider using the log transformation on our data. Here we consider the log transformation on the data which may

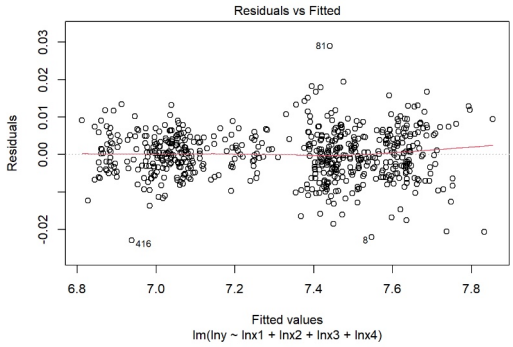
solve the non-constant variance problem.

```
lny<- log(train.data$y)
lnx1<- log(train.data$x1)
lnx2<- log(train.data$x2)
lnx3<- log(train.data$x3)
lnx4<- log(train.data$x4)

lm.sol<-lm(lny ~ lnx1+lnx2+lnx3+lnx4)
lm.sol
```

```
##
## Call:
## lm(formula = lny ~ lnx1 + lnx2 + lnx3 + lnx4)
##
## Coefficients:
## (Intercept)         lnx1         lnx2         lnx3         lnx4
##    0.040115    -0.594215     0.877691     0.714621    -0.001871
```

(figure log)

We can see there is no pattern on the Residual vs Fitted plot.



(figure log plot)

Finally, predict using the regression model and calculate the error rate, where the error rate = (actual value - predicted value) / actual value. In RStudio, take the logarithm of the previously partitioned dataset test.data and plug it into the optimized regression model (figure test diff).

```
lny<- log(test.data$y)
lnx1<- log(test.data$x1)
lnx2<- log(test.data$x2)
lnx3<- log(test.data$x3)
lnx4<- log(test.data$x4)

test.data <- data.frame(y=lny,
                        x1=lnx1,
                        x2=lnx2,
                        x3=lnx3,
                        x4=lnx4)


test.data$pred <- predict(lm.sol,test.data)


test.data$diff <- (abs(test.data$y-test.data$pred)/test.data$y) * 100
test.data
```

Based on the result above, we find the highest difference between prediction and actual value is 0.4318, and the lowest difference between the prediction and actual value is 0.0000398, and the mean of the difference between prediction and actual value is 0.0808885.

```
##        y               x1              x2              x3
##  Min.   :5.781   Min.   :5.781   Min.   :5.793   Min.   :5.778
##  1st Qu.:6.181   1st Qu.:6.181   1st Qu.:6.186   1st Qu.:6.171
##  Median :6.487   Median :6.482   Median :6.505   Median :6.469
##  Mean   :6.396   Mean   :6.394   Mean   :6.407   Mean   :6.382
##  3rd Qu.:6.594   3rd Qu.:6.594   3rd Qu.:6.608   3rd Qu.:6.581
##  Max.   :6.882   Max.   :6.887   Max.   :6.898   Max.   :6.870
##        x4              pred            diff
##  Min.   :14.17   Min.   :5.787   Min.   :0.0000398
##  1st Qu.:14.88   1st Qu.:6.181   1st Qu.:0.0277149
##  Median :15.14   Median :6.489   Median :0.0643825
##  Mean   :15.16   Mean   :6.397   Mean   :0.0808885
##  3rd Qu.:15.42   3rd Qu.:6.596   3rd Qu.:0.1119741
##  Max.   :16.83   Max.   :6.886   Max.   :0.4318202
```

(figure result)

7.2 Regularized Regression

Here we use the same stock data set to perform the ridge and the lasso regression.

Our dependent variable is the y (same as before) which represents the stock price at the closing time, and then we use the function "glmnet" to fit the regularized regression.

```
data_matrix <- as.matrix(data[, -which(names(data) == "y")])
response_vector <- data$y
```

```
ridge_model <- glmnet(data_matrix, response_vector, alpha = 0)
```

```
lasso_model <- glmnet(data_matrix, response_vector, alpha = 1)
```

(figure regularized model fit)

In addition, we use the cross validation approach to find the best λ as I introduced in previous.

```
cv.lasso <- cv.glmnet(data_matrix, response_vector, alpha = 1)
cv.ridge <- cv.glmnet(data_matrix, response_vector, alpha = 0)

best_lambda_lasso <- cv.lasso$lambda.min
best_lambda_ridge <- cv.ridge$lambda.min
```

```
best_lasso_model <- glmnet(data_matrix, response_vector, alpha = 1, lambda = best_lambda_lasso)
best_ridge_model <- glmnet(data_matrix, response_vector, alpha = 0, lambda = best_lambda_ridge)

coefficients_lasso <- coef(best_lasso_model)
coefficients_ridge <- coef(best_ridge_model)
```

```
best_lasso_model <- glmnet(data_matrix, response_vector, alpha = 1, lambda = best_lambda_lasso)
best_ridge_model <- glmnet(data_matrix, response_vector, alpha = 0, lambda = best_lambda_ridge)

coefficients_lasso <- coef(best_lasso_model)
coefficients_ridge <- coef(best_ridge_model)
```

(figure lambda selection)

After obtaining the optimal choice of λ, we use the best λ to fit the model and obtain the coefficients.

```
coefficients_lasso
```

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 2.610713e+01
## x1          1.876031e-01
## x2          7.755867e-01
## x3          2.805628e-04
## x4          .
```

```
coefficients_ridge
```

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 3.553390e+01
## x1          3.175498e-01
## x2          3.201796e-01
## x3          3.302518e-01
## x4          7.124063e-08
```

(figure best lambda model)

Without any surprise, we can see that the coefficient for x4 (amount of trading) in the Ridge regression is close to the 0 (7.124063e-08), and the coefficient for x4 (amount of trading) in Lasso regression is exactly 0.

7.3 Logistic Regression in Credit card examination

Visit the data set

Here I use R to build the logistic regression model for the clients' credit card data.

First I read the csv file and remove null value.

Second I split the data by training set and test set.

Then I fit the logistic model and show the confusion matrix.

One thing need to be aware: although the confusion matrix shows that our model is good, but the warning is not neglectable because maybe our data is not good for performing a reasonable logistic regression which I will discuss later about overfitting problem.

```
# Remove ID column as it's not necessary for the model
credit_data$ID <- NULL

# Set target variable as factor
credit_data$default.payment.next.month <- as.factor(credit_data$default.payment.next.month)

# Split data into training and testing sets
set.seed(123)
train_index <- createDataPartition(credit_data$default.payment.next.month, p = .8, list = FALSE)
train_set <- credit_data[train_index,]
test_set <- credit_data[-train_index,]

# Fit logistic regression model on training data
model <- glm(default.payment.next.month ~ ., data = train_set, family = binomial)

# Generate predicted probabilities for test data
test_set$predicted_probabilities <- predict(model, newdata = test_set, type = "response")

# Create a new column for predicted outcomes based on a threshold of 0.5
test_set$predicted_outcome <- ifelse(test_set$predicted_probabilities > 0.5, 1, 0)

# You can also generate the confusion matrix to evaluate the performance of the model
confusionMatrix(table(test_set$predicted_outcome, test_set$default.payment.next.month))
```

```
Warning: glm.fit: fitted probabilities numerically 0 or 1 occurredConfusion Matrix and Statistics

      0    1
  0 4539  999
  1  133  328

               Accuracy : 0.8113
                 95% CI : (0.8012, 0.8211)
    No Information Rate : 0.7788
    P-Value [Acc > NIR] : 3.605e-10

                  Kappa : 0.2854

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9715
            Specificity : 0.2472
         Pos Pred Value : 0.8196
         Neg Pred Value : 0.7115
             Prevalence : 0.7788
         Detection Rate : 0.7566
   Detection Prevalence : 0.9232
      Balanced Accuracy : 0.6094

       'Positive' Class : 0
```

7.4 Application of Linear Discriminant Analysis on Credit Card

We use the data about credit card application, data frame containing 1,319 observations on 12 variables, and the explanation below is for each variable we'd like to explore.

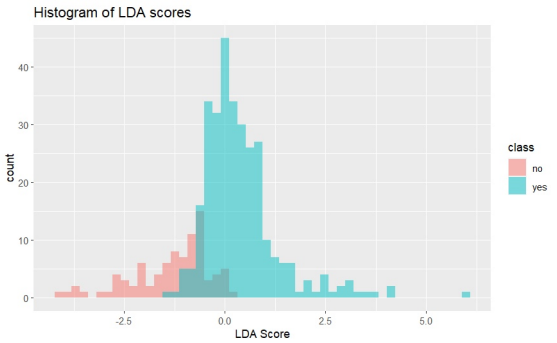
card: Factor.Was the application for a credit card accepted?
reports: Number of major derogatory reports.
age: Age in years plus twelfths of a year.
income: Yearly income (in USD 10,000).
share: Ratio of monthly credit card expenditure to yearly income.
expenditure: Average monthly credit card expenditure.
owner: Factor. Does the individual own their home?
selfemp: Factor. Is the individual self-employed?
dependents: Number of dependents.
months: Months living at current address.
majorcards: Number of major credit cards held.
active: Number of active credit accounts.

First of all, we split our data as training and test.

```
data("CreditCard")


set.seed(123)
splitIndex <- createDataPartition(CreditCard$card, p = 0.7, list = FALSE)
trainData <- CreditCard[splitIndex,]
testData <- CreditCard[-splitIndex,]
trainLabels <- CreditCard$card[splitIndex]
testLabels <- CreditCard$card[-splitIndex]
```

(figure data split)

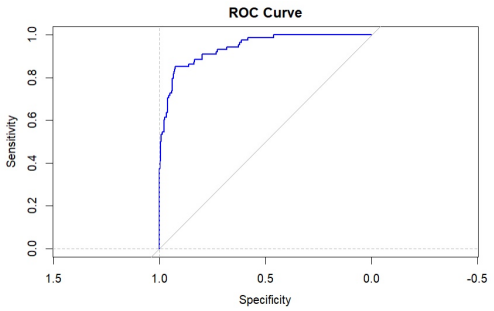
Then we fit the model and I provide the histogram for better visualization.



(figure histogram)

I also generate the confusion matrix that we can evaluate the performance of our lda model. In addition, I plot the ROC curve.
The Receiver Operating Characteristic (ROC) curve is a graphical tool used to evaluate the performance of classification models, especially in binary classification problems. The ROC curve displays the relationship between the True Positive Rate (TPR) and the False Positive Rate (FPR) of the model at various classification thresholds. This helps us understand how many false positives the model might generate when attempting to detect true positives. The ROC curve allows us to compare the performance of multiple models. The Area Under the Curve (AUC) value of the model is a commonly used metric, representing the area under the ROC curve. An AUC value of 1 indicates a perfect model, while 0.5 represents random guessing. Therefore, models with a higher AUC value are generally better than those with a lower AUC value.

```
Confusion Matrix and Statistics

          Reference
Prediction  no yes
       no   38   2
       yes  50 304

               Accuracy : 0.868
                 95% CI : (0.8306, 0.8998)
    No Information Rate : 0.7766
    P-Value [Acc > NIR] : 2.807e-06

                  Kappa : 0.5278

 Mcnemar's Test P-Value : 7.138e-11

            Sensitivity : 0.43182
            Specificity : 0.99346
         Pos Pred Value : 0.95000
         Neg Pred Value : 0.85876
             Prevalence : 0.22335
         Detection Rate : 0.09645
   Detection Prevalence : 0.10152
      Balanced Accuracy : 0.71264
```



(figure lda performance)

My LDA model achieved an accuracy of about 87% on the test data. This implies that the model can correctly classify observations in most cases. The sensitivity and specificity suggest that the model performs relatively well in classifying both positive and negative cases.
The AUC value is close to 1, indicating that the model has a very strong discriminative ability. The higher the AUC value, the better the performance of the model.
The Kappa value is around 0.6, indicating that the model's classification performance surpasses random classification.

7.5 Application of Quadratic Discriminant Analysis on Credit Card

Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) are both variants of discriminant analysis used for classification tasks. The primary distinction between them lies in how they estimate decision boundaries (boundaries for classification). LDA assumes that each class of data has the same covariance, whereas QDA does not make this assumption. This results in LDA having linear decision boundaries, while QDA has quadratic ones. If the relationships between classes in the data are nonlinear, QDA might outperform LDA because QDA can capture these nonlinear decision boundaries. If the classes in the data have significantly different covariance structures, then QDA might be a better choice, as the same covariance assumption of LDA may not hold in such cases.

Similar as the previous section, we use the same data set and fit the quadratic discriminant analysis model for the classification. Our dependent variable is still "card" (whether the application for a credit card accepted), but this time we do not use the full of independent variables: When conducting Quadratic Discriminant Analysis (QDA), one or more of the covariance matrices for the classes are singular or nearly singular, meaning their determinant is close to zero.

So for the simplicity, I remove the variable "expenditure" for completing the QDA.

```
qda_model <- qda(card ~ reports + age + income + share + owner +
selfemp+dependents+months+majorcards+active, data=trainData)


  predictions <- predict(qda_model, newdata=testData)


  accuracy <- sum(predictions$class == testData$card) / nrow(testData)
  cat("Accuracy of QDA model:", accuracy, "\n")


  roc_result <- roc(testData$card, predictions$posterior[, 2], levels=rev(levels(testData$card)))
  auc_value <- auc(roc_result)
  cat("AUC for QDA model:", auc_value, "\n")


  plot(roc_result, main="ROC Curve for Simplified QDA Model")
  abline(h=0, v=1, col="gray")


predictions <- predict(qda_model, testData)
predicted_classes <- predictions$class


conf_matrix <- confusionMatrix(predicted_classes, testData$card)
print(conf_matrix)
```

(figure qda)

Again, we compute the confusion matrix and the ROC curve, we find that the performance of QDA is much better than the LDA for the same data.

```
Confusion Matrix and Statistics

          Reference
Prediction  no yes
       no  111  10
       yes   7 399

               Accuracy : 0.9677
                 95% CI : (0.9489, 0.9811)
    No Information Rate : 0.7761
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.908

 Mcnemar's Test P-Value : 0.6276

            Sensitivity : 0.9407
            Specificity : 0.9756
         Pos Pred Value : 0.9174
         Neg Pred Value : 0.9828
             Prevalence : 0.2239
         Detection Rate : 0.2106
   Detection Prevalence : 0.2296
      Balanced Accuracy : 0.9581

       'Positive' Class : no
```

(figure qda performance)

7.6 PCA in Application

Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of data while retaining as much variability in the data as possible. The main idea behind it is to find the "principal components" of the data, which are orthogonal (perpendicular to each other) and maximize the variance of the data. In many practical applications, a dataset may have tens, hundreds, or thousands of variables. Through PCA, we can reduce the number of these variables while attempting to preserve the main information within the data.
To perform the PCA, we still use the credit card data and our original data has 11 independent variables. Need to be careful that the PCA can only deal with numerical value so I have to check my data to make sure all values are numerical.

```
numericalData <- CreditCard[, sapply(CreditCard, is.numeric)]

pca_result <- princomp(~., data = numericalData,cor =T, center = TRUE, scale. = TRUE)
```

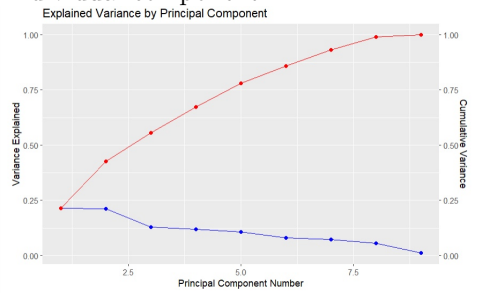
(figure pca)
In R code, we have to functions which can perform the PCA, function "princomp" and "prcomp".
Both "princomp" and "prcomp" are functions in R for performing Principal Component Analysis (PCA). The core distinction between them lies in their computational methods: "princomp" employs eigenvalue decomposition of the data's covariance matrix, whereas "prcomp" uses singular value decomposition of the data matrix, with the latter typically being more numerically stable, especially when the number of observations is less than the number of variables. Furthermore, "prcomp" often outperforms "princomp" in speed when dealing with large datasets. Even though both aim at the same objective, due to its numerical stability and computational efficiency, "prcomp" is generally considered the preferred choice. However, in certain scenarios, "princomp" might also be suitable.

Summary below is the result of the PCA:

```
Loadings:
            Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9
reports      0.238  0.103  0.466  0.431  0.419  0.593                     
age          0.148  0.524 -0.297  0.132 -0.146                0.759       
income              0.514        -0.343         0.100  0.675 -0.277  0.252
share       -0.667                0.232                -0.195         0.668
expenditure -0.660  0.205                0.126                       -0.699
dependents          0.374        -0.563  0.344        -0.641              
months       0.145  0.373 -0.440  0.441 -0.229  0.121 -0.231 -0.575       
majorcards          0.115  0.502 -0.130 -0.776  0.273 -0.180              
active              0.341  0.493  0.300        -0.730                     
```

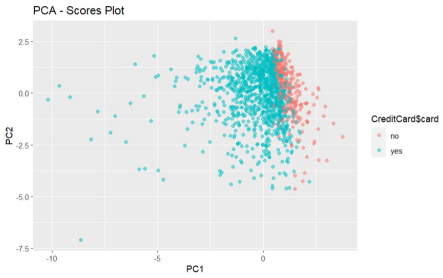
(figure pca result)

Visualize our result, we can find most of components are important: most of them contribute more 10% explained variance to our model. Here the red line represents the cumulative variance and the blue line represents variance for each individual component.



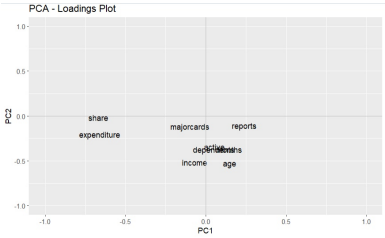
(figure cumulative variance)

In addition, we can make the PCA scores plot. The PCA scores plot is a two-dimensional visualization representation of principal component analysis, where each point represents an observation in the dataset, and its position on the plot is determined by its scores on the first two principal components. This plot provides us with a low-dimensional "snapshot" of the data, revealing the relative relationships between observations, potential clusters, and the primary directions of variation in the data.



(figure pca scores plot)

Besides, we can also examine the loadings plot.



(figure pca loadings)