

Contents

1	Linear Model in Stock Price	2
1.1	Data Resource and Interpretation	2
1.2	Fitting Linear Model with Model Motivation	3
1.3	Model Diagnostic	4
1.4	Log Transformation	5
1.5	Model Performance	6
2	Regularized Regression in Application	7
2.1	Data Resource and Model Motivation	7
2.2	Model Fitting	7
2.3	Result of Regularized Regression	8
3	Credit Card Repayment Task	9
3.1	Logistic Regression on Credit Card Examination	9
3.1.1	Model Motivation	9
3.1.2	Data Resource and Interpretation	10
3.1.3	Logistic Model Fitting	12
3.2	Tree Method	13
3.2.1	Why Decision Tree is Reasonable	13
3.2.2	Fit the Decision Tree	14
3.2.3	Decision Tree Result and Interpretation	15
4	Personal Credit Examination	16
4.1	Application of Linear Discriminant Analysis on Credit Card . . .	16
4.1.1	Model Motivation and Data Resource	16
4.1.2	LDA Model Fitting	18
4.2	Application of Quadratic Discriminant Analysis on Credit Card .	21
4.2.1	Why QDA?	21
4.2.2	QDA Model Fitting	22
4.3	PCA in Application	23

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.

1.1 Data Resource and Interpretation

We downloaded stock price data for a stock (A Share, Guizhou Moutai) from December 1, 2016, to 2021 from the internet.

Based on the data we obtained, we have the following variables: Date (representing the date when the price is recorded), Begin (price at the opening), Highest (representing the highest stock price of the day), Lowest (lowest price of the day), and Amount (total transaction volume for the day). Additionally, our raw data contains a variable called "Percent", which merely describes the change in transaction volume from the previous day to the current day. We don't believe that the "Percent" variable will affect our model, so we will not consider the "Percent" variable when building our model.

日期	收盘	开盘	高	低	交易量	涨跌幅
2021-12-1	1,932.99	1,950.00	1,959.95	1,919.02	2.63M	+0.11%
2021-11-30	1,930.77	1,985.40	1,989.90	1,927.50	3.69M	-2.73%
2021-11-29	1,985.00	1,960.00	1,990.00	1,951.08	4.24M	+2.27%
2021-11-26	1,941.01	1,964.83	1,967.00	1,925.00	3.27M	-0.77%

	Date	Close	Begin	Highest	Lowest	Amount	Percent
1	2021/12/1	1,932.99	1,950.00	1,959.95	1,919.02	2.63M	0.11%
2	2021/11/30	1,930.77	1,985.40	1,989.90	1,927.50	3.69M	-2.73%
3	2021/11/29	1,985.00	1,960.00	1,990.00	1,951.08	4.24M	2.27%
4	2021/11/26	1,941.01	1,964.83	1,967.00	1,925.00	3.27M	-0.77%
5	2021/11/25	1,956.03	1,953.00	1,986.20	1,946.25	3.98M	0.77%
6	2021/11/24	1,941.00	1,900.00	1,966.00	1,897.00	5.45M	2.35%
7	2021/11/23	1,896.43	1,852.00	1,917.00	1,852.00	4.58M	2.27%
8	2021/11/22	1,854.27	1,849.00	1,909.00	1,848.00	4.26M	1.41%
9	2021/11/19	1,828.41	1,794.19	1,849.00	1,788.20	3.12M	2.03%

(figure raw data and data in Rstudio)

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.

In machine learning, splitting data into training and test sets is essential for evaluating model performance. The training set aids in building and tuning the model, while the test set provides an unbiased evaluation of its effectiveness on unseen data. This separation ensures that we don't overfit our model, which occurs when the model performs exceptionally well on training data but poorly on new, unseen data. Overfitting typically results from a model being too complex and capturing noise or random fluctuations in the training data. By using a test set, we can detect overfitting and achieve a more generalizable model.

```
# Set a random seed for reproducibility
set.seed(123)

# Randomly split the data into training and test sets
train_indices <- sample(1:nrow(data), nrow(data) * 0.5)
train_set <- data[train_indices, ]
test_set <- data[-train_indices, ]
```

(data-split)

1.2 Fitting Linear Model with Model Motivation

In this section, our task is to predict stock prices based on observed historical prices. We treat the closing price as the dependent variable, which we hope to predict using other variables, denoted as y . The other variables are defined as follows: opening price is denoted as x_1 , the highest price of the day as x_2 , the lowest price of the day as x_3 , and the trading volume as x_4 .

We've chosen a linear regression model to predict stock prices, which I believe is a conservative and rational approach. First, linear relationships are the most common and easily interpretable way to describe the relationship between independent and dependent variables. Given the variables we've previously defined, it's challenging to deduce their relationship with the dependent variable based solely on the definition of each independent variable. Thus, we conservatively opt for a linear regression model. In the subsequent section, we will conduct model diagnostics to verify the appropriateness of the linear regression model.

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

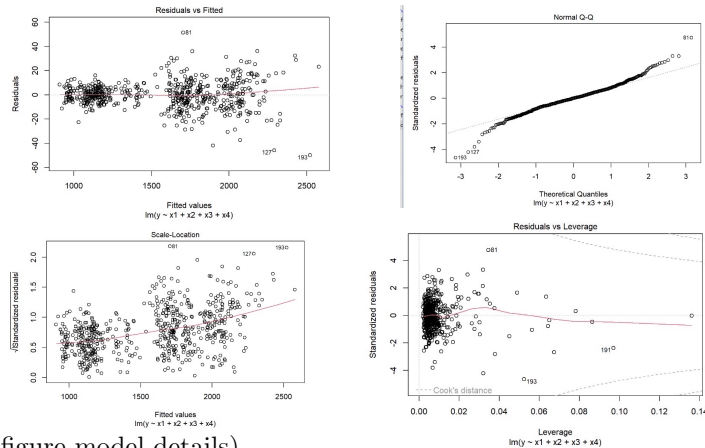
```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3 + x4, data = train.data)
##
## Coefficients:
## (Intercept)      x1          x2          x3          x4
##  2.347e+00  -6.118e-01  8.239e-01  7.863e-01  -1.892e-07
```

```
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
```

1.3 Model Diagnostic

As I mentioned in previous section, we only "assume" there is a linear relation between the closing price of the stock and other variables. Now we need to check whether our fitted model satisfies the linear assumptions and 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.

1.4 Log Transformation

Log-transformation can help in stabilizing the variances and make the data more normal distribution-like, which is an assumption in linear regression for the residuals. Log-transformation can reduce the relative impact of extreme values or outliers, as these can disproportionately influence the results of a linear regression model.

Now we will fit the model after the log transformation and here is the result.

```
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)

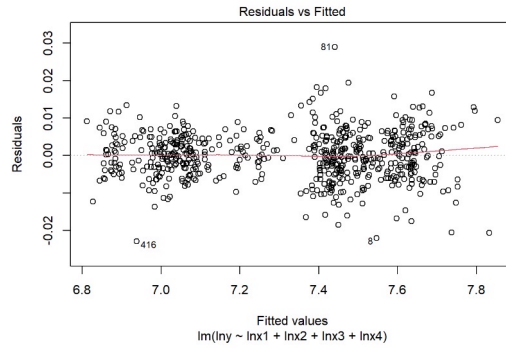
lm.sol<-lm(lny ~ lnx1+lnx2+lnx3+lnx4)
lm.sol

##
## 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)

Again, we check the Residuals vs Fitted plot, can we find that there is no pattern on the plot which is much better than previous. (figure see next page).

(figure log plot)



1.5 Model Performance

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)

2 Regularized Regression in Application

2.1 Data Resource and Model Motivation

In the previous section, we employed a linear regression model to predict the relationship between the closing price and variables such as the opening price, highest price, lowest price, and trading volume. Now, we are faced with a question: Is there truly a relationship between the closing price and the trading volume? This is a very tricky question: some believe that high trading volume implies a drop in price, while others think price changes lead to variations in trading volume. Therefore, in this chapter, we will delve into Regularized Regression to explore the relationships between our predictors and the response variable. This is because Regularized Regression allows us to filter our predictors and determine which ones might be redundant.

Also, we use the same dataset about stock price as the linear regression model which I introduced in previous section.

2.2 Model Fitting

Before we use R code to fit our regularized regression model, let's review the expression of Ridge regression and Lasso regression.

Ridge regression: $\min_{\beta} \left\{ \sum_{i=1}^n (y_i - x_i^T \beta)^2 + \lambda \sum_{j=1}^p \beta_j^2 \right\}$

Lasso regression: $\min_{\beta} \left\{ \sum_{i=1}^n (y_i - x_i^T \beta)^2 + \lambda \sum_{j=1}^p |\beta_j| \right\}$

The important task is to find the proper λ , here are many ways to find the optimal λ , and we choose to use cross-validation approach to find the λ .

```
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 "dgCMatix"
##              s0
## (Intercept) 2.610713e+01
## x1          1.876031e-01
## x2          7.785867e-01
## x3          2.805628e-04
## x4          .
```

```
coefficients_ridge
```

```
## 5 x 1 sparse Matrix of class "dgCMatix"
##              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)

2.3 Result of Regularized Regression

Based on the result above, we find that x_4 (represents the trading amount) is 0 in the Lasso regression, and it is 7.124063e-08 under the Ridge regression.

Without any surprise, we know that the Lasso regression will shrink some of estimators towards 0 which implies that we can ignore the relation between the closing price (dependent variable we want to study) and the trading amount.

Do we have to drop off the estimator x_4 ? Well, I think it is not a bad idea if we delete one variable because in fact the stock price data could be very complicated if we consider other potential variables, so making our model less complex will help reduce the computational cost.

3 Credit Card Repayment Task

In this section, we try to resolve the credit card examination problem. In general, credit card is a type of consumption in advance, therefore the credit card issuer (always the bank) will be likely to see their clients repay the credit card on time. So in this section we will use different methods to explore the credit card examination.

3.1 Logistic Regression on Credit Card Examination

3.1.1 Model Motivation

Logistic Regression is a widely used statistical method for addressing binary classification or multiclass classification problems. The advantages of logistic regression include its simplicity, the ability to interpret linear relationships of input features, and the capacity to produce probability scores.

The core of credit card examination is to predict whether a customer might default in the future. This is a typical binary classification problem: either the customer defaults or does not. Logistic regression is well-suited for such problems because it can predict the probability of an event occurring. Logistic regression not only informs us about the potential default of a customer but also provides the probability of the default. This offers more flexibility for decision-makers as they can devise strategies or decisions based on these probabilities. Logistic regression provides a model that is easy to interpret. Banks and financial institutions often need to explain their models and decisions, especially in contexts involving consumer credit.

In summary, logistic regression offers both an effective and interpretable solution for the credit card scoring problem, enabling financial institutions to better assess and manage credit risk.

3.1.2 Data Resource and Interpretation

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005, and below is the preview of the data frame. (figure data frame)

ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default.payment.next.month
1	20000	2	2	1	24	2	2	-1	-1	-2	-2	3913	3102	689	0	0	0	689	0	0	0	0	0	1
2	120000	2	2	2	36	-1	2	0	0	0	2	2582	1725	2682	3272	3455	3261	0	1000	1000	1000	0	2000	1
3	90000	2	2	2	34	0	0	0	0	0	0	29239	14027	13559	14331	14948	15549	1518	1500	1000	1000	1000	5000	0
4	50000	2	2	1	37	0	0	0	0	0	0	46990	48233	49291	28314	28959	29547	2000	2019	1200	1100	1069	1000	0
5	50000	1	2	1	37	-1	0	-1	0	0	0	8617	5670	39835	20940	19146	19131	2000	36681	10000	9000	689	679	0
6	50000	1	1	2	37	0	0	0	0	0	0	64400	57069	57608	19394	19619	20024	2500	1815	657	1000	1000	800	0
7	5.00E+05	1	1	2	29	0	0	0	0	0	0	367965	412023	445007	542653	483003	473944	55000	40000	38000	20239	11750	13770	0
8	5.00E+05	2	2	2	23	0	-1	-1	0	0	-1	11876	380	601	221	159	567	380	601	0	181	1687	1542	0
9	140000	2	3	1	28	0	0	2	0	0	0	11285	14096	12108	12211	11793	3719	3329	0	432	1000	1000	1000	0
10	20000	1	3	2	35	-2	-2	-2	-2	-1	-1	0	0	0	0	13007	13912	0	0	0	13007	1122	0	0
11	2.00E+05	2	3	2	34	0	0	2	0	0	-1	11073	6787	5535	2513	1828	3731	2306	12	50	300	3738	66	0
12	260000	2	1	2	51	-1	-1	-1	-1	-1	-2	12261	21670	9966	8517	22287	13668	21818	9966	8583	22301	0	3640	0
13	630000	2	2	2	41	-1	0	-1	-1	-1	-1	12137	6500	6500	6500	6500	2870	1000	6500	6500	2870	0	0	0
14	70000	1	2	2	30	1	2	2	0	0	2	65802	67869	65701	66782	36137	56894	3200	0	3000	3000	1500	0	1
15	250000	1	1	2	29	0	0	0	0	0	0	70887	67060	63561	59696	56875	55512	3000	3000	3000	3000	3000	3000	0
16	50000	2	3	3	23	1	2	0	0	0	0	50614	29173	28116	28771	29531	30211	0	1500	1100	1200	1300	1100	0
17	20000	1	1	2	24	0	0	2	2	2	2	15376	18050	17628	18338	17905	19104	3200	0	1500	0	1600	0	1
18	320000	1	1	1	49	0	0	0	-1	-1	-1	253286	246536	194663	70074	5856	195599	10358	10000	75940	20000	195599	50000	0
19	360000	2	1	1	49	1	-2	-2	-2	-2	-2	0	0	0	0	0	0	0	0	0	0	0	0	0
20	180000	2	1	2	29	1	-2	-2	-2	-2	-2	0	0	0	0	0	0	0	0	0	0	0	0	0
21	130000	2	3	2	39	0	0	0	0	0	-1	38358	27688	24489	20616	11802	930	3000	1537	1000	2000	930	33764	0
22	120000	2	2	1	39	-1	-1	-1	-1	-1	-1	316	316	316	0	632	316	316	316	0	632	316	0	1

This dataset is very complicated and before we fitting our model, we need to know the meaning of each column.

There are 25 variables:

ID: ID of each client

LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit)

SEX: Gender (1=male, 2=female)

EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)

MARRIAGE: Marital status (1=married, 2=single, 3=others)

AGE: Age in years

PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months, 9=payment delay for nine months and above)

PAY_2: Repayment status in August, 2005 (scale same as above)

PAY_3: Repayment status in July, 2005 (scale same as above)

PAY_4: Repayment status in June, 2005 (scale same as above)

PAY_5: Repayment status in May, 2005 (scale same as above)

PAY_6: Repayment status in April, 2005 (scale same as above)

BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)

BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)

BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)

BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)

BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)

BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)

PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)

PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)

PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)

PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)

PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)

PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)

default.payment.next.month: Default payment (1=yes, 0=no)

3.1.3 Logistic Model Fitting

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

As usual, the first task is to split the training set and test set.

```
# 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,]
```

(split training and test sets)

Then we fit the model and get the confusion matrix.

```
# 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
```

(figure fit, result)

Overall, the logistic model seems to perform well as we can see the accuracy is 0.8113 and the sensitivity is 0.9715, but there is one thing not to be ignored. We find that there is a warning in the R output:

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

This warning may occur because of the well-defined class data as we introduce in previous section (5.2 Why not Logistic Regression?). In conclusion, the choice of implementing logistic regression may not be a good idea, so let's explore another way to solve the question.

3.2 Tree Method

3.2.1 Why Decision Tree is Reasonable

The branching of a decision tree is determined based on specific features and their corresponding thresholds. Common branching decision criteria include: Information Gain - selecting the feature and threshold that maximize information gain; Gini Impurity - selecting the feature and threshold that minimize Gini impurity. However, these theoretical parts are not important, as we can download packages such as "rpart," "caret," "rpart.plot," "Metrics" to implement the decision tree code in Rstudio.

We regard the decision tree as a method that can be used to solve classification problems, and the advantage of the decision tree is that its interpretability is very strong: In the later Model Result section, you will see a very easy-to-understand tree model, where each branch represents the value of different variables. For the binary classification problem of judging whether credit card-holders will be overdue, we will use the decision tree model, and the decision tree model will tell us which variables can help us make predictions.

Again, we use the same data set introduced in previous (credit card clients data set).

3.2.2 Fit the Decision Tree

When fitting a decision tree, two parameters are particularly important: ‘cp’ and ‘maxdepth’.

‘cp’ is a complexity parameter, used to control the growth of the decision tree. A larger ‘cp’ value results in a smaller, simpler decision tree, as a higher ‘cp’ value restricts the growth of the tree. When a node’s split does not lead to a significant improvement in the objective function (such as classification error), that split will be pruned. A smaller ‘cp’ value makes the decision tree larger and more complex. By lowering the ‘cp’ value, we allow the tree to make more splits, even if those splits don’t greatly improve the model. Overall, larger ‘cp’ values may lead to underfitting, while smaller ‘cp’ values may lead to overfitting.

Similarly, ‘maxdepth’ controls the maximum depth of the decision tree. The depth is the number of nodes in the longest path from the root node to any leaf node. A larger ‘maxdepth’ value allows the tree to grow deeper, thereby producing a more complex model. A smaller ‘maxdepth’ value restricts the size of the tree, making it simpler. Too great a depth may lead to overfitting, while too small a depth may lead to underfitting.

Here I set the $cp = 0.001$, with $maxdepth = 4$, I am not very worried about the overfitting problem because we have a large data set.


```
fit <- rpart(default.payment.next.month ~ ., data=train_data, method="class",
control=rpart.control(cp=0.001, maxdepth=4))
|
predictions <- predict(fit, test_data, type="class")

acc <- accuracy(test_data$default.payment.next.month, predictions)
print(paste("Accuracy:", acc))

rpart.plot(fit)

...

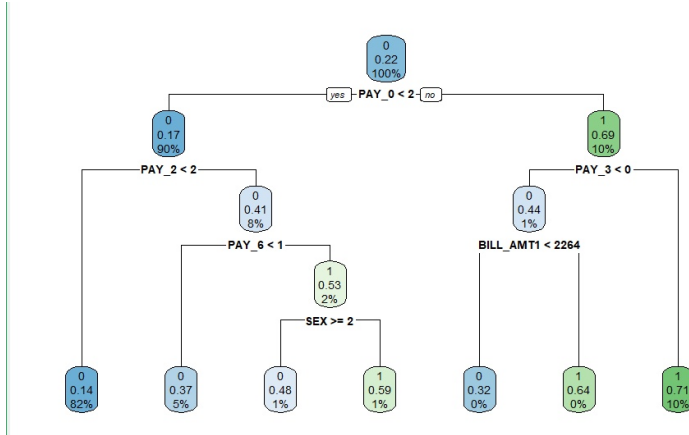
[1] "Accuracy: 0.820833333333333"
```



(figure tree) According to the model fitted above, we find the accuracy is 0.82 which is a good prediction. Besides, the prediction accuracy of the logistic model is also closed to 0.82, maybe we can conclude that the two methods perform similarly.

3.2.3 Decision Tree Result and Interpretation

As I mentioned in previous, the one significant advantage of decision tree is easy to interpret. Let's see the result below.



(figure decision tree)

I will explain the model from top to the bottom. First of all, the decision tree choose the variable "PAY_0" to build the first split, and "PAY_0" represents that the status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, . . . 8=payment delay for eight months, 9=payment delay for nine months and above. In other words, for any customer, the earlier they pay their credit card debt, the lower the value of the "PAY_0" variable will be. Based on the data we get, we find the highest "PAY_0" value is 8, which represents this client repays the credit card very late, and the lowest value of the "PAY_0" is -2, which represents no delay.

According to the first split, we can see that if PAY_0 is less than 2 (delay no more than 2 months), then we will classify them as the group of "less probability for default" and if PAY_0 is large than 2, we will classify them as "more probability for default". This is a reasonable result because sometimes people may forget to repay their credit cards only because of some miscellaneous instead of delay on purpose.

We have total depth = 4 which makes the decision tree model increase the interpretability. Overall, this decision tree provides a guideline for us to predict that if a client will make default. For example, if a person who do not repay late than 2 months in September, and also do not have 2 months delay for the payment in July, so we will classify the client as less chance for default. If a client repay credit card delay for more than 2 months in September, also delay for more than 2 months in July, then we will classify the client as the person who tend to make payment default.

4 Personal Credit Examination

4.1 Application of Linear Discriminant Analysis on Credit Card

Linear Discriminant Analysis (LDA) is a technique for classification and dimensionality reduction. LDA can be used as a classification algorithm, especially when the features are continuous and follow a normal distribution. By maximizing the mean differences between different categories and minimizing the variance within categories, LDA seeks the linear combinations that best differentiate different categories. Compared to some non-linear methods, LDA is relatively simple and computationally efficient. The output of LDA is easy to understand, as it finds the linear combinations that clearly distinguish the categories.

4.1.1 Model Motivation and Data Resource

This time, we have obtained a dataset named "CreditCard," which records the personal information of 1329 credit card applicants. We have divided their personal information into 11 columns, and the dataset includes the results of the credit card application (yes or no).

	card	reports	age	income	share	expenditure	owner	selfemp	dependents	months	majorcards	active
1	yes	0	37.66667	4.5200	0.0332699100	124.983300	yes	no	3	54	1	12
2	yes	0	33.25000	2.4200	0.0052169420	9.854167	no	no	3	34	1	13
3	yes	0	33.66667	4.5000	0.0041555560	15.000000	yes	no	4	58	1	5
4	yes	0	30.50000	2.5400	0.0652137800	137.869200	no	no	0	25	1	7
5	yes	0	32.16667	9.7867	0.0670505900	546.503300	yes	no	2	64	1	5
6	yes	0	23.25000	2.5000	0.0444384000	91.996670	no	no	0	54	1	1
7	yes	0	27.91667	3.9600	0.0125757600	40.833330	no	no	2	7	1	5
8	yes	0	29.16667	2.3700	0.0764337600	150.790000	yes	no	0	77	1	3
9	yes	0	37.00000	3.8000	0.2456279000	777.821700	yes	no	0	97	1	6
10	yes	0	28.41667	3.2000	0.0197800000	52.580000	no	no	0	65	1	18
11	yes	0	30.50000	3.9500	0.0780245600	256.664200	yes	no	1	24	1	20
12	no	0	42.00000	1.9800	0.0006060606	0.000000	yes	no	2	36	1	0
13	no	0	30.00000	1.7300	0.0006936416	0.000000	yes	no	1	42	0	12
14	yes	0	28.83333	2.4500	0.0387955100	78.874170	yes	no	0	26	1	3
15	yes	0	35.33333	1.9080	0.0269067100	42.615000	yes	no	2	120	0	5

(figure creditcard)

Now let me explain the meaning of each column:

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.

Clearly, our task is to evaluate and predict the credit card applications of the applicants using the known data. Here we have adopted LDA for two reasons

Firstly, we can recall that in our previous application of logistic regression, we used it to analyze whether credit cards were overdue. In the database we used for logistic regression, many columns were related; for example, one column described whether the customer delayed paying the credit card in September, while another column described whether the customer delayed paying the credit card in July. In this chapter, the database "CreditCard" we are using has each column with a unique significance. We can see that each column describes features of the credit card applicants: age, income, and family, etc. Therefore, we believe that using Linear Discriminant Analysis will best differentiate the linear combination of different categories and complete the classification task.

Secondly, we know that Linear Discriminant Analysis can achieve good results when the variables are continuous and follow a normal distribution. So, let's verify whether each feature in our data follows a normal distribution, and here we use the Shapiro-Wilk test to test whether the feature follows the normal distribution (see next page).

```

for (variable in names(data)) {
  if (is.numeric(data[[variable]])) {
    print(paste("Testing variable:", variable))
    test_result <- shapiro.test(data[[variable]])
    print(test_result)
  } else {
    print(paste("Skipping non-numeric variable:", variable))
  }
}

```

The following is the result of our Shapiro test and we find that all of the numeric features almost follow the normal distribution. In addition, we should disregard the variables "reports" and "majorcards", because although these two features are numeric, they are not continuous so we do not consider their normality.

Description: df [9 × 3]

	Feature <small><chr></small>	W_Value <small><dbl></small>	P_Value <small><dbl></small>
W	reports	0.3841797	6.742010e-55
W1	age	0.9409574	1.670381e-22
W2	income	0.8297162	2.698195e-35
W3	share	0.6962697	1.448756e-43
W4	expenditure	0.6534963	1.416778e-45
W5	dependents	0.7812855	9.229590e-39
W6	months	0.7263608	5.236243e-42
W7	majorcards	0.4697317	2.186863e-52
W8	active	0.8989383	1.290299e-28

(figure shapiro result)

4.1.2 LDA Model Fitting

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

```

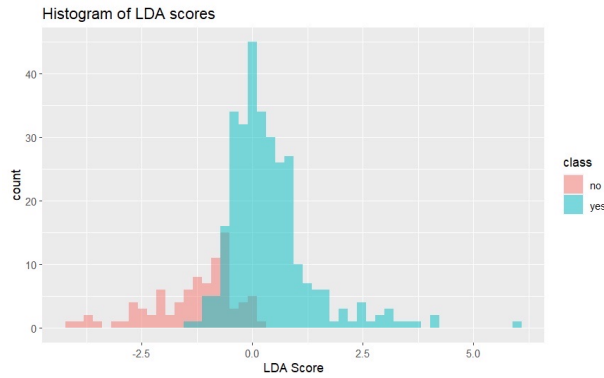
data("CreditCard")

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)

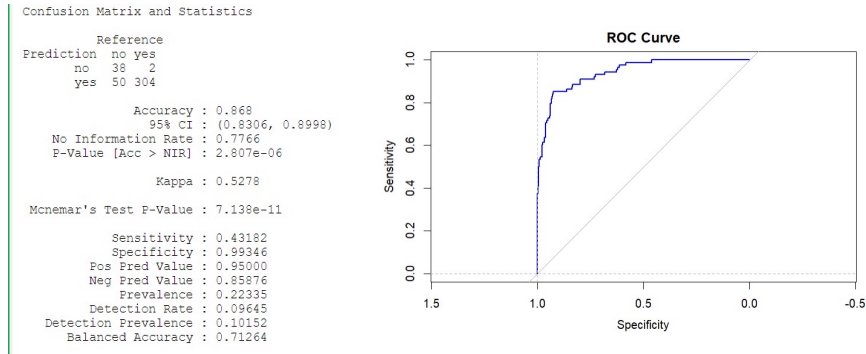
Then we fit the model and I provide the histogram for better visualization.



(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.



(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 dis-

criminative 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.

4.2 Application of Quadratic Discriminant Analysis on Credit Card

Now we will change our point of view, and this section we will use Quadratic Discriminant Analysis on the same data set in previous section.

4.2.1 Why QDA?

In the previous LDA section, we found that when we used LDA to classify credit card applicants, the accuracy reached 87% (a commendable achievement). However, what we need to note is that similar to Linear Discriminant Analysis (LDA), QDA is also used to find the boundaries between different categories. Unlike LDA, QDA does not assume that all categories have the same covariance matrix. Since QDA does not assume that the covariance is the same across all categories, it can capture more complex boundaries. Compared to LDA, QDA can more flexibly estimate the parameters of different categories.

Of course, it should be noted that QDA has certain requirements for the amount of data: QDA needs enough samples to accurately estimate the covariance matrix for each category. If there are too few samples, the estimates may be inaccurate.

In simple terms, the content of this chapter is more like an extension of the previous LDA chapter: we want to test whether QDA has better effects for a 1329*12 dataset.

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.

4.2.2 QDA Model Fitting

Similar as previous, we use the same data set but this time we change LDA to QDA, here is the code how I fit the QDA model.

```
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)

And the following is the performance of the QDA, we find that the performance of QDA is much better than the LDA for the same data.

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
```

4.3 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 two functions which can perform the PCA, function "princomp" and "prcomp".

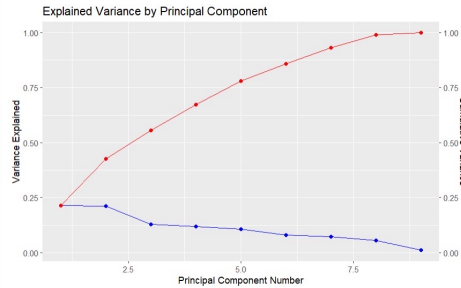
Both "princomp" and "prcomp" are functions in R for performing Principal Component Analysis (PCA). The core distinction between them lies in their computational methods: "princomp" employs eigenvalue decomposition of the data's covariance matrix, whereas "prcomp" uses singular value decomposition of 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			

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.



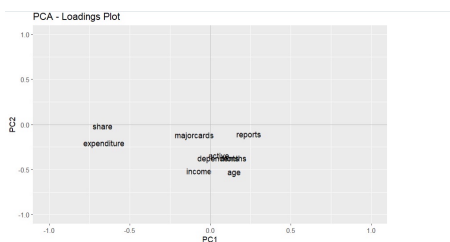
(figure cumulative variance)

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)