Customers’ Interest on Vehicle Insurances

Group 3: Abdullah Bin Umar Khan, Ammar Alzureiqi, Zhaoqi Yang

Introduction

An Insurance company that has provided Health Insurance to its customers wishes to build a model to predict whether the policyholders (customers) from the past year will also be interested in Vehicle Insurance provided by the company.

Just like medical insurance, there is vehicle insurance where every year a customer needs to pay a premium of a certain amount to the insurance company so that in case of an unfortunate accident by the person driving the vehicle, the insurance company will provide compensation (called ‘sum assured’) to the customer.

Building a model to predict whether a customer would be interested in Vehicle Insurance is extremely helpful for the company because it can then accordingly plan its communication strategy to reach out to those customers and optimize its business model and revenue.

Dataset description

The dataset is available on Kaggle, provided by one whose client is an Insurance company. There are three datasets available: *train.csv*, *test.csv*, and *sub.csv*. The *test.csv* file is for prediction and does not include a response variable while the *sub.csv* file is for submission of the predicted response. Since we are only interested in the relationship between the response and the predictors, and not the predictions themselves, we have only made use of the *train.csv* file, which contains the necessary data.

The dataset called *train.csv* includes the following 12 variables:

Variable	**Definition**
id	Unique ID for the customer
Gender	Gender of the customer
Age	Age of the customer
Driving_License	0 : Customer does not have DL, 1 : Customer already has DL
Region_Code	Unique code for the region of the customer
Previously_Insured	1 : Customer already has Vehicle Insurance, 0 : Customer doesn't have Vehicle Insurance
Vehicle_Age	Age of the Vehicle, including 3 groups: < 1 Year, > 2 Years, 1-2 Year

Vehicle_Damage	1 : Customer got his/her vehicle damaged in the past. 0 : Customer didn't get his/her vehicle damaged in the past.
Annual_Premium	The amount customer needs to pay as premium in the year
Policy_Sales_Channel	Anonymized Code for the channel of outreaching to the customer ie. Different Agents, Over Mail, Over Phone, In Person, etc.
Vintage	Number of Days, Customer has been associated with the company
Response	1 : Customer is interested, 0 : Customer is not interested

Table 1: Meanings of Variables in the dataset

The data includes 266,776 observations with no missing values. 88% of the customers are not interested in the vehicle while others are interested. 46% of the customers are female while others are male. The customers are aged from 20 to 85, and a median of 36.54% of the customers have previously insured while the others have not. Almost every customer has a driving license.

Methods

Classification methods require supervised machine learning algorithms to assign a class/label to a set of data points. Some common algorithms include logistic regression, LDA, QDA, and classification trees. In this project, the goal is to classify a policyholder into one of two classes while the models will use various factors to predict if an individual may be interested in vehicle insurance.

Confusion Matrix and Classification Statistics:

In binary classification problems, confusion matrices are used as important indicators of model performance. These matrices compare the actual classifications with the predicted classifications and display the counts of correctly and incorrectly classified observations. The values on the diagonal represent the counts of correctly classified values, while the remaining values represent the counts of incorrectly classified observations.

Additionally, some common statistics that are calculated from these values are used to assess model performance, which include:

- **Accuracy:** The proportion of observations that were classified correctly
- **Specificity:** The proportion of negatives that are correctly classified
- **Sensitivity (Recall):** The proportion of positives that are correctly classified
- **Precision:** The proportion of predicted positives that were correctly classified
- **F1 Score:** The harmonic mean between recall and precision

Logistic Regression:

Logistic regression models, similarly to linear regression models, use a linear combination of the predictors to predict the response. However, instead of estimating the response directly, logistic regression uses the logit link function that links the linear combination of predictors to the estimated probability of having a 'positive' response.

Decision Trees:

A non-parametric supervised machine learning algorithm, which can be used in both regression and classification problems. The general structure of a decision tree is as follows; the root node is the top of the decision tree, as we descend down the tree, each node where a decision must be made (and the predictor space is split) is called an internal node. At the ends of the tree we have the terminal nodes that correspond to a subset of the predictor space, and these provide the predicted value, or label the ones that are to be assigned to predictors that are in those respective subsets. Decision trees result in highly interpretable models that are very simple to explain.

Boosting and Bagging:

Boosting of decision trees is used to improve predictive capabilities by growing multiple trees sequentially. Instead of training one large decision tree (strong classifier) and risking overfitting, the algorithm begins with much smaller trees (weak classifiers) that are fitted and incorporated in training subsequent trees so that each tree essentially learns from the information provided by the tree(s) that were fitted before it. In essence, the boosted model will learn 'slowly' as more trees are fit. Gradient boosting is a type of boosting technique where each tree is specifically evaluated based on its loss function (which can be chosen by the user), by using gradient descent optimization.

LDA and QDA:

Linear discriminant analysis is similar to logistic regression, but instead of modelling $P(Y=k|X=x)$ using the logit link, LDA attempts to model the distribution of the predictors for each class, using Bayes theorem. QDA is similar to LDA except the covariance matrix depends on the class k.

We fitted models using logistic regression, linear discriminant analysis, and quadratic discriminant analysis. We found the best thresholds with ROC analysis for the test data. We applied Synthetic Minority Over-sampling Technique (SMOTE) and simple undersampling to obtain balanced datasets. We transferred the factors to dummy variables and built some k-Nearest-Neighbour classifiers from the data. We also applied simple decision trees, random forests, and boostings to the data. The final decision we made was based on the sensitivity, specificity, and accuracy of each model.

A Brief Introduction to SMOTE

When a dataset is unbalanced, like in our case, that nearly 90% of the responses are 0 (customers not interested in the new vehicle insurance). The predictions of models are likely to be affected by the majority class too much. For instance, our first tree model turned out to produce only 0 predictions - that means the model did not classify the data at all. Resampling is a useful method to fix this problem. We can oversample class 1 or undersample class 0 to obtain a balanced dataset and use this new dataset to train our model.

SMOTE is a method that combines over-sampling the minority (abnormal) class and under-sampling the majority (normal) class, introduced by Nitesh V. Chawla, Kevin W. Bowyer, and their team in 2002. SMOTE can achieve better classifier performance (in ROC space) than only under-sampling the majority class.

Results

Exploratory data analysis

Upon first inspection of the data, it is clear that the data set is severely unbalanced, with 334399 observations classified as 'negative' and only 46710 classified as 'positive'. This poses a problem to statistical modelling of the data, because many classification models are built on the assumption that there are an equal number of observations in each class.

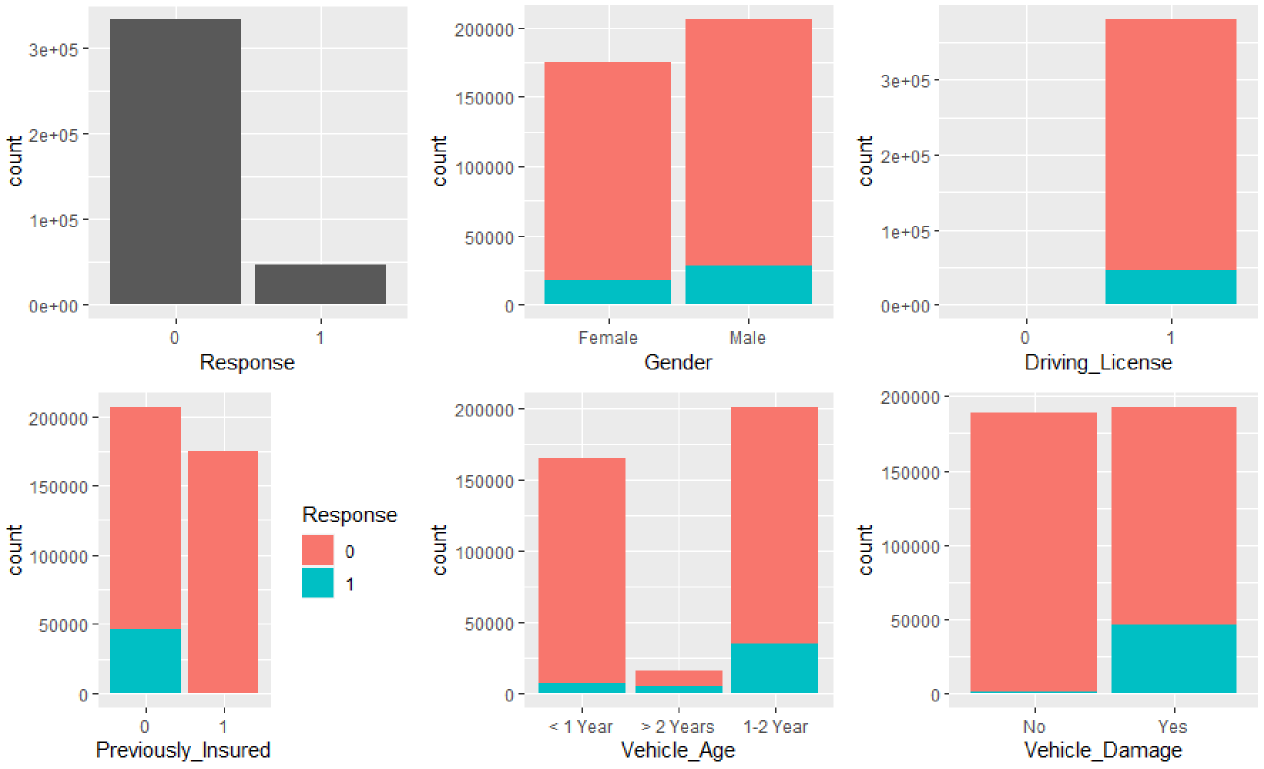

Figure 1: Plots of Category Predictors (with only several levels) versus the Response

The above plots are used to visualize the cross tabulations between categorical features and the response.

Some important things to note are that the distribution of the response is similar for both male and females. Although it is difficult to read the plot for Driving_License, by looking at their relative frequencies, it appears that customers with a driver's license are twice as likely to be interested in adding vehicle insurance.

Additionally, if the policyholder already has vehicle insurance, they will always be uninterested in adding vehicle insurance to their policy with this company. And policyholders with cars older than 2 years old appear to be more interested in vehicle insurance, while policyholders with cars less than 1 year old appear to be the least interested.

Lastly, individuals who have had their vehicle damaged before are far more likely to be interested in vehicle insurance

Therefore we can expect Driving_License, Previously_Insured, Vehicle_age and Vehicle_Damage to be useful as predictors in our models, while Gender might not be.

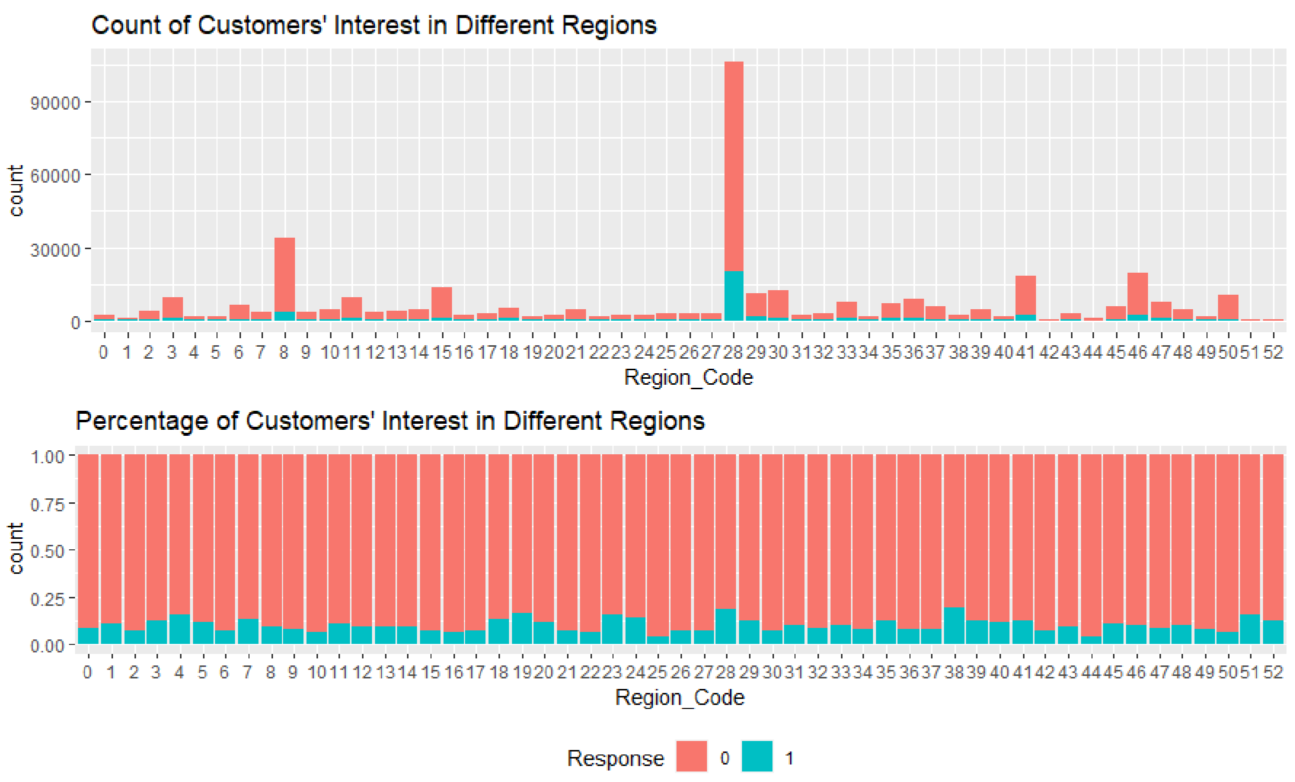

Figure 2: Plot of Region Numbers versus the Response

Although the geographical locations that correspond to the region codes are not publicly disclosed, the insurance agency has access to this information and can use results involving this variable accordingly. The lower graph in Figure 2 demonstrates that the division of the populations in all regions are not the same, with respect to the response. Indicating that the policyholders in some regions are more likely to be interested in vehicle insurance when compared to others.

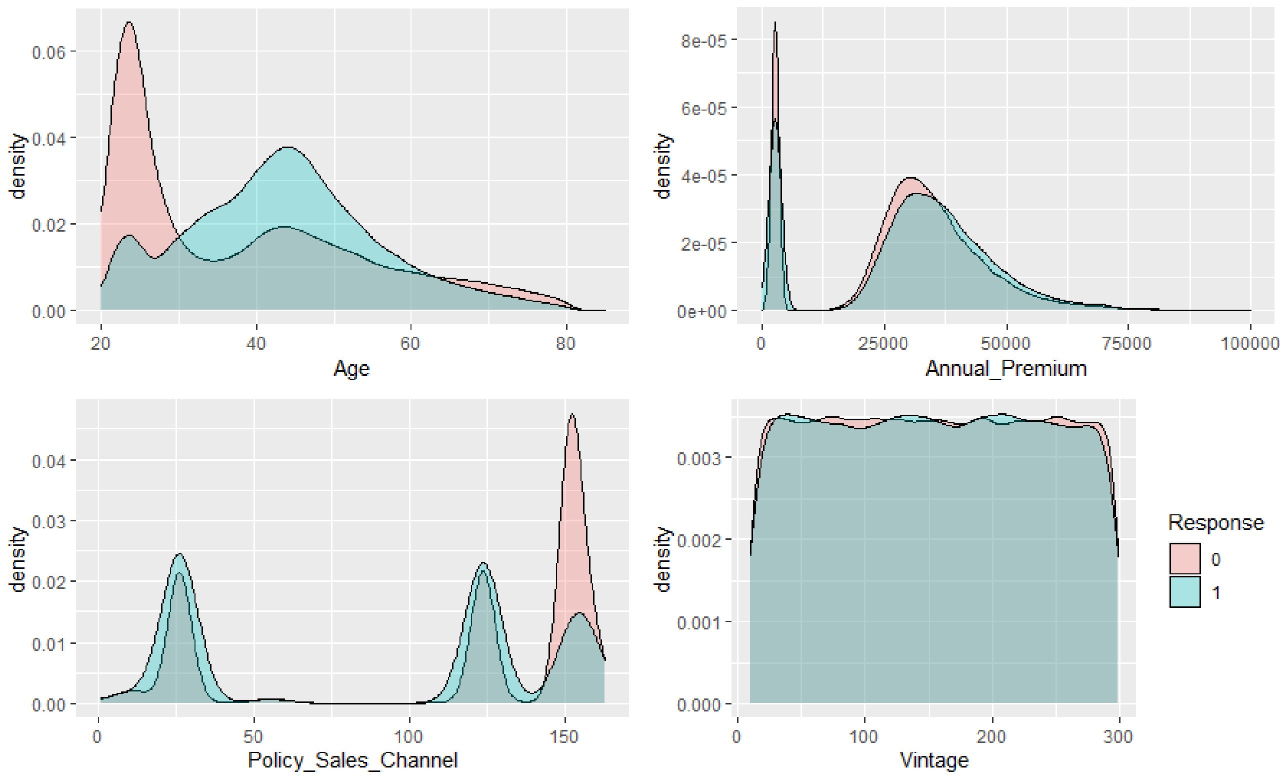

Figure 3: Plots of Continuous Predictors versus the Response

Lastly, we investigate the distributions of the continuous features for each class of the response. From these plots we can conclude that the distributions of the Annual_Premium and Vintage features are similar for each class, and therefore may not be helpful as predictors. On the other hand, the distributions for Age and Policy_Sales_Channel are significantly different and may be helpful in our models.

Main data analysis

We separated the data into a training set and a test set. The training set included 70% of the observations.

We first fitted a logistic regression (LR) model to the data and the model after stepwise variable selection includes 8 predictors: Gender, Age, Driving_License, Region_Code, Previously_Insured, Vehicle_Age, Vehicle_Damage, and Policy_Sales_Channel. The removed variables are consistent with the intuition we got from the plots. Annual_Premium and

Vintage are not important predictors. However, the p-value of the Hosmer-Lemeshow test is less than 0.001, indicating the model fits badly. Figure 4 is the ROC plot for the LR model using the test data. By maximizing the AUC, we specified the sensitivity (0.92), the specificity (0.66), and the accuracy (0.69). The sensitivity is quite large but the specificity is a little small. And the accuracy is probably not large enough.

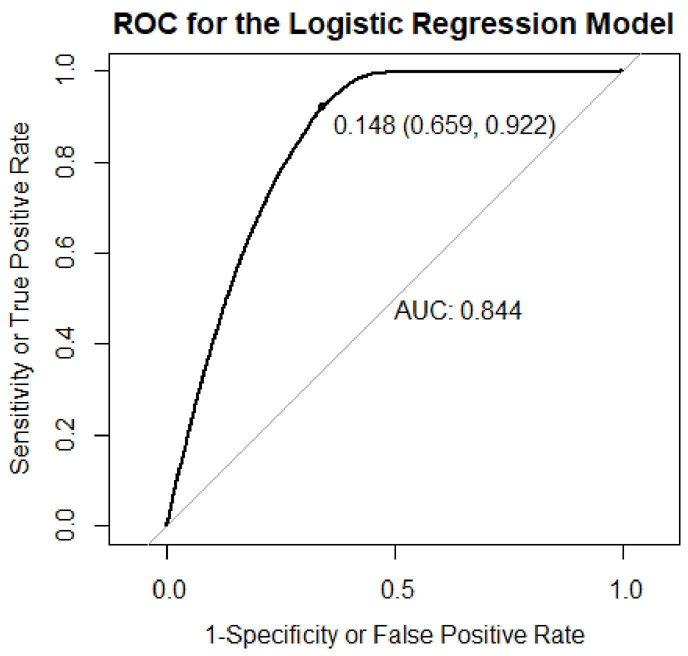

Figure 4: The threshold = 0.148 maximizes the AUC

Then we applied the linear discriminant analysis (LDA) and the quadratic discriminant analysis (QDA) to the data using these 8 predictors. We did the ROC analysis again with the test data. For the LDA model, we obtained the sensitivity (0.91), the specificity (0.66), and the accuracy (0.69). For the QDA model, we obtained the sensitivity (0.88), the specificity (0.65), and the accuracy (0.68). Both the predicted results are not better than the LR model.

To find a better fit, we considered that it was possible that the unbalance of the response variable had a big impact on the model fitting. We then applied SMOTE to our data. The new dataset is made up of 50% of response 0 and 50% of response 1. We fitted a logistic regression model again with this new dataset but unfortunately, the fit was not improved at all. The p-value from the HL test is still less than 0.001 and the test confusion matrix didn't look better, with sensitivity (0.91), specificity (0.61), and accuracy (0.64).

As another resampling method, simple undersampling was also tried. We just separated class 0 (the majority) and class 1 (the minority), sampled a smaller dataset from class 0 and combined the new class 0 and the original class 1. The new dataset is made up of 50% of response 0 and 50% of response 1. The LR model for this dataset yielded p-value for HL test (< 0.001), sensitivity (0.93), specificity (0.66), and accuracy (0.69).

It seems that the resampling method did not improve our models. But when applying kNN and tree methods, the resampling dataset did perform better than the original dataset.

Then we applied kNN to the full train dataset and the second resampling dataset. We removed the variable *Annual_Premium* when fitting kNN because it included too large values (like 540,165) that are far from its median (30,564) and its third quantile (39,400). That makes a variable not useful in kNN because the method calculates the Euclidean distances (or other metrics). We standardized the continuous variables and changed the category variables into dummy variables with one exception, *Region_Code*. The predictor *Region_Code* is a factor with 53 levels. It really influences the calculation time a lot if we change *Region_Code* into 52 dummy variables. We tried to fit kNN to the undersampling dataset with dummy variable *Region_Code* for k = 3, but it cost us 80 min to finish running this only one case. And the sensitivity (0.22) for this model is too low. So we decided not to include *Region_Code* in the kNN classification.

We tested k = 1, ..., 10 but the results are similar for different k's. We didn't do the cross-validation because this process is extremely time-consuming. Figure 5 shows the f1 scores (calculated using train data and undersampling data) for different k's. The f1 score is the harmonic average of the precision and the sensitivity. And the best number of nearest neighbours we found, with both high sensitivity and specificity, is k = 1 for the undersampling dataset, and k = 1 for the full train dataset. The undersampling dataset has a sensitivity (0.72), a specificity (0.72), an accuracy (0.72), and running time (around half of an hour). The full train dataset has a sensitivity (0.27), a specificity (0.91), an accuracy (0.83), and running time (around 2 hours). The kNN model with the full train dataset also suffered a lot from the unbalance.

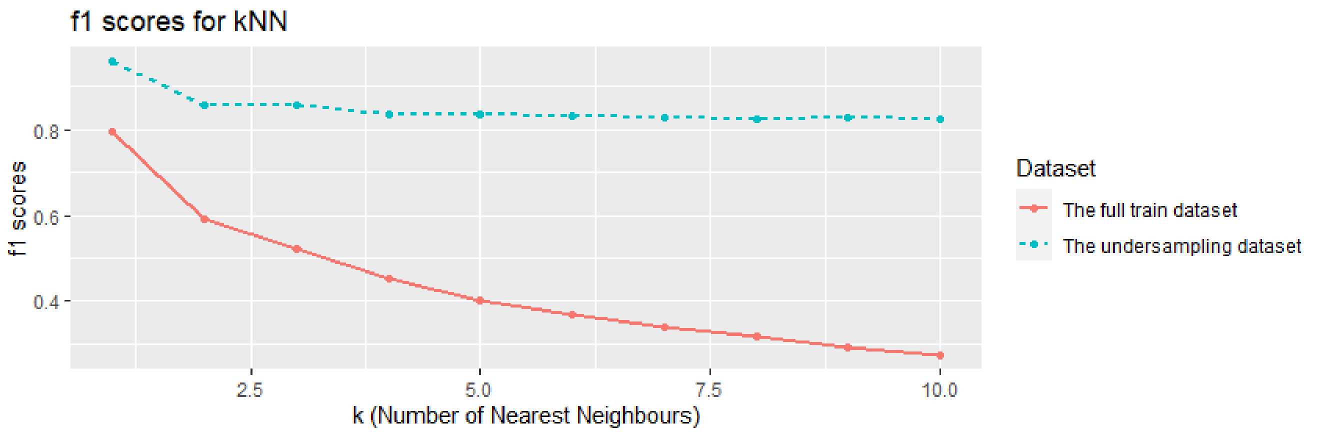

Figure 5: The f1 scores for k = 1, ..., 10 and for the full train data and the undersampling data

Then we started the decision tree part. Because factor predictors must have at most 32 levels in this function when using the *tree* function, we removed the variable *Region_Code* in this part. We fitted a tree with the original unbalanced train data and the decision tree (Figure 6, left) we got was extremely useless. This is the main reason why we turned to resample methods. And on the right of Figure 6 is the decision tree we fitted with the undersampling data, with sensitivity (0.89), specificity (0.69), and accuracy (0.71).

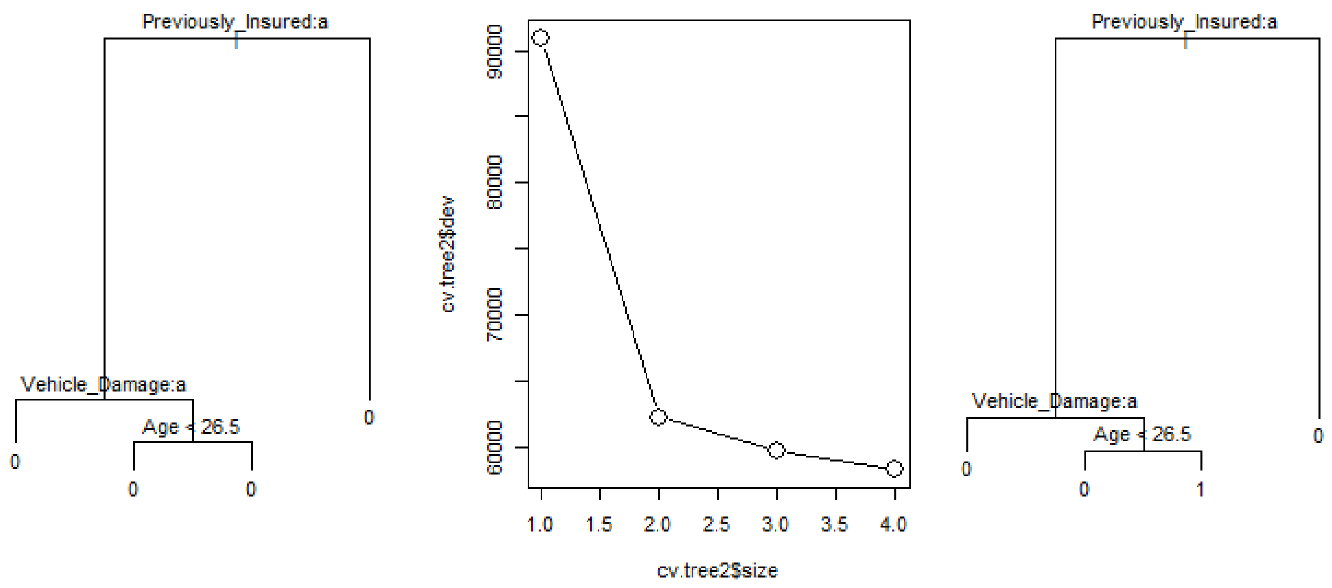

Figure 6: (Left)The tree from the full data; (Mid)The CV selection for the right tree; (Right)The tree from the undersampling data.

We also applied random forest and boosting to the undersampling data. The random forest model has a sensitivity (0.94), a specificity (0.66), and an accuracy (0.69). And the boosting model is with the number of trees 5000 and interaction depth 4. The boosting model has a sensitivity (0.93), a specificity (0.66), and an accuracy (0.69).

Lastly, we tried making a couple of neural networks with 2 hidden layers and changing the number of nodes per layer. Unfortunately, the models always predicted the response to be 0. This result is similar to the tree that was fit on the full data and is also not very useful. Additionally, unlike tree-based models, neural networks aren't very interpretable and are more of a 'black-box' type method which doesn't provide us with any insight towards our goal.

Table 2 is a summary of all the classification methods above. And the best model we think is the kNN model obtained from the undersampling data.

Model	**Sensitivity**	**Specificity**	**Accuracy**
LR with full data	0.92	0.66	0.69
LDA with full data	0.91	0.66	0.69

QDA with full data	0.88	0.65	0.68
LR with SMOTE	0.91	0.61	0.64
LR with undersampling	0.93	0.66	0.69
kNN with full data	0.27	0.91	0.83
kNN with undersampling	0.72	0.72	0.72
kNN with undersampling + Region_Code	0.22	0.94	0.85
tree with full data	0	1	0.88
tree with undersampling	0.89	0.69	0.71
random forest with undersampling	0.94	0.66	0.69
boosting with undersampling	0.93	0.66	0.69
neural networks	0	1	0.88

Table 2: Sensitivity, Specificity and Accuracy of Main Models

Conclusion/Discussion

Conclusion

This dataset is strongly unbalanced and one can easily obtain a wrong accuracy of 0.88, which is the proportion of the response 0. We used some resampling methods to obtain a balanced dataset to train our models. And the best model we found was kNN with undersampling data, whose sensitivity, specificity, and accuracy are all high, but this method is relatively slow in the running speed.

The decision tree model with undersampling data has also got similarly good results as the kNN model, and the tree model is simple, intuitively. Only the customers who have not previously insured, and have got vehicle damages, and whose ages are larger than 26.5 are predicted to have an interest in this new insurance. These conditions are the conditions of the third terminal of the tree. Figure 7 shows that, in the balanced dataset, the tree model fits very well. Most of the responses in terminal 3 are 1, and most of the responses not in terminal 3 are 0. But the separation is not good enough when it comes to the whole dataset.

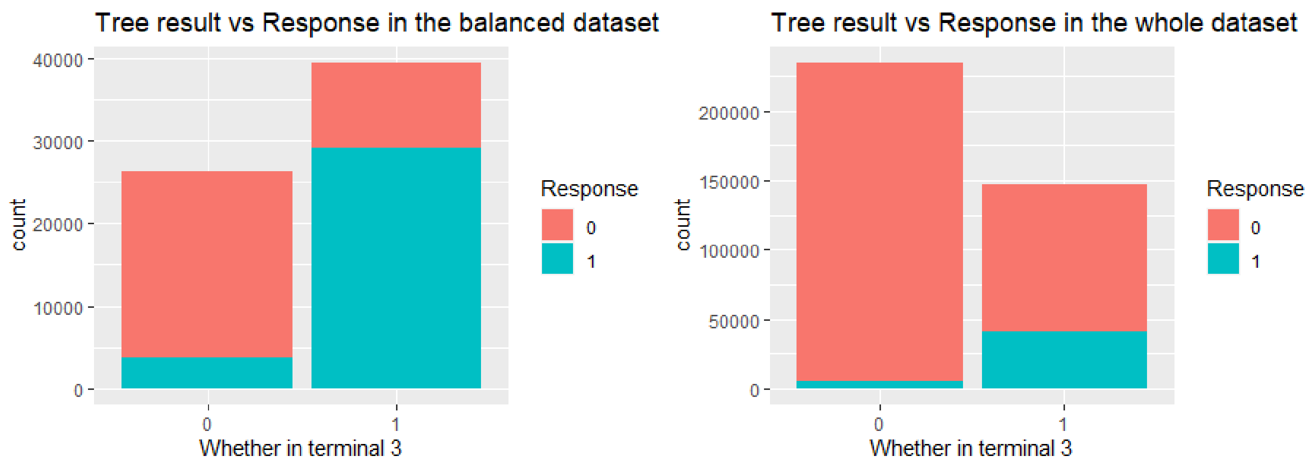

Figure 7: Comparison of the tree predicted response 1 (terminal 3) versus true response 1

Possible Further Work

In the kNN part, we didn’t use cross validation to select k for the long running time. We would be more likely to finish the CV process if we have got an idle computer. Because running the CV means we cannot use the Rstudio for at least a whole day, if we can obtain the correct final results by only once running and we use 10 fold CV. We could have also used CSL (Cost Sensitive Learning) to resample the data based on the cost matrix, which is basically a confusion matrix that focuses more on the false positives and negatives.

We have seen, on the kaggle site, there’s someone obtaining an accuracy of 0.91 with python codes, using a very new method called XGBoost (eXtreme Gradient Boosting). This method is from machine learning and was created by Tianqi Chen in 2014. We have had a quick look at this method and tried it in R. The method ended after 2000 iterations, but our result (sens =0.92, spe = 0.65, accu = 0.68, running time around 2 hours) is not as good as the 0.91 result. It is possibly because we don’t really understand this method so there have been some attributives wrong. If we want to dig on this data further, we may need to learn about the XGBoost method.

Contributions

- Abdullah Bin Umar Khan:
  - Other: Proof-reading and basic editing for written part and some input on Possible Further Work
  - Code: minimal undersampling*
- Ammar Alzureiqi:
  - Code: EDA, LR, Neural Network
  - Written Part: Introduction, Dataset Description, Methods, EDA

- Zhaoqi Yang:
    - Code: LDA, QDA, SMOTE, undersampling, kNN, Tree, Random Forest, Boosting, XGBoosting.
    - Written Part: Introduction, Dataset Description, the Data Analysis part corresponding to my code, the Conclusion and Discussion part corresponding to my code
    - Others: Selecting the dataset

References

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning : with Applications in R. New York :Springer, 2013.

Hastie, Trevor, Trevor Hastie, Robert Tibshirani, and J H. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer, 2001. Print.

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. J. Artif. Intell. Res. 16: 321-357 (2002)

Rachael Tatman. Machine Learning with XGBoost (in R). 2018
kaggle.com/rtatman/machine-learning-with-xgboost-in-r

Appendix for R Code

Summary table

Name	Section
Abdullah Bin Umar Khan	5*
Ammar Alzureiqi	1, 2, 3, 8
Zhaoqi Yang	4, 5, 6, 7

Section 0: Packages

```
## Packages
library(tidyverse)
library(ResourceSelection) #hl test
library(gridExtra) #grid plots
library(pROC) #ROC
library(MASS) #LDA
library(class) #kNN
library(caret) #kNN
library(nnet) #change factor into dummy variables
library(neuralnet) #NN
#install.packages("remotes")
#library(remotes) # to install DMwR
#install_version("DMwR", "0.4.1")
library(DMwR) #SMOTE
library(tree) #tree
library(randomForest) #random forest
library(gbm) #boosting
library(xgboost) # for xgboost
```

Section 1: Data cleaning

```
health = read.csv(file = 'train.csv')

# No missing values
anyNA(health)

# Formatting
health$Gender = factor(health$Gender)
health$Driving_License = factor(health$Driving_License)
health$Region_Code = factor(health$Region_Code)
health$Previously_Insured = factor(health$Previously_Insured)
health$Vehicle_Age = factor(health$Vehicle_Age)
health$Response = as.factor(health$Response)

n = length(health$Response)
summary(health)
str(health)
prop.table(table(train$Response))
prop.table(table(train$Gender))
prop.table(table(train$Previously_Insured))
prop.table(table(train$Driving_License))
```

Section 2: Exploratory data analysis

```
#fig1~fig6: categorical predictors
```

```
fig1 = ggplot(health, aes(Response)) +
  geom_bar() +
  guides(fill=FALSE)
fig2 = ggplot(health, aes(x = Gender, fill = Response)) +
  geom_bar() +
  guides(fill=FALSE)
fig3 = ggplot(health, aes(x = Driving_License, fill = Response)) +
  geom_bar() +
  guides(fill=FALSE)
fig4 = ggplot(health, aes(x = Previously_Insured, fill = Response)) +
  geom_bar()
fig5 = ggplot(health, aes(x = Vehicle_Age, fill = Response)) +
  geom_bar() +
  guides(fill=FALSE)
fig6 = ggplot(health, aes(x = Vehicle_Damage, fill = Response)) +
  geom_bar() +
  guides(fill=FALSE)

DL = sum(as.numeric(health$Driving_License)-1); noDL = n - DL
round(table(Response = health$Response, DL =
health$Driving_License)/c(noDL,noDL,DL,DL), 2)

age = as.numeric(table(health$Vehicle_Age))
table( health$Response, health$Vehicle_Age)/c(age[1], age[1], age[2],
age[2], age[3], age[3])

# fig8: Region: categorical predictor with 52 categories
# These two need to be plot alone.
fig8.count = ggplot(health, aes(x = Region_Code, fill = Response)) +
  geom_bar() +
  labs(title='Count of Customers\' Interest in Different Regions') +
  theme(legend.position = "none")
fig8.percentage = ggplot(health, aes(x = Region_Code, fill =
Response)) +
  geom_bar(position="fill") +
  labs(title='Percentage of Customers\' Interest in Different
Regions') +
  theme(legend.position = "bottom")

# fig7, 9, 10, 11: continuous predictors
```

```
fig7 = ggplot(health, aes(x = Age, fill = Response)) +
  geom_density(alpha=.3) +
  theme(legend.position = "none")
fig9 = ggplot(health, aes(x = Annual_Premium, fill = Response)) +
  geom_density(alpha=.3) +
  xlim(0, 1e05) +
  theme(legend.position = "none")
fig10 = ggplot(health, aes(x = Policy_Sales_Channel, fill =
Response)) +
  geom_density(alpha=.3) +
  theme(legend.position = "none")
fig11 = ggplot(health, aes(x = Vintage, fill = Response)) +
  geom_density(alpha=.3)

grid.arrange(fig1, fig2, fig3, fig4, fig5, fig6, nrow=2, ncol=3)
grid.arrange(fig7, fig9, fig10, fig11, nrow=2, ncol=2)
grid.arrange(fig8.count, fig8.percentage, nrow=2, ncol=1)
```

Section 3: Logistic Regression

```
## Seperate the data into a training set and a test set
set.seed(48509850)
index = sample(n, n*0.7)
train = health[index,]
test = health[-index,]

## Simple logistic regression
logi.mod1 = glm(Response ~ ., data = train, family = "binomial")
logi.mod2 = step(logi.mod1, trace = 0)
summary(logi.mod2)
# Gender + Age + Driving_License + Region_Code + Previously_Insured +
Vehicle_Age + Vehicle_Damage + Policy_Sales_Channel

## HL test
hoslem.test(logi.mod2$y, fitted(logi.mod2),g=10)
# p-value = 1.776e-15

## Predictions for the test data
logi.probs2 = predict(logi.mod2, newdata = test, type = "response")
```

```
## ROC analysis
roc_obj_logi2 = roc(response = test$Response, predictor =
logi.probs2)
roc_logi2 = c(coords(roc_obj_logi2, "b",
                         ret=c("threshold","se","sp","accuracy"),
                         best.method="youden"),auc(roc_obj_logi2))
names(roc_logi2) =
c("Threshold","Sensitivity","Specificity","Accuracy","AUC")
t(roc_logi2)

## ROC plot
plot(roc_obj_logi2, legacy.axes=TRUE, print.auc=TRUE,
print.thres=TRUE,
     main = "ROC for the Logistic Regression Model",
     xlab="1-Specificity or False Positive Rate",
     ylab="Sensitivity or True Positive Rate")

## Confusion Matrix for the test data
logi.cm2 = table(test$Response, ifelse(logi.probs2 > roc_logi2[1], 1,
0))
logi.cm2

## Confusion Matrix for the training data
logi.probs2.fit = fitted(logi.mod2)
roc_obj_logi2.fit = roc(response = train$Response, predictor =
logi.probs2.fit)
roc_logi2.fit = c(coords(roc_obj_logi2.fit, "b",
                     ret=c("threshold","se","sp","accuracy"),
                     best.method="youden"),auc(roc_obj_logi2.fit))
names(roc_logi2.fit) =
c("Threshold","Sensitivity","Specificity","Accuracy","AUC")
t(roc_logi2.fit)
table(train$Response, ifelse(logi.probs2.fit > roc_logi2.fit[1], 1,
0))

##### This part is not mentioned in the project content #####
# logi.mod3 is logi.mod2 - Region_Code effect
```

```
logi.mod3 = glm(Response ~ Gender + Age + Driving_License +
Previously_Insured + Vehicle_Age + Vehicle_Damage +
Policy_Sales_Channel, data = train, family = "binomial")
# HL test
hoslem.test(logi.mod3$y, fitted(logi.mod3),g=10)
# p-value < 2.2e-16
###########################################
```

Section 4: LDA and QDA

```
## LDA
lda_mod1 = lda(Response ~ Gender + Age + Driving_License +
Region_Code + Previously_Insured + Vehicle_Age + Vehicle_Damage +
Policy_Sales_Channel, data = train)

lda_pred1 = predict(lda_mod1, newdata = test)$posterior[,2]

## ROC
roc_obj_lda1 = roc(response = test$Response, predictor = lda_pred1)
roc_lda1 = c(coords(roc_obj_lda1, "b",
                     ret=c("threshold","se","sp","accuracy"),
                     best.method="youden"),auc(roc_obj_lda1))
names(roc_lda1) =
c("Threshold","Sensitivity","Specificity","Accuracy","AUC")
t(roc_lda1)

## Confusion Matrix for the test data
lda.cm1 = table(test$Response, ifelse(lda_pred1 > roc_lda1[1], 1, 0))
lda.cm1

## QDA
qda_mod1 = qda(Response ~ Gender + Age + Driving_License +
Region_Code + Previously_Insured + Vehicle_Age + Vehicle_Damage +
Policy_Sales_Channel, data = train)

qda_pred1 = predict(qda_mod1, newdata = test)$posterior[,2]

## ROC
roc_obj_qda1 = roc(response = test$Response, predictor = qda_pred1)
roc_qda1 = c(coords(roc_obj_qda1, "b",
                    ret=c("threshold","se","sp","accuracy"),
                    best.method="youden"),auc(roc_obj_qda1))
names(roc_qda1) =
c("Threshold","Sensitivity","Specificity","Accuracy","AUC")
t(roc_qda1)

## Confusion Matrix for the test data
qda.cm1 = table(test$Response, ifelse(qda_pred1 > roc_qda1[1], 1, 0))
qda.cm1

rbind(roc_logi2, roc_lda1, roc_qda1)
```

Section 5: Resampling

```
## Dealing with imbanlanced data: resampling: SMOTE
prop.table(table(train$Response))
set.seed(1234)
# too slow! This line of resampling needs 6 min to finish running.
smoted_data <- SMOTE(Response~., train[,-c(1, 9, 11)], perc.over =
100)
head(smoted_data)
dim(smoted_data)
# 131188 observations with 9 variables
prop.table(table(smoted_data$Response))

## Simple logistic regression
logi.mod4 = glm(Response ~ ., data = smoted_data, family =
"binomial")
logi.mod5 = step(logi.mod4, trace = 0)
summary(logi.mod5)
# Age + Driving_License + Region_Code + Previously_Insured +
Vehicle_Age + Vehicle_Damage + Policy_Sales_Channel

## HL test
hoslem.test(logi.mod5$y, fitted(logi.mod5),g=10)
# p-value < 2.2e-16

## Predictions for the test data
logi.probs5 = predict(logi.mod5, newdata = test, type = "response")

## ROC
```

```
roc_obj_logi5 = roc(response = test$Response, predictor =
logi.probs5)
roc_logi5 = c(coords(roc_obj_logi5, "b",
                     ret=c("threshold","se","sp","accuracy"),
                     best.method="youden"),auc(roc_obj_logi5))
names(roc_logi5) =
c("Threshold","Sensitivity","Specificity","Accuracy","AUC")
t(roc_logi5)

## Confusion Matrix for the test data
logi.cm5 = table(test$Response, ifelse(logi.probs5 > roc_logi5[1], 1,
0))
logi.cm5

## Dealing with imbanlanced data: resampling: undersampling
rs_data0 = train[train$Response == 0,]
rs_data1 = train[train$Response == 1,]
set.seed(12345678)
rs_data0 = rs_data0[sample(dim(rs_data0)[1], dim(rs_data1)[1]),]
rs.data = bind_rows(rs_data0, rs_data1)

## Simple logistic regression
logi.mod6 = glm(Response ~ ., data = rs.data, family = "binomial")
logi.mod7 = step(logi.mod6, trace = 0)
summary(logi.mod7)
# Gender + Age + Driving_License + Region_Code + Previously_Insured +
Vehicle_Age + Vehicle_Damage + Annual_Premium + Policy_Sales_Channel

## HL test
hoslem.test(logi.mod7$y, fitted(logi.mod7),g=10)
#p-value = 6.692e-07

## Predictions for the test data
logi.probs7 = predict(logi.mod7, newdata = test, type = "response")

## ROC
roc_obj_logi7 = roc(response = test$Response, predictor =
logi.probs7)
roc_logi7 = c(coords(roc_obj_logi7, "b",
                     ret=c("threshold","se","sp","accuracy"),
                     best.method="youden"),auc(roc_obj_logi7))
names(roc_logi7) =
c("Threshold","Sensitivity","Specificity","Accuracy","AUC")
t(roc_logi7)

## Confusion Matrix for the test data
logi.cm7 = table(test$Response, ifelse(logi.probs7 > roc_logi7[1], 1,
0))
logi.cm7

##### This part is not mentioned in the project content #####
logi.mod8 = glm(Response ~ Gender + Age + Driving_License +
Previously_Insured + Vehicle_Age + Vehicle_Damage +
Policy_Sales_Channel, data = rs.data, family = "binomial")
logi.probs8 = predict(logi.mod8, newdata = test, type = "response")
roc_obj_logi8 = roc(response = test$Response, predictor =
logi.probs8)
roc_logi8 = c(coords(roc_obj_logi8, "b",
                     ret=c("threshold","se","sp","accuracy"),
                     best.method="youden"),auc(roc_obj_logi8))
names(roc_logi8) =
c("Threshold","Sensitivity","Specificity","Accuracy","AUC")
t(roc_logi8)
############################
```

Section 6: kNN

```
## This function calculates the Confusion Matrix, sensitivity,
specificity, and accuracy
sen.spe.acc = function(true_types, predicted){
  confu_mat = table(true_types, predicted)
  Sensitivity = confu_mat[2,2] / sum(confu_mat[2,])
  Specificity = confu_mat[1,1] / sum(confu_mat[1,])
  Accuracy = sum(diag(confu_mat)) / sum(confu_mat)
  return(list(ConfusionMatrix = confu_mat, evaluates = c(Sensitivity
= Sensitivity, Specificity = Specificity, Accuracy = Accuracy)))
}

## kNN, it includes too many categorical predictors
## remove useless variables
health2 = health
```

```
#health2$Response = NULL
health2$Region_Code = NULL
health2$id = NULL
health2$Annual_Premium = NULL # include too large numbers

## to proceed knn, we should convert all predictors to numeric and
standardize the data
health2$Gender = ifelse(health2$Gender == "Female", 0, 1)
health2$Driving_License = ifelse(health2$Driving_License == "0", 0,
1)
health2$Previously_Insured = ifelse(health2$Previously_Insured ==
"0", 0, 1)
health2$Vehicle_Age1 = ifelse(health2$Vehicle_Age == "< 1 Year", 0,
1)
health2$Vehicle_Age1_2 = ifelse(health2$Vehicle_Age == "1-2 Year", 0,
1)
health2$Vehicle_Age = NULL
health2$Vehicle_Damage = ifelse(health2$Vehicle_Damage == "No", 0, 1)
health2$Age = scale(health2$Age)
health2$Policy_Sales_Channel = scale(health2$Policy_Sales_Channel)
health2$Vintage = scale(health2$Vintage)

## seperate train and test set
x_train = health2[index, -8]
x_test = health2[-index, -8]
y_train = health2[index, 8]
y_test = health2[-index, 8]

## I don't know why the CV code from the course slides always turned
out to be an error (saying the f1 metric doesn't work) so I wrote a
funtion to calculate f1 score. And this is for no CV procedure.
## calculate f1 score from a confusion matrix
f1score = function(confumat){
  precision = confumat[2,2] / sum(confumat[,2])
  sensitivity = confumat[2,2] / sum(confumat[2,])
  2*precision*sensitivity / (precision + sensitivity)
}

## Find the f1 score for k = 1 to 10
f1_1 = rep(0, 10)
```

```
for (i in 1:10) {
  knn_pred = knn(train = x_train, test = x_train, cl = y_train, k =
i)
  f1_1[i] = f1score(sen.spe.acc(y_train, knn_pred)$ConfusionMatrix)
  if(which.max(f1_1[1:i]) == i){
    knn_fit1 = knn_pred
    max_k_1 = i
  }
  print(i)
}

knn_pred1 = knn(train = x_train, test = x_test, cl = y_train, k =
max_k_1)
sen.spe.acc(y_test, knn_pred1)

## kNN with undersampling data
x_train0 = x_train[y_train == 0,]
x_train1 = x_train[y_train == 1,]
set.seed(12345678)
index_resample = sample(sum(y_train == 0), sum(y_train == 1))
x_train0 = x_train0[index_resample,]
y_train0 = y_train0[index_resample]
x_train_re = bind_rows(x_train0, x_train1)
y_train_re = factor(rep(c(0, 1), each = sum(y_train == 1)))
head(x_train_re)
rs.data_kNN = mutate(x_train_re, Response = y_train_re)

## Find the f1 score for k = 1 to 10
f1_2 = rep(0, 10)

for (i in 1:10) {
  knn_pred = knn(train = x_train_re, test = x_train_re, cl =
y_train_re, k = i)
  f1_2[i] = f1score(sen.spe.acc(y_train_re,
knn_pred)$ConfusionMatrix)
  if(which.max(f1_2[1:i]) == i){
    knn_fit2 = knn_pred
    max_k_2 = i
  }
  print(i)
}

knn_pred2 = knn(train = x_train_re, test = x_test, cl = y_train_re, k
= max_k_2)
sen.spe.acc(y_test, knn_pred2)

## Plot the f1 scores
data.frame(f1_score = c(f1_1, f1_2),
           k = rep(1:10, 2),
           source = factor(rep(c("The full train dataset", "The
undersampling dataset"), each = 10))) %>%
  ggplot(aes(y = f1_score, x = k, color = source, linetype = source))
+
  geom_line(size = 1) +
  geom_point() +
  labs(title="f1 scores for kNN", x="k (Number of Nearest
Neighbours)", y="f1 scores", color = "Dataset", linetype = "Dataset")

## kNN: adding variable region code
health3 = with(health2, data.frame(Gender, Age, Driving_License,
class.ind(health$Region_Code), Previously_Insured, Vehicle_Age1,
Vehicle_Age1_2, Vehicle_Damage, Policy_Sales_Channel, Vintage,
Response))
x_train3 = health3[index, -63]
x_test3 = health3[-index, -63]

## This line requires 80 min.
knn_pred3 = knn(train = x_train3, test = x_test3, cl = y_train, k =
3)
sen.spe.acc(y_test, knn_pred3)
```

Section 7: Trees, random forests, boostings and xgboostings

```
## Because factor predictors must have at most 32 levels in this
function, I removed the variable Region_Code
## tree with the full train data
tree.mod1 = tree(Response ~ Gender + Age + Driving_License +
Previously_Insured + Vehicle_Age + Vehicle_Damage + Annual_Premium +
Policy_Sales_Channel + Vintage,split="deviance",data = train)

set.seed(31)
cv.tree1 = cv.tree(tree.mod1)
```

```r
plot(cv.tree1$size, cv.tree1$dev, type='b',cex=2)
# This tree does not need to be pruned
#cv.tree1$size[which(cv.tree1$dev == min(cv.tree1$dev))]
#tree.mod1.p = prune.misclass(tree.mod1, best = 6)
par(mfrow = c(1, 3))
plot(tree.mod1)
text(tree.mod1)
pred.tree1 = predict(tree.mod1, newdata = test, type = "class")
#fitted.probs.tree1 = predict(tree.mod1)[,2]
sen.spe.acc(test$Response, pred.tree1)
# This result is terrible

## tree with undersampling data
tree.mod2 = tree(Response ~ Gender + Age + Driving_License +
Previously_Insured + Vehicle_Age + Vehicle_Damage + Annual_Premium +
Policy_Sales_Channel + Vintage,split="deviance",data = rs.data)
set.seed(31)
cv.tree2 = cv.tree(tree.mod2)
plot(cv.tree2$size, cv.tree2$dev, type='b',cex=2)
plot(tree.mod2)
text(tree.mod2)
pred.tree2 = predict(tree.mod2, newdata = test, type = "class")
sen.spe.acc(test$Response, pred.tree2)
tree.mod2

## plot the tree result vs response
fig12 = rs.data %>%
  mutate(terminal3 = as.factor((Previously_Insured == "0") *
(Vehicle_Damage == "Yes") * (Age >= 26.5))) %>%
  ggplot(aes(x = terminal3, fill = Response)) +
  geom_bar() +
  labs(title='Tree result vs Response in the balanced dataset', x =
"Whether in terminal 3")

fig13 = health %>%
  mutate(terminal3 = as.factor((Previously_Insured == "0") *
(Vehicle_Damage == "Yes") * (Age >= 26.5))) %>%
  ggplot(aes(x = terminal3, fill = Response)) +
  geom_bar() +
  labs(title='Tree result vs Response in the whole dataset', x =
"Whether in terminal 3")

grid.arrange(fig12, fig13, nrow=1, ncol=2)

## random forest with the full train data
set.seed(1)
#within 10 min
rf.mod1 = randomForest(Response ~ Gender + Age + Driving_License +
Previously_Insured + Vehicle_Age + Vehicle_Damage + Annual_Premium +
Policy_Sales_Channel, data = train, mtry = 3,importance = T)
pred.rf1 = predict(rf.mod1, newdata = test)
sen.spe.acc(test$Response, pred.rf1)
# The above result is really influenced by the imbanlance of data

## random forest with undersampling data
set.seed(13579)
rf.mod2 = randomForest(Response ~ Gender + Age + Driving_License +
Previously_Insured + Vehicle_Age + Vehicle_Damage + Annual_Premium +
Policy_Sales_Channel, data = rs.data, mtry = 3,importance = T)
pred.rf2 = predict(rf.mod2, newdata = test)
sen.spe.acc(test$Response, pred.rf2)

## boosting with undersampling data
set.seed(1)
boost.mod1 = gbm(as.numeric(as.character(Response)) ~ Gender + Age +
Driving_License + Previously_Insured + Vehicle_Age + Vehicle_Damage +
Annual_Premium + Policy_Sales_Channel, data = rs.data,
distribution="bernoulli", n.trees = 5000, interaction.depth = 4)
pred.bst1 = predict(boost.mod1, newdata = test, n.trees=5000, type =
"response")

## ROC
roc_obj_bst1 = roc(response = test$Response, predictor = pred.bst1)
roc_bst1 = c(coords(roc_obj_bst1, "b",
                     ret=c("threshold","se","sp","accuracy"),
                     best.method="youden"),auc(roc_obj_bst1))
names(roc_bst1) =
c("Threshold","Sensitivity","Specificity","Accuracy","AUC")
t(roc_bst1)
```

```
## Confusion Matrix for the test data
bst.cm1 = table(test$Response, ifelse(pred.bst1 > roc_bst1[1], 1, 0))
bst.cm1

## xgboost
dtrain <- xgb.DMatrix(data = as.matrix(x_train3), label =
as.logical(as.numeric(as.character(y_train))))
dtest <- xgb.DMatrix(data = as.matrix(x_test3), label =
as.logical(as.numeric(as.character(y_test))))

negative_cases = sum(y_train == "0")
postive_cases = sum(y_train == "1")

xgboost.mod1 = xgboost(data = dtrain, # the data
                       max.depth = 6, # the maximum depth of each
decision tree
                       nround = 50000, # number of boosting rounds
                       early_stopping_rounds = 5, # if we dont see an
improvement in this many rounds, stop
                       objective = "binary:logistic", # the objective
function
                       scale_pos_weight =
negative_cases/postive_cases, # control for imbalanced classes
                       gamma = 1) # add a regularization term
pred.xgbst1 <- predict(xgboost.mod1, dtest)

## ROC
roc_obj_xgbst1 = roc(response = test$Response, predictor =
pred.xgbst1)
roc_xgbst1 = c(coords(roc_obj_xgbst1, "b",
                    ret=c("threshold","se","sp","accuracy"),
                    best.method="youden"),auc(roc_obj_xgbst1))
names(roc_xgbst1) =
c("Threshold","Sensitivity","Specificity","Accuracy","AUC")
t(roc_xgbst1)

## Confusion Matrix for the test data
xgbst.cm1 = table(test$Response, ifelse(pred.xgbst1 > roc_xgbst1[1],
1, 0))
```

```
xgbst.cm1
```

Section 8: Neural Networks

```
# some data processing needed in order to use neuralnet()

# this function can't use factor predictor variables,

# so we convert them to dummy variables

train1 = train

train1$Gender = as.numeric(train1$Gender)-1

train1$Driving_License = as.numeric(train1$Driving_License)-1

train1$Previously_Insured = as.numeric(train1$Previously_Insured)-1

train1$Vehicle_Damage =
as.numeric(as.factor(train1$Vehicle_Damage))-1

test1 = test

test1$Gender = as.numeric(test1$Gender)-1

test1$Driving_License = as.numeric(test1$Driving_License)-1

test1$Previously_Insured = as.numeric(test1$Previously_Insured)-1

test1$Vehicle_Damage = as.numeric(as.factor(test1$Vehicle_Damage))-1


# fit a neural net using all predictors with 2 layers containing 2
and 1 nodes respectively

nn <- neuralnet(Response ~ Gender + Age + Driving_License +
Previously_Insured +

                  Vehicle_Damage + Annual_Premium + Vintage,
data=train1, hidden=c(2,1),

                linear.output=FALSE)

# Confusion Matrix

prob <- compute(nn, test1)$net.result[,2]
```

```
pred <- ifelse(prob>0.5, 1, 0)

confusionMatrix(as.factor(pred), test$Response)$table


# fit a neural net using all predictors with 2 layers containing 5
and 3 nodes respectively

nn2 <- neuralnet(Response ~ Gender + Age + Driving_License +
Previously_Insured +

                  Vehicle_Damage + Annual_Premium + Vintage,
data=train1, hidden=c(5,3),

                linear.output=FALSE)

# Confusion Matrix

prob2 <- compute(nn2, test1)$net.result[,2]

pred2 <- ifelse(prob2>0.5, 1, 0)

confusionMatrix(as.factor(pred2), test$Response)$table


# fit a neural net using all predictors with 2 layers containing 5
and 3 nodes respectively,this model uses a subset of the predictors
only

# Inlcudes: Gender, Age, Driving License, Previously_Insured,
Vehicle_Age, Vehicle_Damage, Annual_Premium and Vintage

nn3 <- neuralnet(Response ~ Gender + Age + Driving_License +
Previously_Insured +

                  Vehicle_Damage + Annual_Premium + Vintage,
data=train1[,c(2,3,4,6,7,8,9,11,12)], hidden=c(5,3),

                linear.output=FALSE, threshold=0.01)

# Confusion Matrix

prob3 <- compute(nn3, test1)$net.result[,2]

pred3 <- ifelse(prob3>0.5, 1, 0)
```

```
confusionMatrix(as.factor(pred3), test$Response)$table
```