

# Final Project : Credit Card Fraud Detection

Dan Zylkowski

February 27th 2020

## Abstract

Credit card fraud is the most common type of identity fraud, and businesses are desperate to find a solution to stop the loss it creates. A recent study by [Juniper Research](#) has found that retailers are expected to lose \$130 billion in digital CNP (Card-not-Present) fraud between 2018 and 2023. Businesses need to be able to detect fraudulent credit card transactions, and machine learning could be a solution.

## Introduction

This project will examine how machine learning algorithms can be used to predict if a credit card transaction from the data set is fraudulent or not. Multiple data resampling methods will be used on the imbalanced data to train a machine learning model, and the results will be compared. A single resampling method will then be chosen and used on the imbalanced data to fit multiple machine learning models, and the model with the best ROC performance on the test set will be identified. Variable importance for a given model will be determined, which will show which features are the most important predictors for the model. Finally, an ensemble model will be created to combine the predictions of different models. The tools in the [caret package](#) will be used to handle the entire process of training, testing, predicting, and comparing the machine learning models in this project.

## Preparations

The following packages will be used for this analysis:

```
library(readr) # For loading the data
library(skimr) # To skim the data
library(corrplot) # To plot corrplot
library(caret) # For machine learning
library(class) # For machine learning
library(ggplot2) # For graphics
library(e1071) # For statistics
library(DMwR) # For SMOTE sampling
library(ROSE) # For ROSE sampling
library(doParallel) # For parallel processing
library(dplyr) # For data manipulation
library(caretEnsemble) # For model ensembling
library(pROC) # For ROC
```

## Dataset

The dataset contains transactions made by credit cards in September 2013 by European cardholders over the course of two days. There are 492 fraudulent transactions out of 284,807 total transactions. The dataset is highly unbalanced, with the positive class (fraudulent transactions) accounting for 0.172% of all transactions. Due to this extreme imbalance, ROC will be used as the model metric instead of accuracy. The dataset can be downloaded at the following URL (<https://www.kaggle.com/mlg-ulb/creditcardfraud/download>). There are 31 numerical variables in total. The variables are named V1, V2,...V28, and are the result of principal component analysis (PCA) transformation. PCA is used to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. There are two additional variables, labeled Time and Amount. Time represents the number of seconds between each transaction and the first transaction, and Amount is the transaction amount. Finally, there is the Class variable which equals 0 when the transaction is legitimate, and 1 when the transaction is fraudulent. Now let's take a quick look at the data set using the skim() function.

### Data summary




Name	creditcard
Number of rows	284807
Number of columns	31

### Column type frequency:

numeric	31
---------	----

Group variables	None
-----------------	------

### Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Time	0	1	9481 3.86	4748 8.15	0.0 0	5420 1.50	8469 2.00	13932 0.50	17279 2.00	
V1	0	1	0.00	1.96	- 56. 41	-0.92	0.02	1.32	2.45	__ -- 
V2	0	1	0.00	1.65	- 72. 72	-0.60	0.07	0.80	22.06	__ -- 
V3	0	1	0.00	1.52	-	-0.89	0.18	1.03	9.38	__

					48. 33						--■
V4	0	1	0.00	1.42	- 5.6 8	-0.85	-0.02	0.74	16.88	■--	
V5	0	1	0.00	1.38	- 113 .74	-0.69	-0.05	0.61	34.80	-- ■--	
V6	0	1	0.00	1.33	- 26. 16	-0.77	-0.27	0.40	73.30	■--	
V7	0	1	0.00	1.24	- 43. 56	-0.55	0.04	0.57	120.5 9	■--	
V8	0	1	0.00	1.19	- 73. 22	-0.21	0.02	0.33	20.01	-- ■--	
V9	0	1	0.00	1.10	- 13. 43	-0.64	-0.05	0.60	15.59	--■	
V10	0	1	0.00	1.09	- 24. 59	-0.54	-0.09	0.45	23.75	--■	
V11	0	1	0.00	1.02	- 4.8 0	-0.76	-0.03	0.74	12.02	■--	
V12	0	1	0.00	1.00	- 18. 68	-0.41	0.14	0.62	7.85	-- ■--	
V13	0	1	0.00	1.00	- 5.7 9	-0.65	-0.01	0.66	7.13	■--	
V14	0	1	0.00	0.96	- 19. 21	-0.43	0.05	0.49	10.53	-- ■--	
V15	0	1	0.00	0.92	- 4.5 0	-0.58	0.05	0.65	8.88	■--	
V16	0	1	0.00	0.88	- 14. 13	-0.47	0.07	0.52	17.32	--■	
V17	0	1	0.00	0.85	-	-0.48	-0.07	0.40	9.25	--	

					25.16						
V18	0	1	0.00	0.84	-	-0.50	0.00	0.50	5.04		
					9.50						
V19	0	1	0.00	0.81	-	-0.46	0.00	0.46	5.59		
					7.21						
V20	0	1	0.00	0.77	-	-0.21	-0.06	0.13	39.42		
					54.50						
V21	0	1	0.00	0.73	-	-0.23	-0.03	0.19	27.20		
					34.83						
V22	0	1	0.00	0.73	-	-0.54	0.01	0.53	10.50		
					10.93						
V23	0	1	0.00	0.62	-	-0.16	-0.01	0.15	22.53		
					44.81						
V24	0	1	0.00	0.61	-	-0.35	0.04	0.44	4.58		
					2.84						
V25	0	1	0.00	0.52	-	-0.32	0.02	0.35	7.52		
					10.30						
V26	0	1	0.00	0.48	-	-0.33	-0.05	0.24	3.52		
					2.60						
V27	0	1	0.00	0.40	-	-0.07	0.00	0.09	31.61		
					22.57						
V28	0	1	0.00	0.33	-	-0.05	0.01	0.08	33.85		
					15.43						
Amount	0	1	88.35	250.12	0.0	5.60	22.00	77.16	25691.16		
					0.0	0.00	0.00	0.00	1.00		
Class	0	1	0.00	0.04	0.0	0.00	0.00	0.00	1.00		

We can see from above that there are no missing values, and we can also see that the range of the Time and Amount variables are much larger than the other predictor variables. This

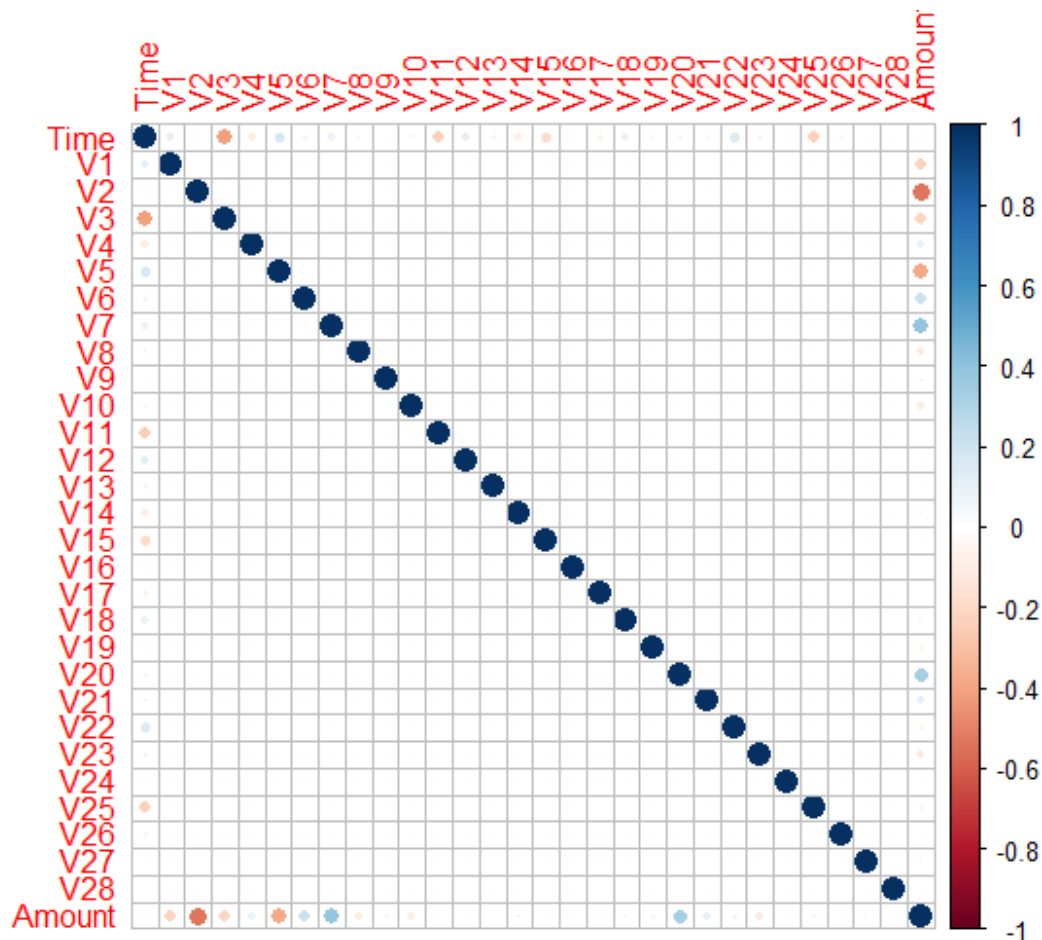
makes sense because the other variables have been PCA transformed. We may need to scale and center the Time and Amount before trying to fit any machine learning models. Now let's take a look at the distribution of the data.

### Distribution of the data

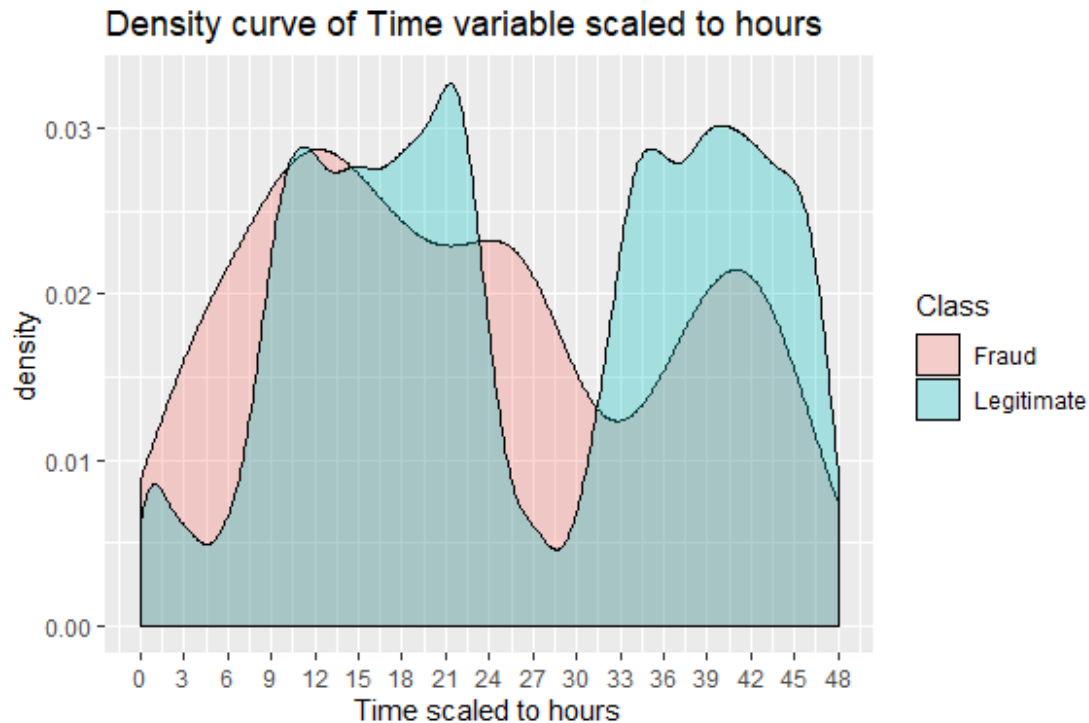
```
# Changing class level, and making Class a factor
creditcard$Class[creditcard$Class==0]<- "Legitimate"
creditcard$Class[creditcard$Class==1]<- "Fraud"
creditcard$Class <- factor(creditcard$Class)
```

The data has been loaded into a dataframe called creditcard. The Class variable has been changed to a factor since this is a classification problem. As mentioned previously, the feature variables V1 through V28 have undergone PCA transformation, so these variables are not linearly uncorrelated. We can verify the predictor variable correlations by looking at a plot of the correlation matrix by using the corrplot function from the corrplot library. We can see in the correlation plot below that none of the PCA transformed variables are correlated.

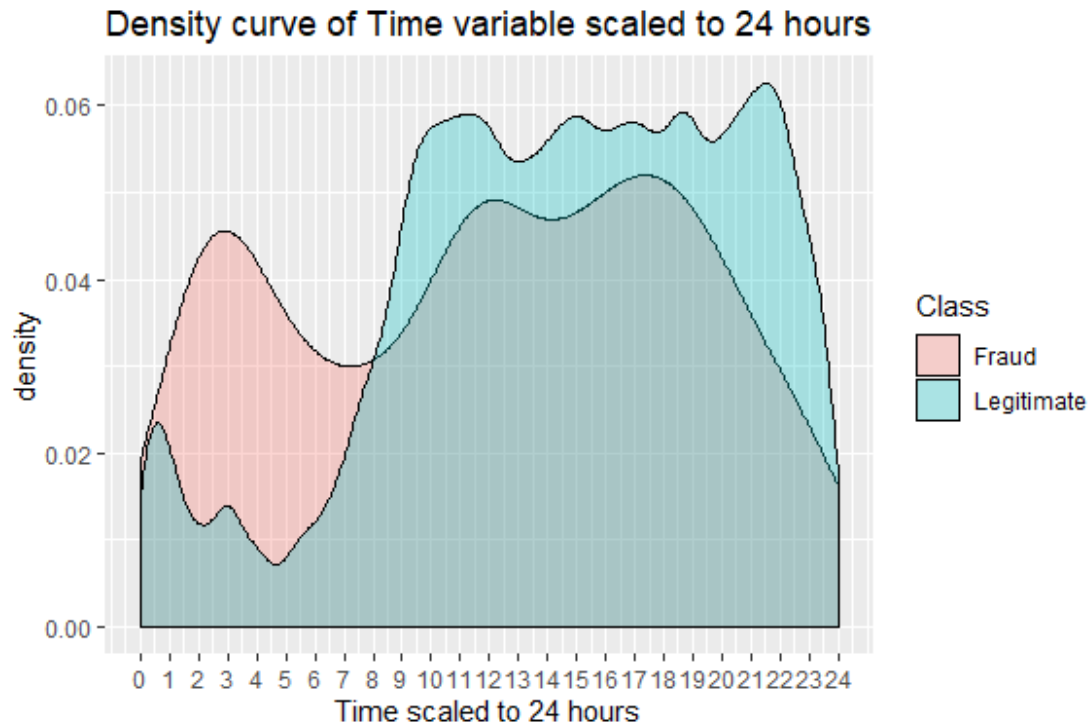
```
predictor_correlations <- cor(creditcard[,31])
corrplot(predictor_correlations)
```



We will now plot the distribution of the Time and Amount variables. First, we will look at the density curve of the Time variable. The Time variable represents the number of seconds a transaction occurred after the first transaction. We will scale the Time variable from seconds to hours and plot the density.

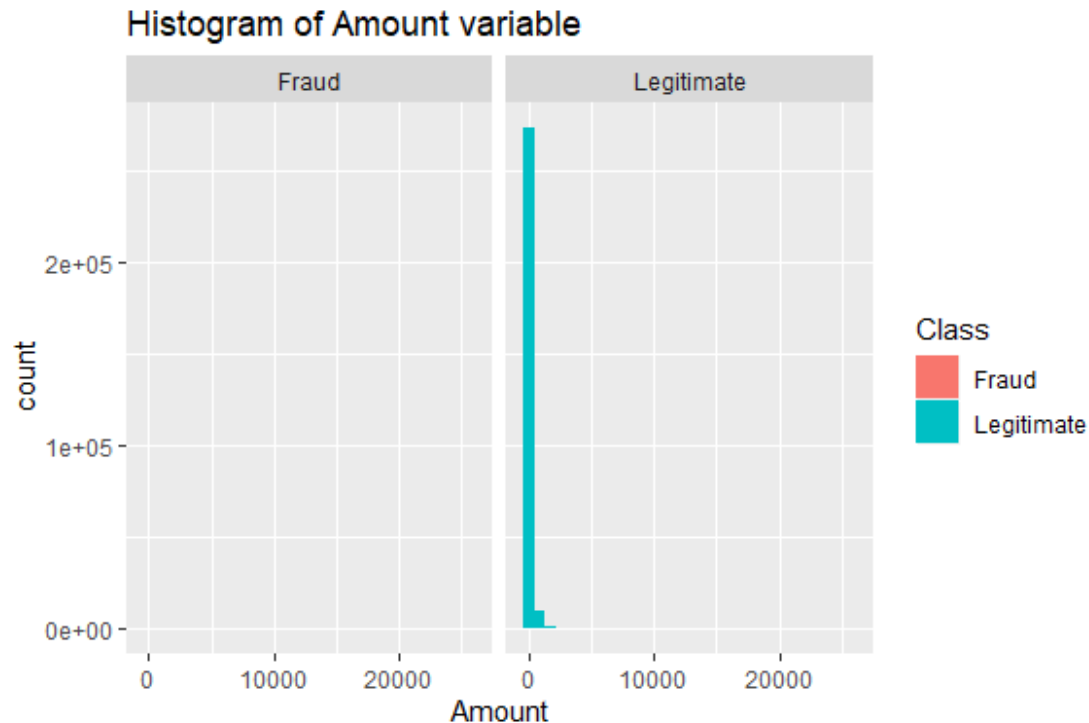


We can see there are 48 hours of transactions, which coincides with the transactions occurring over two days. To determine the relative time of day that each transaction occurred, we will scale the Time variable to a 24-hour scale. Below is the density plot of the Time variable scaled to 24 hours.



We can see in the 24 hour Time density plot that between hours 1-7 that the density plot of fraudulent transactions is much higher than the density plot of legitimate transactions. While some insights could be drawn from the Time variable, it seems to be more of a transaction ordering variable.

Next, we will look at a histogram plot of the Amount variable. We can see that the large and imbalanced data set makes it difficult to identify new information from the histogram.



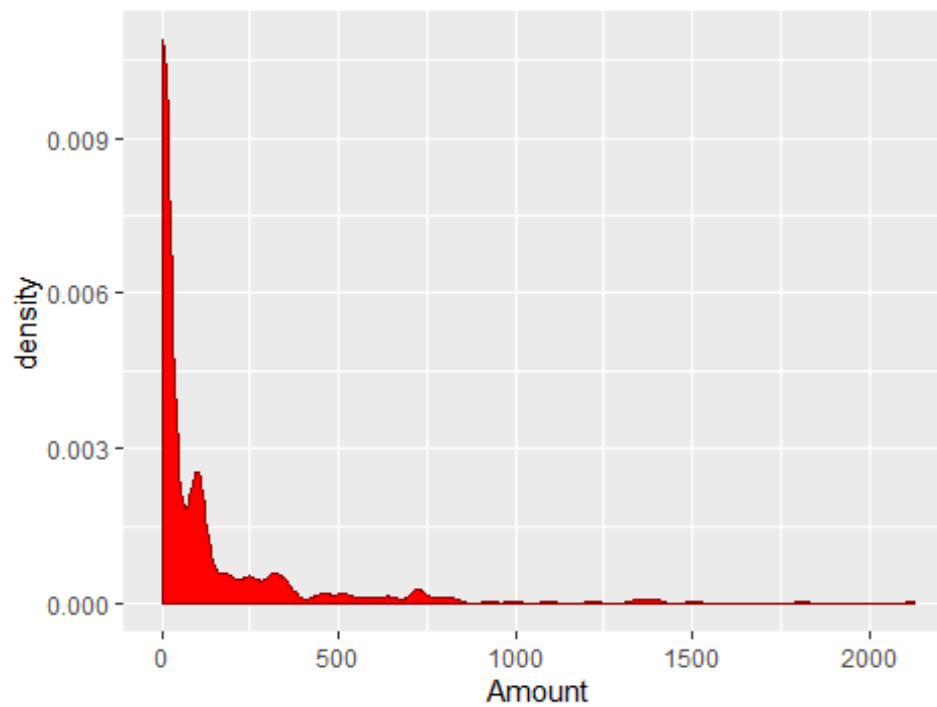
We can take a closer look at the Amount variable by filtering transaction amounts by fraud type. Since the dataset is so large, we will create two new dataframes, one containing the Amount variable for fraudulent transactions, and one containing the Amount variable for legitimate transactions. Then we will plot the density curve for each Amount variable.

```
fraud_only<- (creditcard %>% filter(Class == "Fraud"))[,30]
legit_only <- (creditcard %>% filter(Class == "Legitimate"))[, 30]

ggplot(fraud_only, aes(x=Amount)) +
  geom_density(color="darkred", fill="red")+
  ggtitle("Density curve of Amount variable for fraudulent transactions")
```

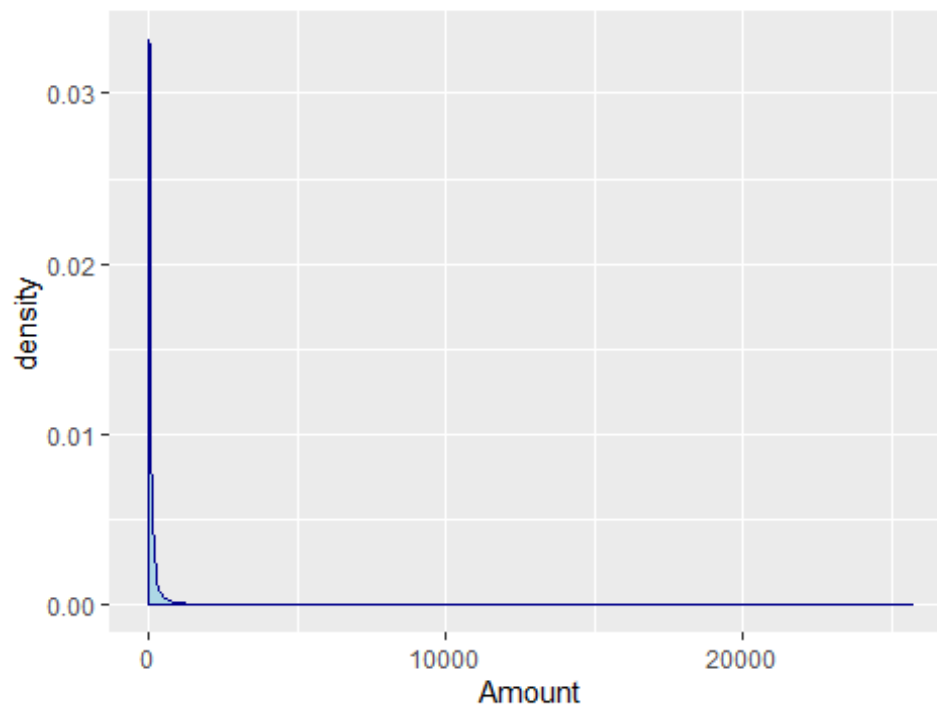


Density curve of Amount variable for fraudulent trans



```
ggplot(legit_only, aes(x=Amount)) +  
  geom_density(color="darkblue", fill="lightblue") +  
  ggtitle("Density curve of Amount variable for legitimate transactions")
```

Density curve of Amount variable for legitimate transa



The Amount variable is skewed highly to the right. We will scale and center the Time and Amount variables before fitting machine learning models since the PCA transformed variables have already been centered and scaled.

```
# Scale and center the Amount and time variables
creditcard$Amount <- scale(creditcard$Amount, center = TRUE, scale = TRUE)
creditcard$Time <- scale(creditcard$Time, center = TRUE, scale = TRUE)
```

## Data Splitting

We will split the data into a training set and testing set using stratified sampling of the Class variable, by using the createDataPartition() function in the caret package. The function ensures that the highly imbalanced structure of the original data set is preserved in both the training and test sets.

```
# Create the training and test sets
set.seed(2020)
index <- createDataPartition(creditcard$Class, p = .70,
                             list = FALSE,
                             times = 1)
imbal_train <- creditcard[index,]
test_data <- creditcard[-index,]
```

## Imbalanced data set

We can verify the imbalanced nature of the data set by using the table() and prop.table() functions from base R. We can see below that the Class imbalance is 99.83% to 0.17% in the original set, the training set, and the testing set.

```
prop.table(table(creditcard$Class))*100
```

```
      Fraud Legitimate
0.1727486 99.8272514
```

```
prop.table(table(imbal_train$Class))*100
```

```
      Fraud Legitimate
0.1730486 99.8269514
```

```
prop.table(table(test_data$Class))*100
```

```
      Fraud Legitimate
0.1720485 99.8279515
```

Imbalanced data sets can be difficult for classifier algorithms since they have a bias toward the majority class. The minority class is often ignored, and only the majority class is predicted, which in this case lead to an overall 99.83% accuracy rate, but a 0% rate of fraud detection. As a result, there is a high probability of misclassification of the minority class as

compared to the majority class. One way of dealing with this challenge is by balancing classes in the training data before providing the data as input to a machine learning algorithm. It is important to note that all sampling will be applied to the imbalanced\_train set. We should not resample the test set, as it needs to maintain the class imbalance that we would expect to see “in the wild.”

We will next look at down-sampling, up-sampling, and hybrid sampling methods in the following sections, and use the caret package with these methods to sample inside of cross-validation.

## Resampling methods implemented in caret

We first give an overview of the idea behind each sampling method.

Down-sampling is used to randomly subset the majority class in the training set so that class frequency matches the minority class.

Up-sampling randomly resamples the minority class in the training set so the class frequency matches the majority class.

Hybrid sampling methods like [SMOTE](#) and [ROSE](#) down-sample the majority class and synthesize new data points in the minority class. We will also use the [ovun.sample\(\)](#) function to create an evenly balanced data set from the imbalanced train set.

With the exception of the `ovun.sample()` method, caret offers built-in functionality to resample data using all of the above methods on the training data during the model building-process. To use the `ovun.sample()` method, we will need to create a balanced set before using caret.

## Resampling the training set

We start by using the `trainControl()` and `train()` functions in the caret package to setup K-fold cross-validation and fit a Recursive Partitioning and Regression Trees (rpart) model to the `imbal_train` data without any re-sampling. We will then in turn apply down-sampling, up-sampling, ROSE sampling, and SMOTE sampling to the `imbal_train` set and fit an rpart model for each. The sampling method is specified inside of the `trainControl()` function by setting `sampling = “up”, “down”, “rose”, or “smote”` accordingly. The sampling method can also be specified by changing the `control$sampling` variable directly. The `ovun.sample()` sampling method is not supported inside of `trainControl` and must be used outside of `trainControl` before fitting the model.

```
Mycluster <- makeCluster(detectCores()-2)
registerDoParallel(Mycluster)

# Generate evenly balanced data set using ovun.sample()
balanced_train <- ovun.sample(Class ~ ., data = imbal_train, method = "both",
p=0.5, N=30000, seed = 2020)$data

# No sampling is specified
```

```

control <- trainControl(method="cv", number=10,
                        summaryFunction=twoClassSummary,
                        savePredictions=TRUE, classProbs=TRUE,
                        allowParallel = TRUE)

# Model fit to imbal_train with no sampling
set.seed(2020)
rpart_imbalanced <- train(Class~., data=imbal_train,
                          method="rpart", metric="ROC", trControl=control)

# Model fit to balanced_train by setting data=balanced_train
set.seed(2020)
rpart_balanced <- train(Class~., data=balanced_train,
                       method="rpart", metric="ROC", trControl=control)

# Sampling set to "down"
control <- trainControl(method="cv", number=10,
                        summaryFunction=twoClassSummary,
                        savePredictions=TRUE, classProbs=TRUE,
                        allowParallel = TRUE, sampling = "down")

# Model fit to imbal_train with down-sampling
set.seed(2020)
rpart_down <- train(Class~., data=imbal_train,
                   method="rpart", metric="ROC", trControl=control)

# Sampling set to "up" by setting control$sampling = "up"
control$sampling = "up"

# Model fit to imbal_train with up-sampling
set.seed(2020)
rpart_up <- train(Class~., data=imbal_train,
                 method="rpart", metric="ROC", trControl=control)

# Sampling set to "rose" by setting control$sampling = "rose"
control$sampling = "rose"

# Model fit to imbal_train with ROSE sampling
set.seed(2020)
rpart_rose <- train(Class~., data=imbal_train,
                   method="rpart", metric="ROC", trControl=control)

# Sampling set to "smote" by setting control$sampling = "smote"
control$sampling = "smote"

# Model fit to imbal_train with SMOTE sampling
set.seed(2020)
rpart_smote <- train(Class~., data=imbal_train,
                    method="rpart", metric="ROC", trControl=control)

```

## Comparing resampling methods

In the table below, we can see how poorly the original imbalanced train set compared to the other sampled sets.

```
models <- list(original = rpart_imbalanced,
               balanced = rpart_balanced,
               down = rpart_down,
               up = rpart_up,
               ROSE = rpart_rose,
               SMOTE = rpart_smote)
```

```
resampling <- resamples(models)
summary(resampling, metric = "ROC")
```

Call:

```
summary.resamples(object = resampling, metric = "ROC")
```

Models: original, balanced, down, up, ROSE, SMOTE

Number of resamples: 10

ROC

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
original	0.7998342	0.8245852	0.8477794	0.8478267	0.8675545	0.8998392	0
balanced	0.9330937	0.9478325	0.9510458	0.9492795	0.9521720	0.9559500	0
down	0.8258245	0.9004178	0.9164040	0.9126810	0.9334570	0.9719375	0
up	0.8284623	0.9030934	0.9197295	0.9176435	0.9378599	0.9858054	0
ROSE	0.8467855	0.9146448	0.9182229	0.9215489	0.9429103	0.9753148	0
SMOTE	0.8172666	0.9007253	0.9152029	0.9109225	0.9273129	0.9742582	0

## Comparing model test set results

To compare the results of each sampling method on the testing set we first define a function to aggregate the ROC results from the model testing. Then we will create a dataframe with the test results. The code below was adapted from the [caret website](#).

```
test_roc <- function(model, data) {
  roc_obj <- roc(data$class,
                predict(model, data, type = "prob")[, "Legitimate"],
                levels = c("Fraud", "Legitimate"))
  ci(roc_obj)
}
```

```
inside_test <- lapply(models, test_roc, data = test_data)
inside_test <- lapply(inside_test, as.vector)
inside_test <- do.call("rbind", inside_test)
colnames(inside_test) <- c("lower", "ROC", "upper")
```

```
inside_test <- as.data.frame(inside_test)
inside_test
```

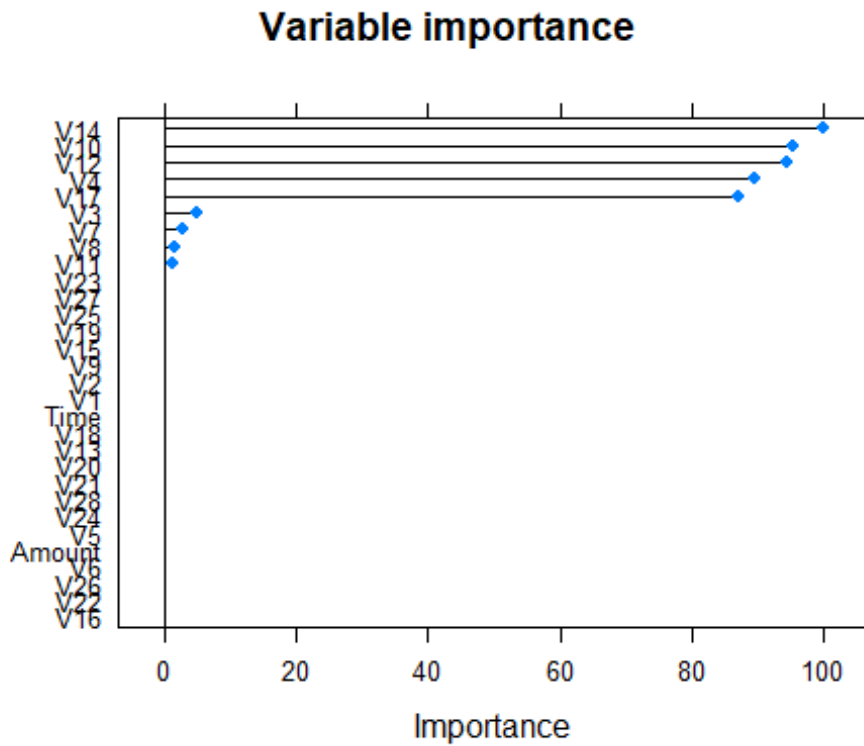
	lower	ROC	upper
original	0.8057719	0.8433820	0.8809922
balanced	0.9262292	0.9481982	0.9701673
down	0.8949412	0.9221552	0.9493691
up	0.8940792	0.9212934	0.9485077
ROSE	0.9066226	0.9325702	0.9585179
SMOTE	0.8993219	0.9265341	0.9537463

In the table above, we again see how poorly the model fit to the original imbalanced train set performed. The model fit to every other sampling method performed significantly better (up to a nearly 9% ROC increase). We can see from the test results that the balanced data set has the highest ROC. We will therefore use the balanced\_train data set when fitting subsequent models.

## Calculate variable importance

The caret package provides tools to report on the importance of variables in your data when fit to a given model. We will calculate the variable importance for the rpart\_balanced model that was previously fit. We can also graph the results using the plot() function. From the table below, we can see that V14, V10, V12, V4, V17, V3, V7, V8, and V11 are the most important variables in the model. Keep in mind that variable importance is dependent on the model, so it might change if a different model is examined. In other words, we should not discard any variables based on the variable importance of a single model. Variable importance can be used to eliminate redundant features that are highly correlated. For example, if there were multiple highly correlated (correlation >0.75) variables that also had high importance in a model, we could potentially discard one or more of them from the model. In this case, since the variables V1, V2,..., V28 are PCA transformed, they are uncorrelated by design, and none should not be discarded.

```
importance <- varImp(rpart_balanced)
plot(importance, main="Variable importance")
```



importance

rpart variable importance

only 20 most important variables shown (out of 30)

	Overall
V14	100.000
V10	95.112
V12	94.227
V4	89.338
V17	87.109
V3	4.947
V7	2.986
V8	1.799
V11	1.367
V5	0.000
V21	0.000
Amount	0.000
V15	0.000
V6	0.000
V22	0.000
V27	0.000
V9	0.000
V20	0.000
V19	0.000
V18	0.000

## Fitting Multiple Models

The caretEnsemble package allows for multiple models to be fit inside of the trainControl() function. To do this, we need only specify the model by tag. The tag for each of the 238 available machine learning models can be found on the official caret website [here](#). We will fit five machine models in the code below: gradient boosting (gbm), linear discriminant analysis (lda), support vector machine with a radial kernel basis (svmradial), naive Bayes (nb), and recursive partitioning and regression trees (rpart).

```
control_multiple <- trainControl(method="cv", number=10,
                                summaryFunction=twoClassSummary,
                                savePredictions=TRUE, classProbs=TRUE,
                                allowParallel = TRUE)
algorithmList <- c('gbm', 'lda', 'svmRadial', 'rpart', 'nb')
set.seed(2020)

model_list <- caretList(Class~., data=balanced_train,
                        trControl=control_multiple, methodList=algorithmList)
```

Iter	TrainDeviance	ValidDeviance	StepSize	Improve
1	1.2409	nan	0.1000	0.0729
2	1.1217	nan	0.1000	0.0595
3	1.0173	nan	0.1000	0.0523
4	0.9324	nan	0.1000	0.0426
5	0.8605	nan	0.1000	0.0361
6	0.7980	nan	0.1000	0.0311
7	0.7443	nan	0.1000	0.0267
8	0.6946	nan	0.1000	0.0247
9	0.6498	nan	0.1000	0.0224
10	0.6119	nan	0.1000	0.0188
20	0.3894	nan	0.1000	0.0064
40	0.2457	nan	0.1000	0.0016
60	0.1992	nan	0.1000	0.0010
80	0.1662	nan	0.1000	0.0003
100	0.1450	nan	0.1000	0.0006
120	0.1261	nan	0.1000	0.0008
140	0.1141	nan	0.1000	0.0002
150	0.1071	nan	0.1000	0.0003

```
results <- resamples(model_list)
summary(results)
```

Call:

```
summary.resamples(object = results)
```

Models: gbm, lda, svmRadial, rpart, nb

Number of resamples: 10

ROC



	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
gbm	0.9986212	0.9992089	0.9994151	0.9992936	0.9994939	0.9995675	0
lda	0.9706014	0.9736130	0.9745217	0.9745753	0.9757583	0.9784451	0
svmRadial	0.9980589	0.9983906	0.9985764	0.9986801	0.9989313	0.9995194	0
rpart	0.9392635	0.9448075	0.9490006	0.9490884	0.9538434	0.9589280	0
nb	0.9663484	0.9691016	0.9709916	0.9713758	0.9730676	0.9774696	0

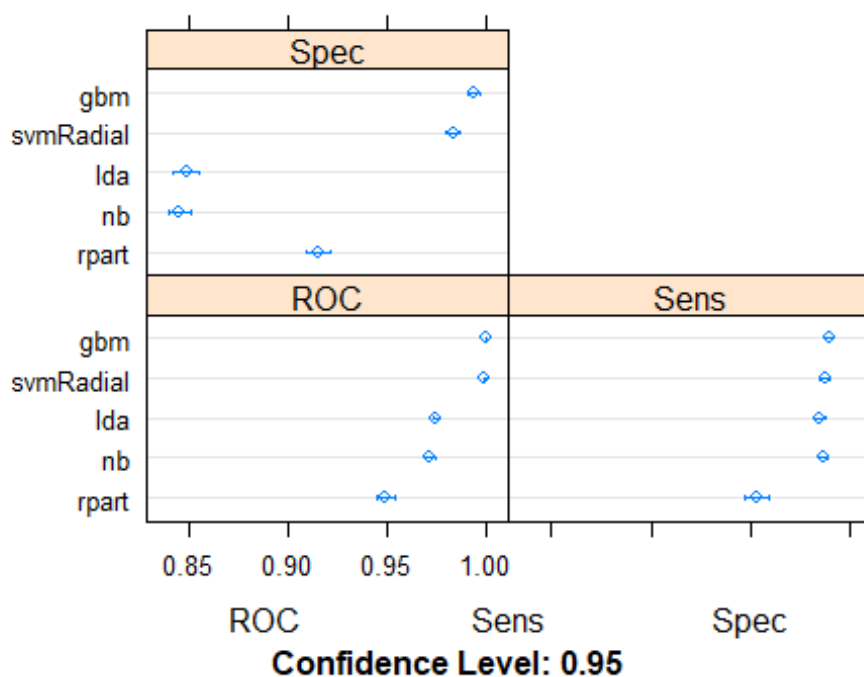
Sens

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
gbm	0.9867900	0.9887715	0.9894285	0.9894965	0.9899273	0.9940555	0
lda	0.9795109	0.9816711	0.9841480	0.9848058	0.9887715	0.9900925	0
svmRadial	0.9841480	0.9849736	0.9864508	0.9875147	0.9894320	0.9947160	0
rpart	0.9431593	0.9464908	0.9527741	0.9534935	0.9597094	0.9663144	0
nb	0.9821664	0.9848009	0.9877807	0.9866561	0.9887696	0.9900925	0

Spec

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
gbm	0.9845222	0.9922652	0.9936070	0.9934733	0.9952917	0.9986541	0
lda	0.8337820	0.8448856	0.8475774	0.8484043	0.8536594	0.8641560	0
svmRadial	0.9751009	0.9801480	0.9838493	0.9827741	0.9850269	0.9905851	0
rpart	0.9004038	0.9108345	0.9165545	0.9152186	0.9209287	0.9266980	0
nb	0.8270525	0.8430611	0.8462315	0.8451753	0.8500084	0.8559892	0

`dotplot(results)`



## Comparing multiple models test results

	lower	ROC	upper
gbm	0.9654861	0.9791287	0.9927712
lda	0.9603631	0.9755420	0.9907208
svmRadial	0.9495295	0.9697633	0.9899971
rpart	0.9262292	0.9481982	0.9701673
nb	0.9561224	0.9719095	0.9876966

While the ROC dropped slightly between training and testing, the gradient boosting model had the highest ROC for both.

## Building an Ensemble model

We can combine the predictions of multiple caret models using the `caretStack()` function in the `caretEnsemble` package. First, we want to check that the models are not highly correlated (correlation > 0.75). We want the correlations low because we want the models to be good at predicting in different ways. To check model correlations, we use the `modelCor()` function.

```
modelCor(results)
```

	gbm	lda	svmRadial	rpart	nb
gbm	1.0000000	0.5099949	0.1303352	0.5709761	0.4941467
lda	0.5099949	1.0000000	0.3128617	0.4788207	0.7647076
svmRadial	0.1303352	0.3128617	1.0000000	0.2383766	0.4294977
rpart	0.5709761	0.4788207	0.2383766	1.0000000	0.3811141
nb	0.4941467	0.7647076	0.4294977	0.3811141	1.0000000

We can see that the lda and nb models slightly exceed the 0.75 correlation threshold, but all of the other correlations are much lower. While we could remove either the lda or nb models, we will leave them both in the ensemble since the correlation was only slightly exceeded.

```
# Model stacking with glm
```

```
stackControl <- trainControl(method="cv", number=10,  
                             summaryFunction=twoClassSummary,  
                             savePredictions=TRUE, classProbs=TRUE,  
                             allowParallel = TRUE)
```

```
set.seed(2020)
```

```
stack.glm <- caretStack(model_list, method="glm", metric="ROC",  
trControl=stackControl)  
print(stack.glm)
```

A glm ensemble of 5 base models: gbm, lda, svmRadial, rpart, nb

Ensemble results:

Generalized Linear Model

30000 samples

5 predictor

```
2 classes: 'Legitimate', 'Fraud'
```

```
No pre-processing
```

```
Resampling: Cross-Validated (10 fold)
```

```
Summary of sample sizes: 27000, 27000, 26999, 27001, 26999, 27000, ...
```

```
Resampling results:
```

ROC	Sens	Spec
0.9994699	0.9929314	0.9947513

Comparing the results above to the training results, we see that the stacked model has a 0.99947 ROC, which is slightly higher than the 0.99929 ROC from the gbm model alone.

## Conclusions

After utilizing multiple sampling methods, it was determined that the `ovun.sample` method produced the best ROC results when applied to the imbalanced training set. The resampled set contains a 50/50 balance and 30,000 rows. Additionally, we found that the gradient boosting model performed better than the other four tested models. The variable importance was determined, which gave insights into the most important predictor variables. Finally, an ensemble model was created and had a slightly higher ROC than the gradient boosting model alone. The five machine learning models tested are a small fraction of the 238 models available in the `caret` package, so while our results are promising, there are most certainly other models or ensembles that could produce better results.

Additionally, we did not explore model tuning, which `caret` can do in a general way by using the `tunelength` parameter, or in fully customizable ways by specifying parameters inside of the `train()` function. The results of this project have shown how machine learning can be used to detect fraud. It is no surprise that some of the largest businesses and banks in the world utilize machine learning for fraud detection.

## Reference

[Official caret package website](#)