Credit Card Fraud Detection using Classification

Team ID: 10
Name: Jordan Brennan ; Exploratory Data Analysis & Visuals, Random Forests
Name: Kideok Kwon ; Modeling, Prediction, Subset Selection, Automation
Name: Yuyan Li ; Background, Performance Metrics, Class Imbalance
Name: Samantha Soendoro ; Presentation Design, Sampling Methods, Final Words

Note: most topics were discussed, brainstormed, and written together. The assignments are based on the emphasis provided by each teammate

1 Introduction

This document is the final report for the course project of Group 10 of STA 141A, Fall 2019.

1.1 Background

In this project, we analyze **Credit Card Transactions** from an anonymous European Cardholder. This dataset contains 284,807 rows, with 31 features, one being the target variable, which is a binary variable that indicates if the transaction is Fraudulent or not. This is the variable we are interested in predicting.

As with many fraud-related data, this dataset is highly imbalanced, as there are much fewer fraudulent transactions than non-fraudulent transactions. More precisely, **out of the 284,807 rows, only 0.17% of them are Fraudulent**. Meaning, there are 485 fraudulent cases, with 284,322 legitimate transactions. Additionally, as this type of data is naturally confidential, the variables have been made anonymous. To make this dataset anonymous, PCA was applied to reduce dimensionality (from an unspecified number of original columns), which also has a bonus effect of hiding column names. (This was done by the data provider)

Table.1 Description of features

Features	**Description**
Time	Number of seconds elapsed between this transaction and the first transaction in the dataset
V1, V2, … , V28	Result of a PCA Dimensionality reduction to protect user identities and sensitive features
Amount	Transaction amount

Class	1 for fraudulent transactions, 0 otherwise

Due to the specified nature of the dataset, analyses must rely fully on statistical and machine learning theory. Domain knowledge on security will not be useful. This is an incredible project as it exposes the users to different sampling techniques and the importance of it, as well as understanding the trade-off between accuracy and computational speed between different machine learning algorithms and its parameters. Alongside that, this project is a great introduction to more modern data science trends in, as a dataset of this size is, without a doubt, more akin to the size of data that might be exposed in the industry rather than the size of the datasets that might be provided in a classroom setting.

1.2 Statistical Questions of Interest

The primary question of interest is how to predict Fraudulent Credit Card Transactions using Classification Algorithms. To achieve this however, due to the type of dataset, many statistical variables must be considered. **First is choosing the correct performance metric for evaluating accuracy.** By using a conventional accuracy metric, any accuracy score would be very misleading. This is due to the fact that the data is highly imbalanced, meaning, even if the model was to "always predict not fraud", it would be considered "99.83% accurate". **Second, is how to handle the problem of data imbalance.** While machine learning algorithms are typically robust for slight imbalance between classes, an imbalance to this degree makes it difficult for most algorithms. **Another consideration is best subset selection.** Due to the lack of context for each predictor, one must use statistical methods to select the best subset of the potential 30 predictors. The methods below will explore different techniques for doing this, such as L1 and L2 penalties applied to Logistic Regression, as well as exploring different subset selection methods. Additionally, there will be a discussion of the caveats of the various various selection methods.

2 Preprocessing

2.1 Identifying the Correct Performance Metric

By using the standard accuracy metric for Classification algorithms, the following confusion matrix reports an average accuracy of **99.93%**. A sample below is produced using a classic Logistic Regression algorithm with no data preprocessing and no tuned parameters.

```
[85293,     6]
[   52,    92]
```

While the total accuracy score seems high, the percentage of fraudulent transactions that were correctly identified is only **63.45%**, while the percentage of legitimate transactions that were correctly identified was **99.99%**. One can notice that, due to the imbalance of the classes, the

total accuracy is heavily misleading, as our goal is to be able to correctly identify fraud, and that was not done efficiently.

The best way to deal with this is instead of aggregating the accuracies, we look at them individually. We can use the Recall Score, which, in the example, is **63.45%**.The Formula for Recall is as follows:

Recall = True Positive/(True Positive + False Negative)

Additionally, it is good to keep in mind not just the % of Fraud that was predicted correctly, but also how good the prediction of the non-fraud was. This will be discussed in the next section.

2.2 Dealing with Class Imbalance

The next step is dealing with the class imbalance. As noted previously, running Logistic Regression on the raw, unsampled data outputs a Recall Score close to **63.45%**, which is unideal. This means that only **63.45%** of Fraud is being detected. Other algorithms such as LDA, and SVM also do not work well with high imbalance. While Tree Methods such as Random Forests are much more robust with class imbalance, a **0.17%** Fraud rate is daunting for any method.

To combat this, it is crucial that some form of Sampling method is applied. Upon different experimentation, we resided on comparing two different methods.

1. **Base Case:** Uses the same imbalance ratio to conduct the analysis
2. **Oversample and Undersample:** Undersamples the Larger Class, and Oversample the lower class to achieve a *ratio of 50:50*.

There are certain caveats that must be noted before conducting the sampling.

For the undersampling procedure, it is crucial to understand that there is potential for important information to be excluded. This is why the two performance metrics that was specified earlier is important. For the undersampling, it must be insured that the score that specifies what percent of non-fraud detected does not dip too far down. As a side note, given these caveats, it is clear that the sampling should be done without replacement.

The oversampling procedure, which is by nature a with replacement sampling technique, also has a potentially dangerous caveat. One must make sure that, when performing oversampling on a dataset, it must be done *after* splitting the data into Train and Test. This is true because the dataset is sampled with replacement. If the sampling is done *before* the split, then it is possible for duplicate samples to appear in both test and training sets, which would theoretically inflate the accuracy.

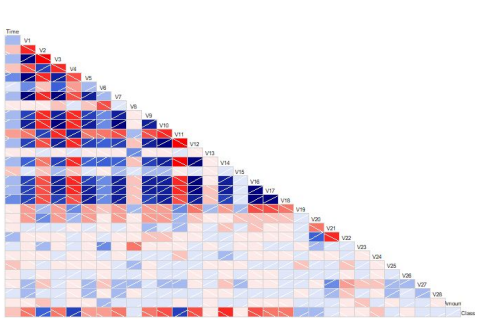
3 Exploratory Data Analysis and Visualization

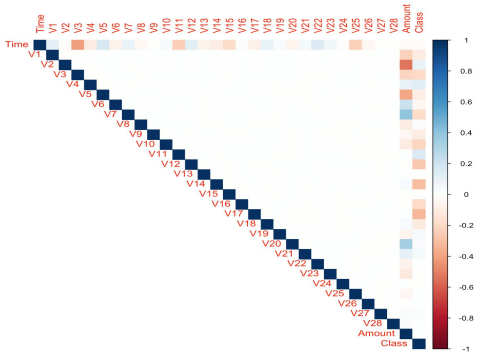
It is helpful to explore the data using various visualization techniques. On a side-note, while standardization of features may be useful before analyzing correlation between different predictors, this was not required in this specific scenario, as the data was run through PCA, which, by convention, is done after standardization.

Additionally, the estimated inferences from the plots will also be tested using more quantitative methods in later sections.

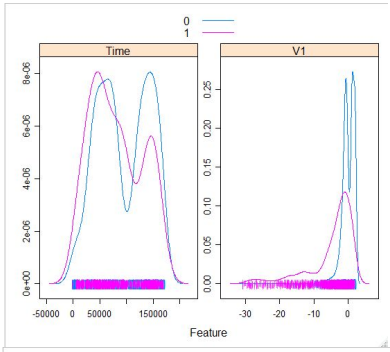
3.1 Distribution Plots of Class for each Potential Predictor

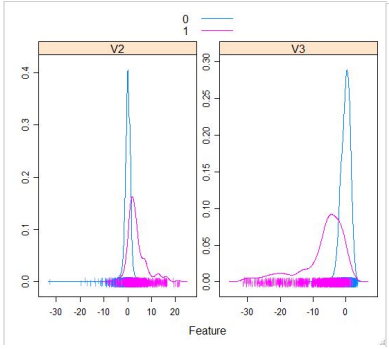
The following are correlation plots.
(Graph 3.1.1)

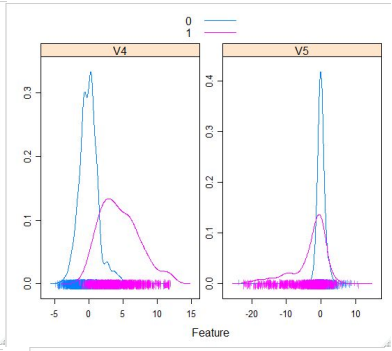


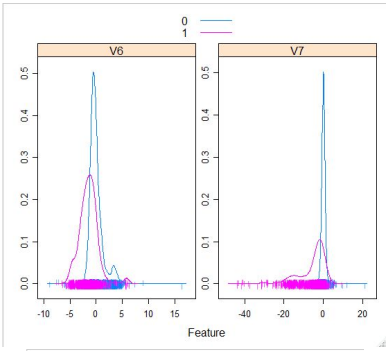


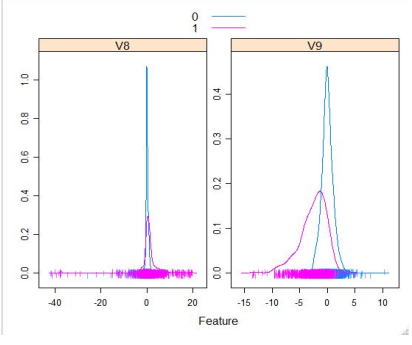
The following are the plots for each predictor mapped with Class. *(Graph 3.1.2)*

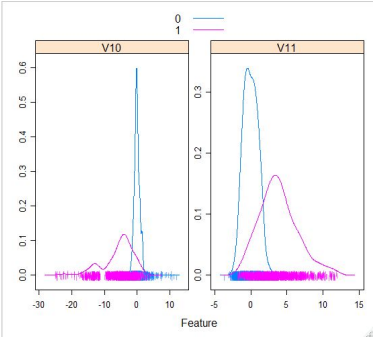


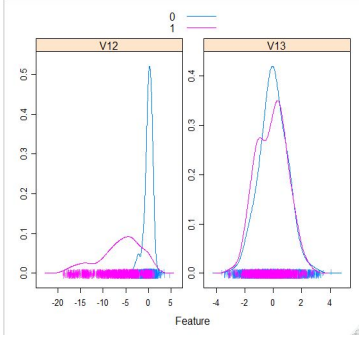


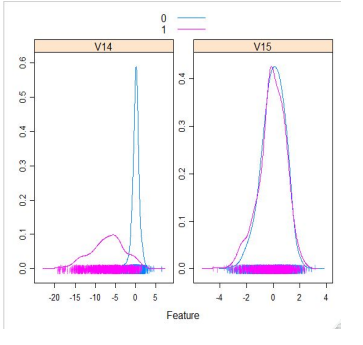












Note: The plot for the other predictors can be visualized by running the R code in the Appendix. Omitted here due to space issues.

Based on the descriptive analysis, we observe that:

1. The correlation plots below can give us an idea of which variables have strong or weak, positive or negative relationships with Class.(See in graph 3.1.1) Here, **V10, V12, V14, V16,** and **V19** have a fairly strong positive relationship with Class and **V2, V4**, and **V11** have a strong, negative relationship with Class. Using featurePlot, we can take a look at what variables in the dataset are less impactful to detecting fraud.
2. The featurePlots are used to analyze which variables are important in predicting whether or not transactions are fraudulent or non-fraudulent. The featurePlots consist of two different lines, the purple line represents Class 1 (fraudulent transaction) and the blue line represents Class 0 (non-fraudulent). (See in graph 3.1.2) Therefore the two classes above have a similar shape and frequency as one another. This therefore suggests that **V15, V20, V22, V24, V25**, and **V26** are less impactful on our target variable, Class.

Note: These estimations will be re-evaluated in the model building process

4 Modeling and Prediction

To evaluate model accuracy score with the basis of context, we can set a **base case** for our accuracy (Using the performance metric we established earlier). We refer to the model specified earlier to establish our base case:

Base Model: *Accuracy Score ~63%*

- All Predictors
- Raw Dataset (no sampling methods)
- Logistic Regression (no penalty)

The following modeling methods are built with the conditions and constraints discussed in the previous sections. It will be in our best interest to attempt to achieve an accuracy stronger than **63%**. To handle the preprocessing efficiently, we wrote various functions to simplify the process. While the preprocessing function can be observed in the Appendix, the following are the procedure that occurs each time the code is run:

1. Split Train and Test set, 70:30 ratio (or else if specified in the parameter)
2. Applies Sampling Method, either a 1 or 2, details specified in earlier sections
3. Outputs the new Data, split between Train and Test

With modeling, the above mentioned function will be wrapped by the functions used by the Algorithm.

4.1 Logistic Regression (with Random Forests)

Logistic Regression is a powerful classification algorithm. It is typically faster than most other classification algorithms, although Tree Methods such as Random Forests seem to often win the race. It also, theoretically speaking, performs better than algorithms such as LDA when the conditions are not met for LDA. As an example, referring to *Introduction to Statistical Learning*, LDA performs better than Logistic Regression when classes are well separated, as well as if the distributions of the classes look normal. As these assumptions feel violated for the most part, Logistic Regression is a clear first consideration. Other algorithms, such as SVM (Support Vector Machines) is infamous for their atrocious computational speed, thus, would not be an initial consideration as our dataset is fairly large.

Alongside Logistic Regression, Random Forests were used as a method for extracting the most significant predictors. Random Forest Feature Extraction is one of the methods we used for modifying the Dataset.

Note: To evaluate the accuracy of a model type, ***each model type was run numerous times*** *and thus the reported accuracy scores are the average of each model type.*

Note 2: From here on out, each model will specify the accuracy of the Fraud, and then followed by the accuracy of the non-Fraud, as discussed earlier.

4.1.1 Applying Logistic Regression (No Penalty) on Balanced Data

The following are Logistic Regression Models ran with no penalty and on both the balanced dataset and base model for comparison.

Model 4.1.1 (Base): *Accuracy Score ~58%, 99.98%*

- All Predictors
- Raw Dataset (no sampling methods)
- Logistic Regression (no penalty)

Model 4.1.1 (Balanced): *Accuracy Score ~90.32%, 97.73%*

- All Predictors
- Balanced Dataset using Oversampling and Undersampling
- Logistic Regression (no penalty)

4.1.2 Applying Logistic Regression (L1 and L2 Penalty) on Balanced Data

The following are Logistic Regression Models ran with L1 and L2 regularization. L1 and L2 are popular methods to - in a sense - , choosing the best predictors. How it actually works is that the least significant predictors are minimized or eliminated, depending on which penalty is used.

Model 4.1.2 (Balanced, L1 Penalty): *Accuracy Score ~90.24%, 97.71%*

- All Predictors
- Balanced Dataset using Oversampling and Undersampling
- Logistic Regression (L1 Penalty)

Model 4.1.2 (Balanced, L2 Penalty): *Accuracy Score ~92.10%, 97.72%*

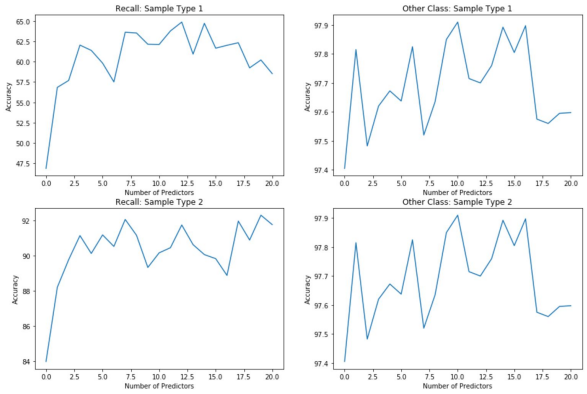
- All Predictors
- Balanced Dataset using Oversampling and Undersampling
- Logistic Regression (L2 Penalty)

*Note: Logistic Regression with L2 Penalty ran **significantly** faster than L1 or base*

4.1.3 Logistic Regression, using Random Forests for Feature Selection on Balanced Data

A popular method for feature selection is using Random Forests to extract the most significant features. According to *Introduction to Statistical Learning*, this is done through comparing tree node impurity along with the number of samples that reach the particular node.

Note: The feature importance can be seen in the Appendix in the Python section



Upon finding the order of feature importance, we can use a loop to find the best number of predictors to include, using Recall Accuracy and use a visual criteria such as the Elbow Method like the KNN, because parsimony is also a factor in this particular model selection. This is visualized on the left.

The right graphs are less significant. Carefully notice that the scales are different. Using the elbow criterion for the left graphs, we can infer that using the **top 3** predictors may create a good model.

Model 4.1.3 (Balanced, RF Feature Selection): *Accuracy Score ~91.14%, 97.71%*

4.2 Other Model Considerations

4.2.1 Other Algorithms

Other algorithms were considered and experimented, and some can be observed in the appendix. However, the best model we were able to produce was Logistic Regression, with extensive parameter tuning.

4.2.2 Caveats with Best Subset Selection

In the textbook, *Introduction to Statistical Learning*, subset selection using forward and backward stepwise selection is emphasized and encouraged. However, multiple sources and forums online seem to not prefer this technique, especially criticizing the criterion used, AIC, AICc, or BIC. Thus, we decided to step away from this approach. In addition, there was an instance where AIC/BIC criterion could have been utilized, however we chose a more computationally expensive method using cross-validation to ensure optimal accuracy.

5 Discussion

The results of our study provide not only an accurate method of predicting credit card detection, but also the general means to do so, using various functions that was written with the group. There is a deep and careful exploration of data preprocessing techniques, and each method that is explored is heavily insured with both benefits and major caveats.

In our findings, we were able to raise our base 63% accuracy to around 92%, using Logistic Regression with L2 Penalty using an Undersampled/Oversampled dataset. This was a challenging project as we encountered many problems not covered in the course, such as dealing with many predictors, high class imbalance, and redefining performance metrics.

Given more time, resources, and knowledge of Machine Learning algorithms, a future consideration is to revisit this project using Isolation Forests and Gradient Boosting. Upon light exposure, it seems that Isolation Forests specialize in anomaly and fraud detection, and Gradient Boosting methods such as XGBoost seems to be at the forefront of Machine Learning, alongside Deep Learning.

6. Appendix

Note: All material from this point on is for the Appendix.

Sources used:

https://www.rdocumentation.org/packages/Seurat/versions/3.1.1/topics/FeaturePlot
https://cran.r-project.org/web/packages/unbalanced/unbalanced.pdf
https://data-flair.training/blogs/data-science-machine-learning-project-credit-card-fraud-detection/

https://stackoverflow.com/questions/14463277/how-to-disable-python-warnings
https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html
https://medium.com/datadriveninvestor/rethinking-the-right-metrics-for-fraud-detection-4edfb629c423
https://stats.stackexchange.com/questions/20836/algorithms-for-automatic-model-selection/20856#20856
https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.subplots.html
https://www.geeksforgeeks.org/confusion-matrix-machine-learning/
http://faculty.marshall.usc.edu/gareth-james/ISL/
https://stackoverflow.com/questions/25427650/sklearn-logisticregression-without-regularization
https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3

Data: https://www.kaggle.com/mlg-ulb/creditcardfraud (https://www.kaggle.com/mlg-ulb/creditcardfraud)

In [105]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [106]:
```
df = pd.read_csv("creditcard.csv")
df.drop('Time',axis=1,inplace=True)
```

In [107]:
```
# Class Distribution
print('Class 1 Length:',len(df[df['Class']==1]))
print('Class 0 Length:',len(df[df['Class']==0]))
print(round((len(df[df['Class']==1])/len(df))*100,2),'% of Data is Fraud')
```

```
Class 1 Length: 492
Class 0 Length: 284315
0.17 % of Data is Fraud
```

Base Case: Logistic Regression with all Predictors

Run on Imbalanced Dataset

Run on Balanced Dataset

In [108]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [109]:
```
def preprocess(df,sample_type,ratio=0.70):
    """
    splits train and test, then resamples data
    Output: X_train, X_test, y_train, y_test
    """
    from sklearn.model_selection import train_test_split
    X = df.drop('Class',axis=1)
    y = df['Class']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    df_train = pd.concat([X_train, y_train], axis=1, sort=False)

    if(sample_type == 1):
        #1 A big % Sample of the raw, entire dataset
        dfS1 = df_train[df_train['Class'] == 1].sample(round(len(df_train[df_tr
        dfS0 = df_train[df_train['Class'] == 0].sample(round(len(df_train[df_tr
        df_newsample = pd.concat([dfS1,dfS0])

    if(sample_type == 2):
        #2 Oversampling Class 1 and Undersample Class 0
        dfS1 = df_train[df_train['Class'] == 1].sample(round(len(df_train[df_tr
        dfS0 = df_train[df_train['Class'] == 0].sample(round(len(df_train[df_tr
        df_newsample = pd.concat([dfS1,dfS0])

    X_train = df_newsample.drop('Class',axis=1)
    y_train = df_newsample['Class']


    return (X_train,X_test,y_train,y_test)
```

In [141]:
```
def run_logreg(data,sample_type):
    """
    1. Takes in data
    2. Runs Logistic Regression
    3. Outputs Recall
    """
    X_train, X_test, y_train, y_test = preprocess(data,sample_type)
    from sklearn.linear_model import LogisticRegression

    logmodel = LogisticRegression(penalty='l2')
    logmodel.fit(X_train,y_train,)

    predictions = logmodel.predict(X_test)

    from sklearn.metrics import classification_report,confusion_matrix
    rep = confusion_matrix(y_test,predictions)
    recall = np.round((rep[1][1]/sum(rep[1]))*100,2)
    precision = np.round((rep[0][0]/sum(rep[0]))*100,2)
    #print(len(X_train),len(X_test),len(y_train),len(y_test))
    #print(recall)
    #print(rep)
    return('recall:', recall, 'non-fraud:',precision)
```

In [142]:
```
average_recall_1 = []
average_recall_2 = []
average_nonfraud_1 = []
average_nonfraud_2 = []
for x in range(10):
    temp1 = run_logreg(df,1)
    temp2 = run_logreg(df,2)
    average_recall_1.append(temp1[1])
    average_nonfraud_1.append(temp1[3])
    average_recall_2.append(temp2[1])
    average_nonfraud_2.append(temp2[3])

print('recall for base data: ',round(np.mean(average_recall_1),2),'non-fraud: '
print('recall for balanced data: ',round(np.mean(average_recall_2),2),'non-frau
```

```
recall for base data:  63.52 non-fraud:  99.98
recall for balanced data:  92.1 non-fraud:  97.72
```

Choosing the Best Predictors

While `Introduction to Statistical Learning` introduces seemingly promising methods for best subset selection, it seems that, according to various sources, this may not be so reliable, based on more modern findings. The algorithm relied on using a fitting criterion such as R^2 or AIC to pick the best combination of variables.

Thus, it may be worth looking at 2 alternative options:

- Built-in Regularization Methods such as Lasso, Ridge, and Elastic-Net.
- Implement a subset selection algorithm, but by using accuracy after using it on train-test-split.

`for next time: look into how regularization is built-in to the scikit-learn regression functions.`

Why Subset Selection isn't ideal?

- https://stats.stackexchange.com/questions/350587/why-is-best-subset-selection-not-favored-in-comparison-to-lasso (https://stats.stackexchange.com/questions/350587/why-is-best-subset-selection-not-favored-in-comparison-to-lasso)
- https://towardsdatascience.com/stopping-stepwise-why-stepwise-selection-is-bad-and-what-you-should-use-instead-90818b3f52df (https://towardsdatascience.com/stopping-stepwise-why-stepwise-selection-is-bad-and-what-you-should-use-instead-90818b3f52df)

How to select best predictors

- https://chrisalbon.com/machine_learning/trees_and_forests/feature_selection_using_random_fo (https://chrisalbon.com/machine_learning/trees_and_forests/feature_selection_using_random_f

In [112]:
```
# Write program that picks best predictors
```

In [113]:
```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
```

In [114]:
```
X_train, X_test, y_train, y_test = preprocess(df,2)
```

In [115]:
```
#Create list of column names
feat_labels = X_train.columns.tolist()
```

In [116]:
```
# Create a random forest classifier
clf = RandomForestClassifier()

# Train the classifier
clf.fit(X_train, y_train)

importance_list = []
# Print the name and gini importance of each feature
for feature in zip(feat_labels, clf.feature_importances_):
    importance_list.append(feature)
```

In [117]:
```
importance_list.sort()
```

In [118]:
```
sorted(importance_list, key=lambda x: x[1],reverse=True)
```

Out[118]:
```
[('V14', 0.21813373430775512),
 ('V12', 0.1417328810969784),
 ('V4', 0.10690809274177429),
 ('V3', 0.10423131665845961),
 ('V16', 0.061933994054364325),
 ('V7', 0.05858339014163422),
 ('V9', 0.04555370778227845),
 ('V10', 0.03978991708809655),
 ('V17', 0.036704067611840155),
 ('V21', 0.016172851678543092),
 ('V8', 0.012603643998747489),
 ('V13', 0.01223355928437633),
 ('V18', 0.011610083441147816),
 ('Amount', 0.011609669684307319),
 ('V1', 0.011518393922091776),
 ('V5', 0.010117549074782002),
 ('V20', 0.010052283332702729),
 ('V11', 0.009902279562967051),
 ('V26', 0.009591891580903882),
 ('V6', 0.009387382846320347),
 ('V19', 0.008256617175597092),
 ('V15', 0.008140507578531153),
 ('V28', 0.007103920614908966),
 ('V23', 0.007020358824875228),
 ('V27', 0.006774048736845699),
 ('V22', 0.00671512395356999),
 ('V2', 0.006302171675422795),
 ('V25', 0.006019434893518539),
 ('V24', 0.005297126656659555)]
```

In [119]:
```
# Modify Base DataFrame?
dfA = df[['Class','V12']]

print(run_logreg(dfA,1))
print(run_logreg(dfA,2))
```

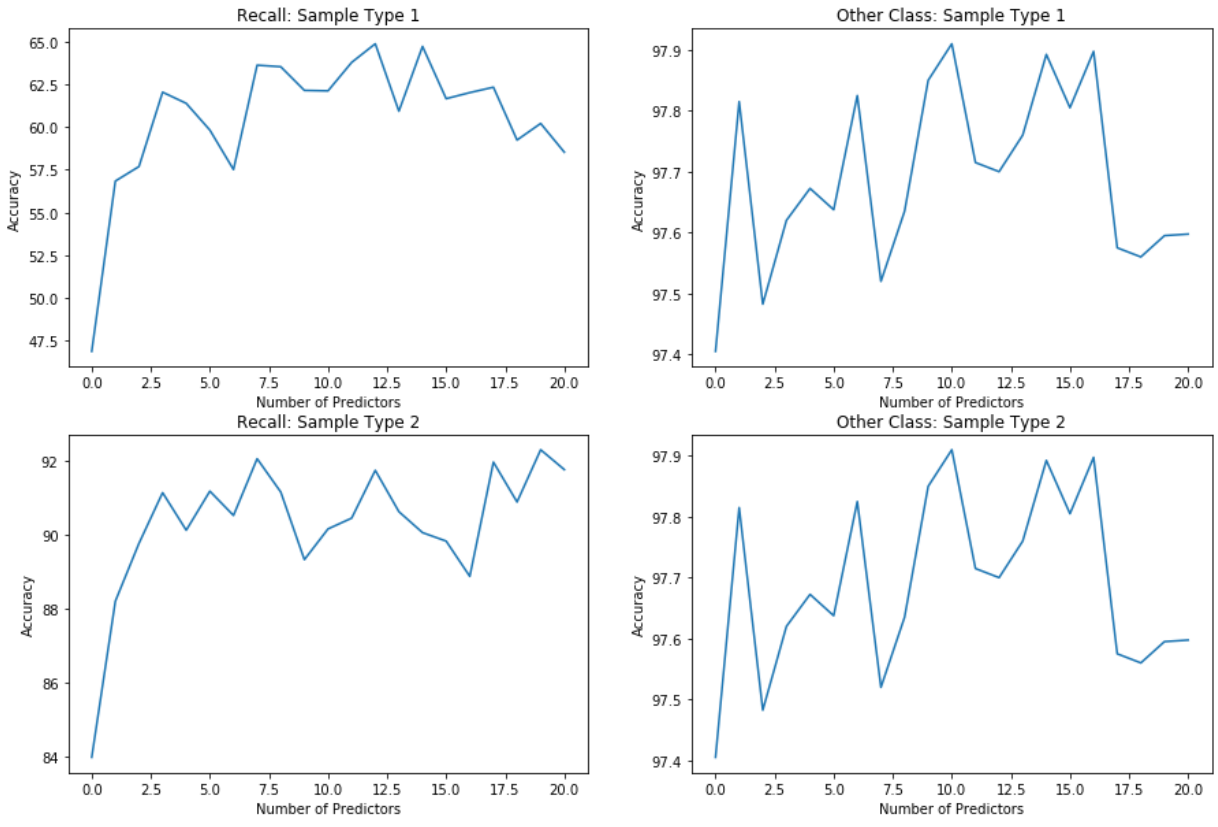
```
('recall:', 43.83, 'precision:', 99.98)
('recall:', 85.81, 'precision:', 93.61)
```

In [120]:
```
# Create a function that figures out the best number of predictors iteratively.
columns = ['Class','V14','V10','V4','V12','V17','V3','V21','V16','V7','V19','V1
```

In [121]:
```
recall_1 = []
precision_1 = []
recall_2 = []
precision_2 = []
# Run function 29x10 times to figure out best predictor
for x in range(2,len(columns)-7):
    # Define columns
    cols = columns[:x]
    df_temp = df[cols]
    tempRecall_1 = []
    tempPrecision_1 = []
    tempRecall_2 = []
    tempPrecision_2 = []
    for x in range(4):
        #Output for Type 1 Sampling Method
        output1 = run_logreg(df_temp,1)
        tempRecall_1.append(output1[1])
        tempPrecision_1.append(output1[3])

        #Output for Type 2 Sampling Method
        output2 = run_logreg(df_temp,2)
        tempRecall_2.append(output2[1])
        tempPrecision_2.append(output2[3])

    #Append the average of the 10 for each variable to the respective lists. To
    recall_1.append(np.mean(tempRecall_1))
    precision_1.append(np.mean(tempPrecision_1))
    recall_2.append(np.mean(tempRecall_2))
    precision_2.append(np.mean(tempPrecision_2))
```

In [131]:
```
plt.figure(figsize=(15,10))

plt.subplot(2,2,1)
plt.plot(recall_1)
plt.title('Recall: Sample Type 1')
plt.ylabel('Accuracy')
plt.xlabel('Number of Predictors')

plt.subplot(2,2,2)
plt.plot(precision_2)
plt.title('Other Class: Sample Type 1')
plt.ylabel('Accuracy')
plt.xlabel('Number of Predictors')

plt.subplot(2,2,3)
plt.plot(recall_2)
plt.title('Recall: Sample Type 2')
plt.ylabel('Accuracy')
plt.xlabel('Number of Predictors')

plt.subplot(2,2,4)
plt.plot(precision_2)
plt.title('Other Class: Sample Type 2')
plt.ylabel('Accuracy')
plt.xlabel('Number of Predictors')
```

Out[131]:
```
Text(0.5, 0, 'Number of Predictors')
```



In [123]:
```
round(recall_1[3],2)
round(recall_2[3],2)
```

Out[123]: 91.14

In [124]:
```
recall_2
```

Out[124]:
```
[83.965,
 88.195,
 89.7625,
 91.135,
 90.1175,
 91.175,
 90.51750000000001,
 92.05499999999999,
 91.15249999999999,
 89.32000000000001,
 90.1525,
 90.44500000000001,
 91.74000000000001,
 90.62,
 90.05250000000001,
 89.82249999999999,
 88.8675,
 91.96249999999999,
 90.8825,
 92.29750000000001,
 91.76]
```

In [127]:
```
def temp(data,sample_type,Cost=1000000000000):
    """
    1. Takes in data
    2. Runs Logistic Regression
    3. Outputs Recall
    """
    X_train, X_test, y_train, y_test = preprocess(data,sample_type)
    from sklearn.linear_model import LogisticRegression

    logmodel = LogisticRegression(C=Cost)
    logmodel.fit(X_train,y_train,)

    predictions = logmodel.predict(X_test)

    from sklearn.metrics import classification_report,confusion_matrix
    rep = confusion_matrix(y_test,predictions)
    recall = np.round((rep[1][1]/sum(rep[1]))*100,2)
    precision = np.round((rep[0][0]/sum(rep[0]))*100,2)
    #print(len(X_train),len(X_test),len(y_train),len(y_test))
    #print(recall)
    #print(rep)
    #return('recall:', recall, 'precision:',precision)
    return rep
```

In [130]:
```
temp(df,1)
```

Out[130]:
```
array([[85293,     6],
       [   52,    92]], dtype=int64)
```

```
cc <- read.csv("~/Downloads/creditcard.csv")

#sampling: 70% of each class
library(class)
n1=length(which(cc$Class==0))*0.7
n2=length(which(cc$Class==1))*0.7
idx_test=c(sample(which(cc$Class==0), n1, replace=FALSE),
           sample(which(cc$Class==1), n2, replace=FALSE))
cc_test = cc[idx_test,]
cc_train = cc[-idx_test,]

#knn all predictors
knn_pred = knn(
  train = cc_train,
  test = cc_test,
  cl = cc_train$Class,
  k = 5)
knn_con = table(true = cc_test$Class, model = knn_pred)
knn_con
cc.pred.knn_error = (knn_con[1,2] + knn_con[2,1])/sum(knn_con)

#knn 2 variables balanced sampling
#knn using only the best 10 predictors
list_best_predictor = c(12,14,10,4,17,11,3,19,8,5)
list_best=sort(list_best_predictor)
library(class)
for(i in 1:10){
  for(j in 1:10){
    if(j>i) {
      knn_pred = knn(
      train = cc_train[,c(list_best[i],list_best[j])],
      test = cc_test[,c(list_best[i],list_best[j])],
      cl = cc_train$Class,
      k = 5)
      knn_con = table(true = cc_test$Class, model = knn_pred)
      knn_con
      cc.pred.knn_error = (knn_con[1,2] + knn_con[2,1])/sum(knn_con)
      print(c(round(list_best[i],1),round(list_best[j],1),cc.pred.knn_error))
      cc.pred.knn_error[i] = cc.pred.knn_error
    }}}
cc.pred.knn_error

#knn 3 variables balanced sampling
```

```
library(class)
for(i in 1:10){
  for(j in 1:10){
    for(x in 1:10){
    if(j>i & x>j) {
      knn_pred = knn(
        train = cc_train[,c(list_best[i], list_best[j], list_best[x])],
        test = cc_test[,c(list_best[i], list_best[j], list_best[x])],
        cl = cc_train$Class,
        k = 5)
      knn_con = table(true = cc_test$Class, model = knn_pred)
      knn_con
      cc.pred.knn_error = (knn_con[1,2] + knn_con[2,1])/sum(knn_con)
      print(c(list_best[i],list_best[j],list_best[x], cc.pred.knn_error))
    }}}}
```

```
my_data <- read.table(file = "clipboard",
                      sep = "\t", header=TRUE)
attach(my_data)
summary(my_data)
str(my_data)

library(caret)
table(my_data$Class)
prop.table(table(my_data$Class))

#Logistic regression before splitting

glm.model=glm(Class~.,my_data,family=binomial())
summary(glm.model)

glm.model.pred.prob = predict(glm.model, my_data, type = "response")
# Convert predictions to class lables (1 or 2, for category 1 or 2, respectively).
glm.model.pred = (glm.model.pred.prob > 0.5) + 1
# Create the confusion matrix by tabulating true classes against predicted classes.
glm.model.conf = table(true = my_data$Class, predicted = glm.model.pred)
glm.model.conf

# Precision: tp/(tp+fp):
glm.model_prec = glm.model.conf[1,1]/sum(glm.model.conf[1,1:2])
glm.model_prec
# Recall: tp/(tp + fn):
glm.model_recall= glm.model.conf[1,1]/sum(glm.model.conf[1:2,1])
glm.model_recall
# F1 score F1 Score might be a better measure to use if
# there is an uneven class distribution (large number of Actual Negatives).
glm.model_F1score = (2*glm.model_recall*glm.model_prec) /
sum(glm.model_recall,glm.model_prec)
glm.model_F1score

#Random Oversampling of the data (ROS)

n_legit <- 284807
new_frac_legit <- 0.50
new_n_total <- n_legit/new_frac_legit # = 284807/0.50 = 569614
library(ROSE)
oversampling_result <- ovun.sample(Class ~ .,
                                   data = my_data,
                                   method = "over"
```

```
                          ,
                          N = new_n_total,
                          seed = 2018)
oversampled_credit <- oversampling_result$data
table(oversampled_credit$Class)

barplot(table(oversampled_credit$Class), col = 4)

#Random Undersampling of the data

n_fraud <- 492
new_frac_fraud <- 0.50
new_n_total1 <- n_fraud/new_frac_fraud # = 492/0.50 = 984
undersampling_result <- ovun.sample(Class ~ .,
                          data = my_data,
                          method = "under"
                          ,
                          N = new_n_total1,
                          seed = 2018)
undersampled_credit <- undersampling_result$data
table(undersampled_credit$Class)

barplot(table(undersampled_credit$Class), col = 4)

#Undersampling and Oversampling Combination of the data


n_new <- nrow(my_data) # = 24600
fraction_fraud_new <- 0.50
sampling_result <- ovun.sample(Class ~ .,
                        data = my_data,
                        method = "both"
                        ,
                        N = n_new,
                        p = fraction_fraud_new,
                        seed = 2018)
sampled_credit <- sampling_result$data
table(sampled_credit$Class)

barplot(table(sampled_credit$Class), col = 4)

#Now let's look at the compare the three methods
```

```
prop.table(table(oversampled_credit$Class))
prop.table(table(undersampled_credit$Class))
prop.table(table(sampled_credit$Class))


#We will choose the combination of under/oversampling
#Next, split data into training and testing groups

require(caTools)
set.seed(101)
sample = sample.split(my_data$Class, SplitRatio = .75)
train = subset(my_data, sample == TRUE)
test  = subset(my_data, sample == FALSE)


#Note the unbalance between classes
prop.table(table(train$Class))
prop.table(table(test$Class))


#Here we use ubSMote resampling and splitting using unbalanced package from R
#In order to produce featurePlot

balanced <- ubSMOTE(X = my_data[,-31], Y = as.factor(my_data$Class),
                    perc.over=200, perc.under=800,  verbose=TRUE )

balanceddf <- cbind(balanced$X, Class = balanced$Y)

for (i in seq(from =1, to = 30, by = 2))
{
  show(
    featurePlot(
      x = balanceddf[, c(i,i+1)],
      y = balanceddf$Class,plot = "density",
      scales = list(x = list(relation="free"),
                    y = list(relation="free")),
      adjust = 1.5, pch = "|", layout = c(2,1 ), auto.key=TRUE
    )
  )
}


#Run logistic regression on the newbalanced data without those variables
```

```
log.reg.glm=glm(Class~.,test,family=binomial())
summary(log.reg)

log.reg.glm.pred.prob = predict(log.reg.glm, test, type = "response")
# Convert predictions to class lables (1 or 2, for category 1 or 2, respectively).
log.reg.glm.pred = (log.reg.glm.pred.prob > 0.5) + 1
# Create the confusion matrix by tabulating true classes against predicted classes.
log.reg.glm.conf = table(true = test$Class, predicted = log.reg.glm.pred)
log.reg.glm.conf
# Precision: tp/(tp+fp):
log.reg.glm_prec = log.reg.glm.conf[1,1]/sum(log.reg.glm.conf[1,1:2])
log.reg.glm_prec
# Recall: tp/(tp + fn):
log.reg.glm_recall= log.reg.glm.conf[1,1]/sum(log.reg.glm.conf[1:2,1])
log.reg.glm_recall
# F1 score F1 Score might be a better measure to use if
# there is an uneven class distribution (large number of Actual Negatives).
log.reg_F1score = (2*log.reg.glm_recall*log.reg.glm_prec) /
sum(log.reg.glm_recall,log.reg.glm_prec)
log.reg_F1score



library(rpart)
tree1 = rpart(Class ~ ., data = train)

library(partykit)
plot(as.party(tree1))

threshold <- 0.5
predicted_classes <- predict(tree1, test, type = "vector") >= threshold

# Confusion matrix & recall
library(caret)
conf.tree = table(data = predicted_classes, reference = test$Class)
conf.tree

#Recall
conf.tree_rec = conf.tree[2,2]/sum(conf.tree[2,1:2])
conf.tree_rec
# Precision: tp/(tp+fp):
conf.tree_prec = conf.tree[1,1]/sum(conf.tree[1,1:2])
```

```
conf.tree_prec
# Recall: tp/(tp + fn):
conf.tree_recall= conf.tree[1,1]/sum(conf.tree[1:2,1])
conf.tree_recall
# F1 score F1 Score might be a better measure to use if
# there is an uneven class distribution (large number of Actual Negatives).
conf.treeF1score = (2*conf.tree_recall*conf.tree_prec) / sum(conf.tree_prec,conf.tree_recall)
conf.treeF1score
```