Final Project Report

Model Design and Web Application Development

of

Credit Card Frauds Detection

INFO 7390

Advanced Data Science & Architecture

Professor:

Srikanth Krishnamurthy

Students:

Team 2

Chenlian Xu

Qianli Ma

Date:

Apr 27th , 2018

Table of Contents

1. EDA for Credit card Fraud 3
1.1 Category Attributes 3
1.2 Category Distribution 6
1.3 Pair Plot 6
2. Precision-Recall Trade-off 9
3. Benchmark 10
4. Feature Engineering 11
5. Feature Selection 12
5.1 Logistic Regression: 13
5.2 SVM: 13
5.3 Random Forest: 13
5.4 GBT: 13
6. Model Selection 15
6.1 Logistic Regression 15
6.2 SVM 16
6.3 Random Forest 17
6.4 GBT 19
6.5 Optimal Model 21
7. Model Deployment 23
8. Web Application Development 24
8.1 Relational Database 24
8.2 Application Development 25

1. EDA for Credit card Fraud

https://github.com/MarcusNEU/INFO7390_2018Spring/blob/master/FinalProject/Model%20Design/eda.ipynb

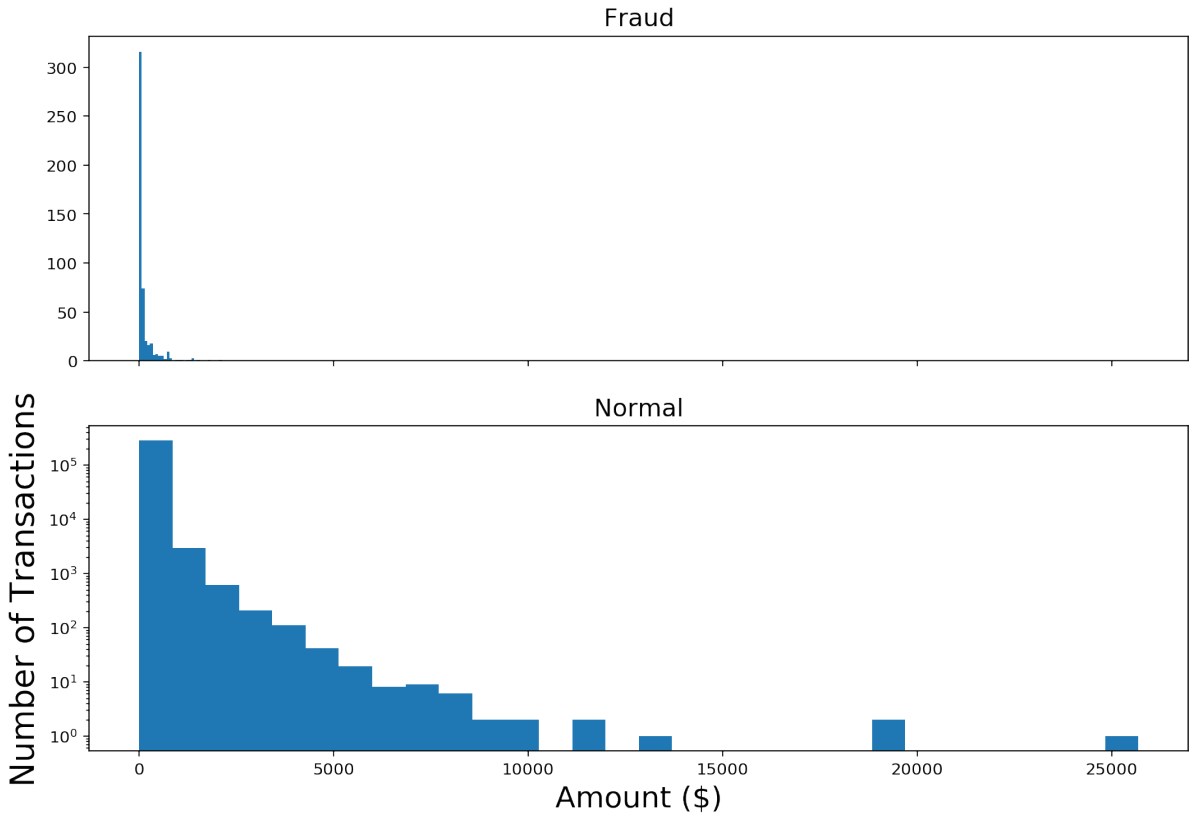
The credit card data set comes from the European credit card transaction data during two days in September 2013. In the total number of 284,807 times of transactions, 492 cases of fraud were discovered, which means the data set is extremely unbalanced, and fraud frequency only accounts for 0.172% of the total transactions.

This data set has 31 columns. V1, V2,, ... ,V28 are the main features which are processed by PCA because of its sensibility, 'Time' is the time in seconds between each transaction and the first transaction. 'Amount' represents the transaction amount. 'Class' represents the response variable, 1 means fraud, and 0 means normal.

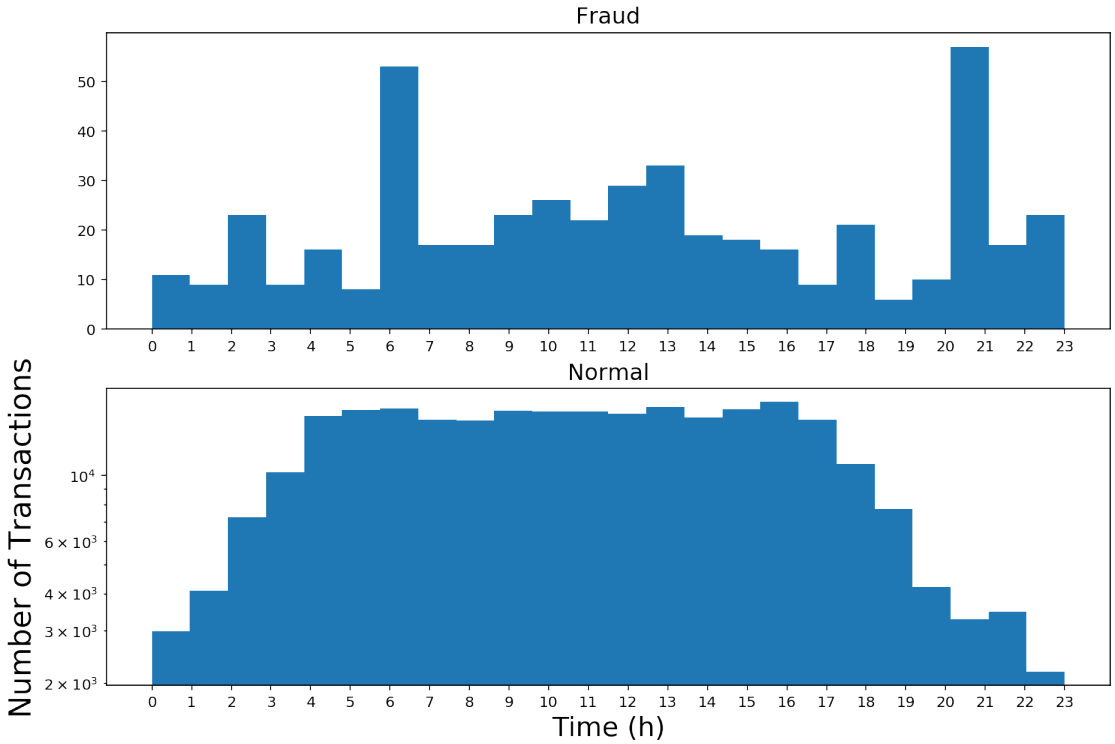
1.1 Category Attributes

```
Fraud
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

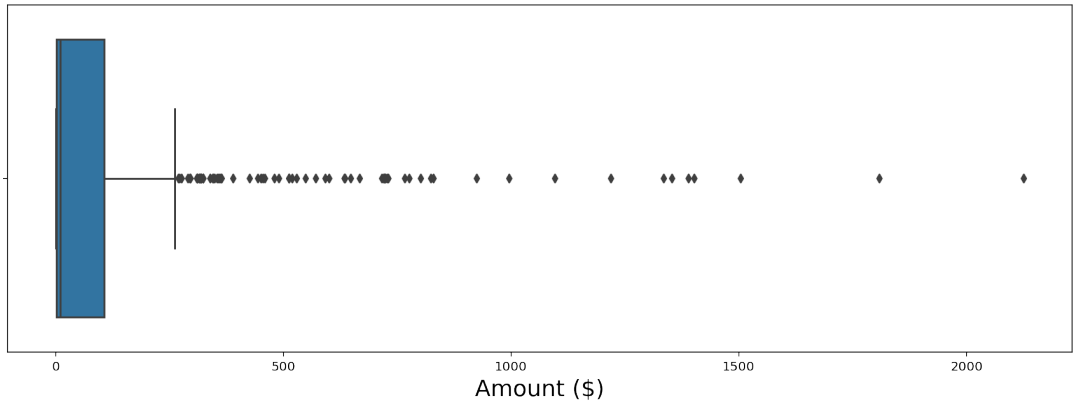
```
Normal
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```



The picture shows that Fraud activity usually happens in the spot where the transaction amount is small, 50% of frauds' amount is less than 9.25 while 50% of normal transactions' amount is 22. And the max amount value of fraud class is 2125.87, but the max value of amount of normal class 25691.16.



The picture illustrate most normal transaction occurs between 5 and 17. But fraud activity most possibly occurs 6 and 20.

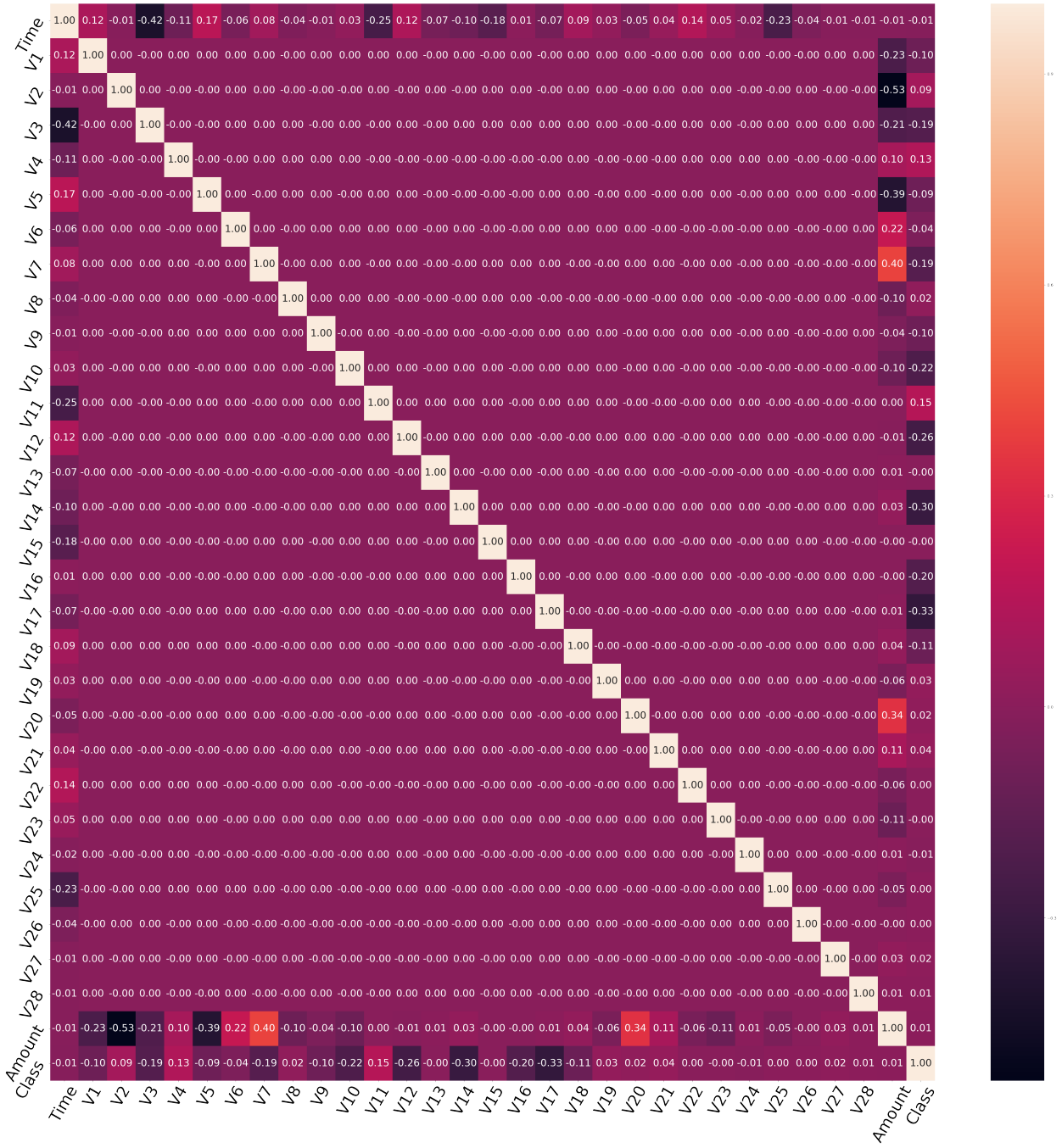


1.2 Category Distribution

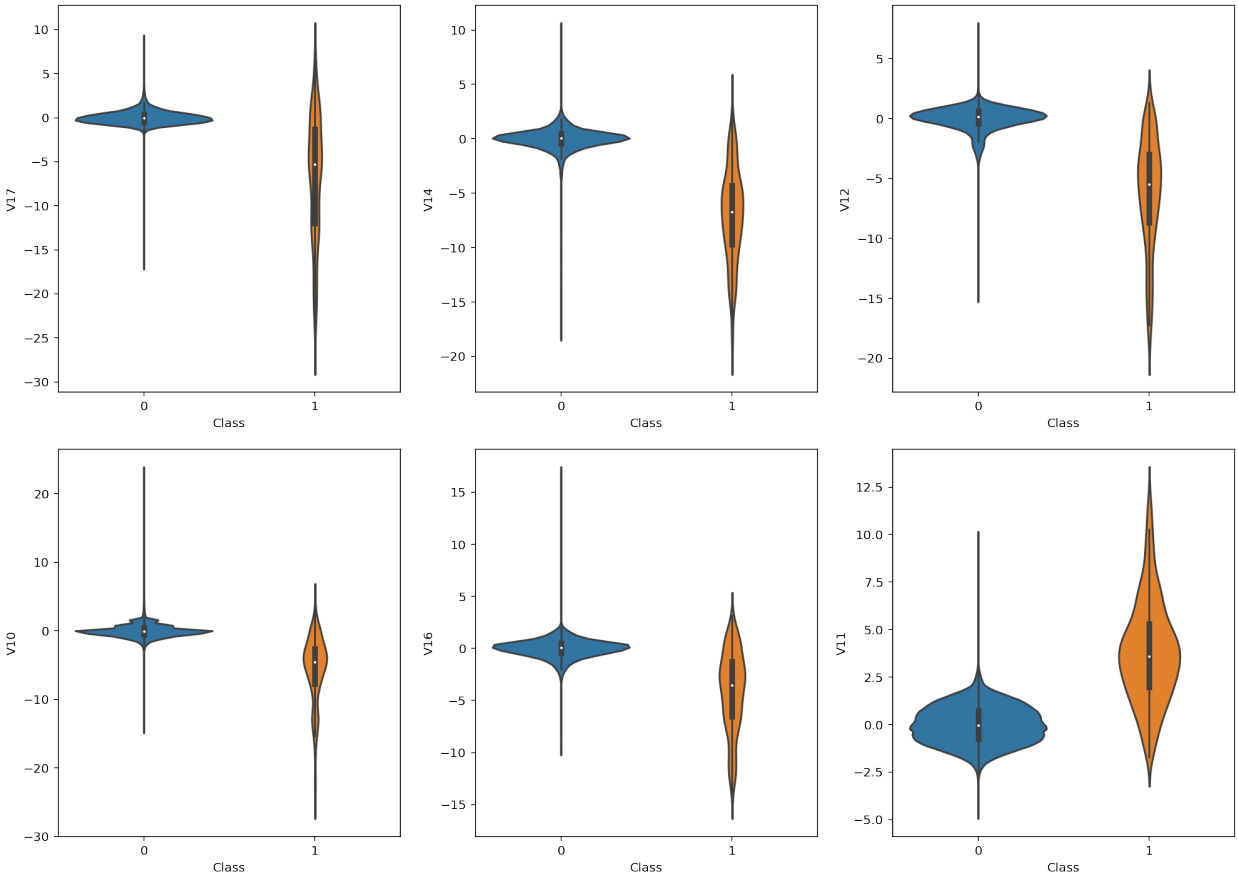
The picture shows that the data set vary unbalanced, fraud class just account for 0.172% of the total transactions which hints that resample process is needed when choose train data.

1.3 Pair Plot

Following picture shows the pair-wise relationships between the columns. columns have high correlation with class was listed, V17, V14, V12, V10, V16 has a negative relationship with class column while V11 has positive relationship with class. We explore the distribution of these columns in different label.


Time V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 Amount Class

Number of features	corelation
V17	-0.33
V14	-0.30
V12	-0.26
V10	-0.22
V16	-0.20
V11	0.15



The picture shows the density of the value. The columns we explored has a skewed distribution when the fraud activity happen and has a relative normal distribution.

2. Precision-Recall Trade-off

To get a trade-off of precision and recall rate, we need to define a method to measure the overall performance of a model with a certain precision and recall rate. In this project, we simply use the average amount of normal and fraud transactions to represent the weights of precision and recall.

```
loss = (1 - precision) * 88.29 + (1 - recall) * 122.12
```

We tried total loss of FN and FP cases like loss = FN*122.12 + FP * 88.29 or divided by (FN+FP) which equals to the average loss of misclassification cases. These two are better than the former as they have clearer definitions and mathematical meanings. Actually, the current loss function we used is an approximate for average loss of frauds and misclassification transactions. It will get an approximate optimal solve under a smaller undersampling proportion compared with the other two loss functions. Thus, the one we used is the optimal one for calculation and research in a limited time.

3. Benchmark

https://github.com/MarcusNEU/INFO7390_2018Spring/blob/master/FinalProject/Model%20Design/benchmark.ipynb

In this part, two sets of model will be trained:

(1) Data set without resampling: all the models will be trained on 75% of the original data set and test on the other 25%.

The result is shown as follows:

Models	Precision	Recall	F1	Accuracy	Custom Loss
Logistic Regression	0.81	0.60	0.69	0.99	65.62
Support Vector Machine	1.00	0.03	0.05	0.99	118.46
***Random Forest**	0.94	0.75	0.83	0.99	35.83
Gradient Boosting Tree	0.80	0.47	0.60	0.99	82.38

(2) Data set with resampling: all the models will be trained on 75% of the resampling data set and test on the whole data set:

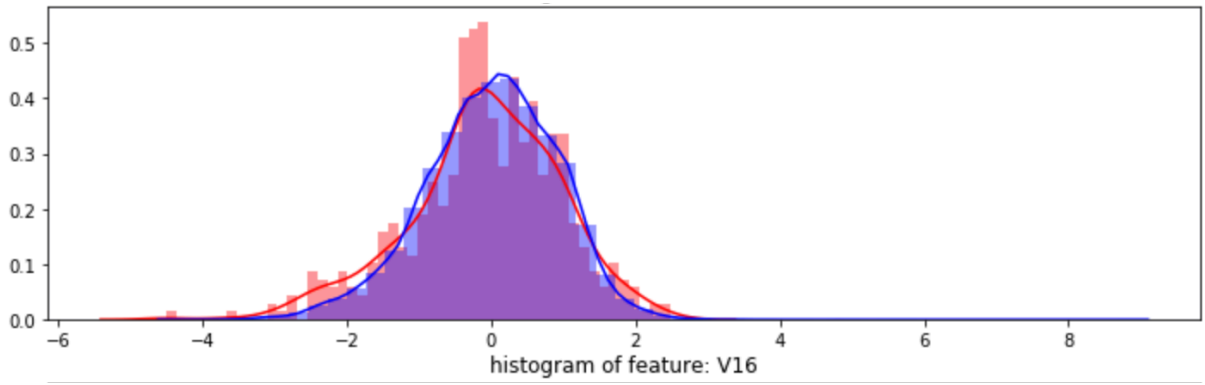
```
for i in range(120,200):
    undersample_data = undersample(normal_indices,fraud_indices,i)
    X_undersample = undersample_data.iloc[:, undersample_data.columns != "Class"]
    y_undersample = undersample_data.iloc[:, undersample_data.columns == "Class"]
    X_undersample_train, X_undersample_test, y_undersample_train, y_undersample_test = train_test_split(X_undersample,
    X_test = df.iloc[:, df.columns != "Class"]
    y_test = df.iloc[:, df.columns == "Class"]
    lr = LogisticRegression(random_state=0)
    loss.append(custom_loss_fuction(lr, X_undersample_train, X_test, y_undersample_train, y_test))
    proportion.append(i)
```

As the data set is very unbalanced, we need to resample the data set. Undersample will be used in the project since the positive class is very less. Different proportion of undersampling (negative/positive) will give different train set. Thus, models' performances will be affected by the undersampling proportion. The optimal undersampling proportions for all the 4 models will be found out by iteration. Thus, after all the iteration, we will get the optimal proportions together with the min loss value of each model. The result will not be shown here since it will be using for comparison with the result after feature engineering and will not be very meaningful at this stage.

4. Feature Engineering

https://github.com/MarcusNEU/INFO7390_2018Spring/blob/master/FinalProject/Model%20Design/feature_engineering%26prediction_algorithms.ipynb

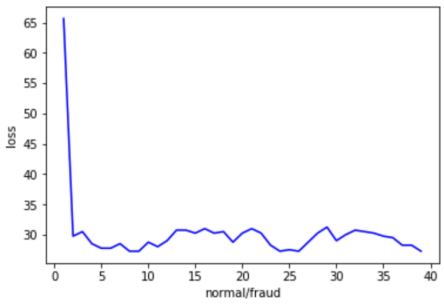
By looking into the differences between normal and fraud transactions, we get some insights of how to do feature engineering on anonymous features. Since this is a classification problem, two classes of the same feature share a very similar distribution would not do much to the model learning.



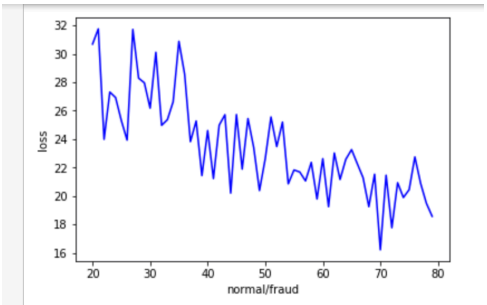
We simply drop all of the features that have very similar distributions between the two types of transactions.

```
df = df.drop(['V8','V13','V15','V20','V22','V23','V24','V25','V26','V27','V28'], axis =1)
```

Then the same things as we did for benchmark: iteration for optimal undersampling proportions. Finally, we get results of optimal proportion and min loss of each model.




```
optimal proportion of normal/fraud is :  8
min loss is :  27.303252032520334
```



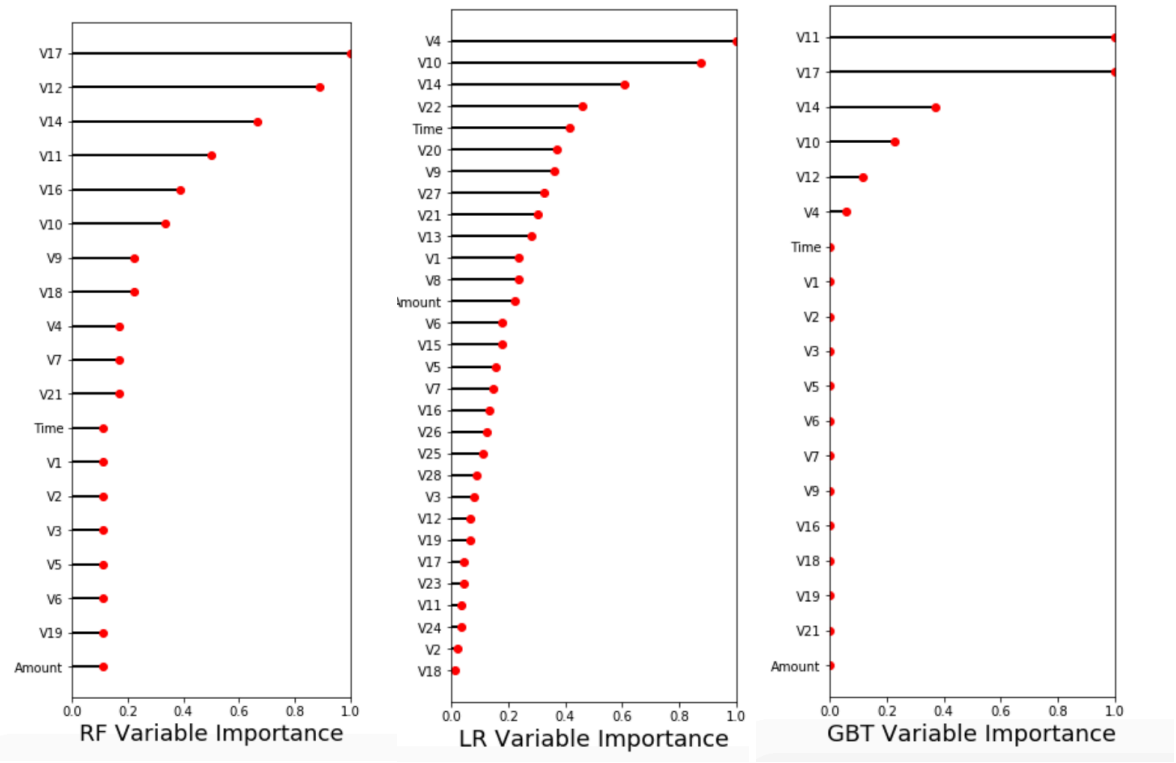

```
optimal proportion of normal/fraud is :  70
min loss is :  16.207662072717387
```

We compare the performance of benchmark and the models after feature engineering. Then we pass the better one to do feature selection.

5. Feature Selection

https://github.com/MarcusNEU/INFO7390_2018Spring/blob/master/FinalProject/Model%20Design/feature_selection.ipynb

n this part, all the model will be trained on the corresponding optimal undersampling proportion calculated from the former parts. we simply use feature importance to rank the features and select top X features for all the models.



Since non-linear SVM is a black box classifier for which we do not know the mapping function Φ explicitly. There will be not any further pre-processing on SVM. The benchmark reaches the min loss at exactly the same proportion (=8) as the one after feature engineering. Thus, we will be using the one after feature engineering for further discussion.

The pre-processing part is over by now after benchmark, feature engineering and feature selection. Here is the result of the whole pre-processing part:

5.1 Logistic Regression:

Features	Precision	Recall	F1	Accuracy
19 selected by fe	0.54	0.54	0.54	0.99
Top 21	0.34	0.66	0.47	0.99
All*	0.60	0.51	0.55	0.99

Optimal proportion: 178

5.2 SVM:

Features	Precision	Recall	F1	Accuracy
19 selected by fe*	1.00	0.77	0.54	0.99
All	1.00	0.77	0.47	0.99

Optimal proportion: 8

5.3 Random Forest:

Features	Precision	Recall	F1 Score	Accuracy
19 picked by fe*	0.82	0.96	0.88	0.99
Top 11	0.78	0.96	0.86	0.99
Top 6	0.79	0.96	0.87	0.99
All features	0.81	0.96	0.88	0.99

Optimal proportion: 70

5.4 GBT:

Features	Precision	Recal	F1 Score	Accuracy	Custom Loss
19 picked by fe	0.74	0.88	0.81	0.99	37.61
Top 6 after fe*	0.68	0.93	0.79	0.99	36.80

Optimal Proportion: 116

Here are the optimal models we got after pre-processing. They will be further discussed in the next part:

Models	Precision	Recall	F1	Accuracy	Custom Loss	Diff vs 1st table
Logistic Regression	0.60	0.51	0.55	0.99	95.15	29.53
Support Vector Machine	1.00	0.77	0.87	0.99	28.09	-90.37
***Random Forest**	0.82	0.96	0.88	0.99	20.78	-15.05
Gradient Boosting Tree	0.68	0.93	0.79	0.99	36.80	-45.58

3 models perform better after selecting an appropriate proportion for undersampling especially for SVM and GBT. Random Forest is the best among the 4 models. Models in this part and the next part are trained on the undersampled dataset and tested on the whole data set. Thus, the performances here might be worse than the 1st part.

6. Model Selection

https://github.com/MarcusNEU/INFO7390_2018Spring/blob/master/FinalProject/Model%20Design/model_selection.ipynb

This is the final part of the actual model design process. We will be tuning hyperparameters for all the current optimal models we got from the former parts. Grid search methods are defined here to implement the tuning.

6.1 Logistic Regression

Tuning C and penalty:

```
def Kfold_tuning_lr(X_train_data, y_train_data):
    fold = KFold(n_splits=5, shuffle=False, random_state=0)

    c_param_range = [0.01, 0.1, 1, 10, 100]
    penalties = ['l1', 'l2']
```

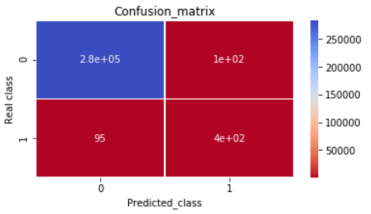
```
   Penalty  C_parameter  Mean loss
0       l1         0.01  53.480438
1       l1         0.10  45.782357
2       l1         1.00  45.118195
3       l1        10.00  44.948828
4       l1       100.00  44.948349
5       l2         0.01  64.449536
6       l2         0.10  58.179545
7       l2         1.00  62.394229
8       l2        10.00  60.905203
9       l2       100.00  60.931849

*********************************************************************************
Best model to choose from cross validation is with Penalty =  l1 and best c =  100.0
*********************************************************************************
```

Since C=10 and C=100 have very similar results, we will simply use C=10.

The optimal logistic regression model is:

```
print "the model classification for 132 proportion"
optimal_lr = LogisticRegression(C=10, penalty='l1', random_state=0)
```

```
the model classification for 132 proportion
the recall for this model is : 0.806910569105691
The accuracy is : 0.9993012812185094
TP 397
TN 284211
FP 104
FN 95
```




```
Classification Report:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00    284315
          1       0.79      0.81      0.80       492

avg / total       1.00      1.00      1.00    284807

The loss is :  41.9077459714717
```

6.2 SVM

Tuning C and gamma:

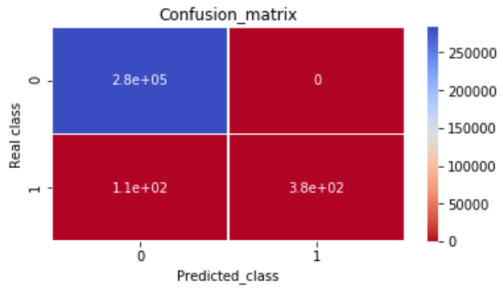
```
def Kfold_tuning_svm(X_train_data, y_train_data):
    fold = KFold(n_splits=5, shuffle=False, random_state=0)

    c_param_range = [0.1, 1, 2, 5]
    gamma_range = [0.01, 0.1, 'auto', 1]
```

```
    0    0.1   0.01        NaN
    1    0.1    0.1        NaN
    2    0.1   auto        NaN
    3    0.1      1        NaN
    4    1.0   0.01  46.161881
    5    1.0    0.1  45.273756
    6    1.0   auto  45.273756
    7    1.0      1  45.273756
    8    2.0   0.01  73.268439
    9    2.0    0.1  45.501406
    10   2.0   auto  45.728842
    11   2.0      1  45.273756
    12   5.0   0.01  73.268439
    13   5.0    0.1  45.501406
    14   5.0   auto  45.728842
    15   5.0      1  45.273756

    ********************************************************************************
    Best model to choose from cross validation is with C =  1.0 and best gamma =  0.1
    ********************************************************************************
```

The optimal SVM model is:

```
print "the model classification for 8 proportion"
optimal_svm = SVC(gamma=0.1, random_state=0)
```

```
the model classification for 8 proportion
the recall for this model is : 0.7764227642276422
The accuracy is : 0.9996137735378695
TP 382
TN 284315
FP 0
FN 110
```



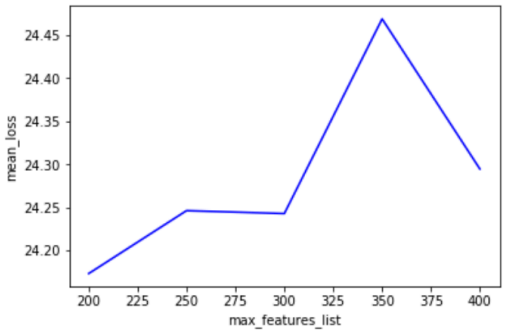

```
Classification Report:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00    284315
          1       1.00      0.78      0.87       492

avg / total       1.00      1.00      1.00    284807

The loss is :  27.303252032520334
```

6.3 Random Forest

Tuning n_estimators, max_features, max_depth, min_samples_split and min_samples_leaf.

```
def Kfold_tuning_rf(X_train_data, y_train_data):
    fold = KFold(n_splits=5, shuffle=False, random_state=0)

    n_estimators_range = [10, 100, 150, 200]
    max_features_type = ['auto', 'log2']
    max_depth_range = [10, 20, 30, None]
```

```
25          200         auto        20.0  24.173481
26          200         auto        30.0  24.196070
27          200         auto         NaN  24.196070
28          200         log2        10.0  34.680868
29          200         log2        20.0  24.173481
30          200         log2        30.0  24.196070
31          200         log2         NaN  24.196070

*********************************************************************************************
Best model to choose from cross validation is with n_estimators =  200 , best max_features =  auto
and best max_depth =  20.0
*********************************************************************************************
```



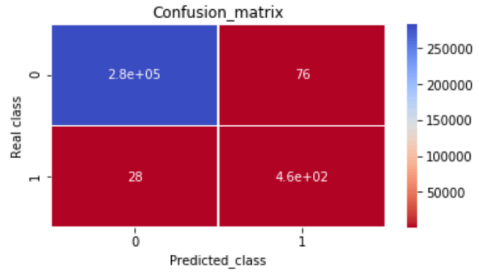

```
def Kfold_tuning_rf1(X_train_data, y_train_data):
    fold = KFold(n_splits=5, shuffle=False, random_state=0)

    max_features_range = ['auto', 1, 2, 8]
    min_samples_split_range = [2, 4, 8]
    min_samples_leaf_range = [1, 2, 4]
```

```
29          8                2                 4  38.130634
30          8                4                 1  25.678454
31          8                4                 2  34.780427
32          8                4                 4  38.130634
33          8                8                 1  32.087464
34          8                8                 2  37.050924
35          8                8                 4  38.130634

*********************************************************************************************
Best model to choose from cross validation is with best max_features =  1 , best min_samples_split =  2
and best min_samples_leaf =  1
*********************************************************************************************
```

The optimal random forest model is:

```
print "the model classification for 70 proportion"
optimal_rf = RandomForestClassifier(n_estimators=200, max_features=1, max_depth=20, random_state=0)
```

```
the model classification for 70 proportion
the recall for this model is : 0.943089430894309
The accuracy is : 0.9996348404358039
TP 464
TN 284239
FP 76
FN 28
```



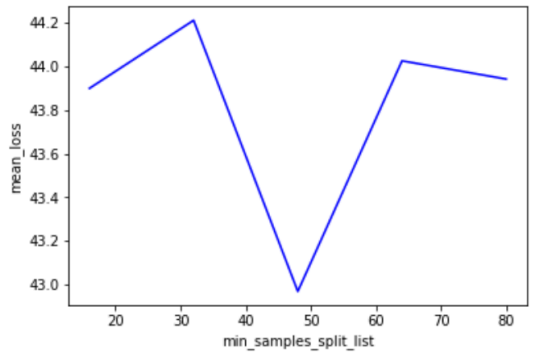

```
Classification Report:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00    284315
          1       0.86      0.94      0.90       492

avg / total       1.00      1.00      1.00    284807

The loss is :  19.37591869918699
```

6.4 GBT

Fixed learning_rate, tuning n_estimators, max_depth, min_samples_split and subsample:

```
def Kfold_tuning_gbt(X_train_data, y_train_data):
    fold = KFold(n_splits=5, shuffle=False, random_state=0)

    n_estimators_range = [100, 200, 300]
    max_depth_range = [3, 5, 7, 9]
    min_samples_split_range = [2, 4, 8, 16]
```

```
42          300          7                 8  46.333546
43          300          7                16  45.495394
44          300          9                 2  47.287604
45          300          9                 4  45.013025
46          300          9                 8  45.707437
47          300          9                16  44.948059

*********************************************************************************************
Best model to choose from cross validation is with n_estimators =  300.0 and best max_depth =  5.0
and best min_samples_split =  16.0
*********************************************************************************************
```




```
def Kfold_tuning_gbt1(X_train_data, y_train_data):
    fold = KFold(n_splits=5, shuffle=False, random_state=0)

    subsample_range = [0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9]
```

```
   Subsample  Mean loss
0       0.60  93.188976
1       0.65  87.075131
2       0.70  79.050653
3       0.75  59.514786
4       0.80  57.399793
5       0.85  57.345648
6       0.90  53.465095

**********************************************************************
Best model to choose from cross validation is with subsample =  0.9
**********************************************************************
```

When subsample = 0.9, the mean loss is 53.46 which is larger than the one(42.97) with subsample=1(default). Thus, the optimal subsample=default(1.0).

Then tuning learning_rate and change n_estimators inversely proportionally.

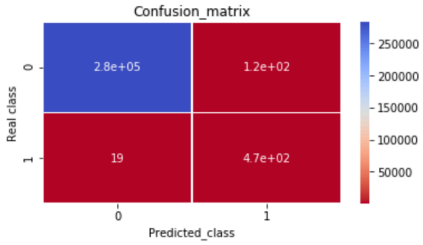
```
def Kfold_tuning_gbt2(X_train_data, y_train_data):
    fold = KFold(n_splits=5, shuffle=False, random_state=0)

    learning_rate_range = [0.1, 0.05, 0.01, 0.005]
    n_estimators_range = [300, 600, 3000, 6000]
```

```
   Learning_rate  N_estimators  Mean loss
0          0.100           300  42.968934
1          0.050           600  40.521305
2          0.010          3000  30.931739
3          0.005          6000  25.880417

**************************************************************************
Best model to choose from cross validation is with best learning rate =  0.005
**************************************
```

The optimal GBT model is:

```
print "the model classification for 116 proportion"
optimal_gbt = GradientBoostingClassifier(learning_rate=0.005, n_estimators=6000, max_depth=5,
                                         min_samples_split=48, random_state=0)
```

```
the model classification for 116 proportion
the recall for this model is : 0.9613821138211383
The accuracy is : 0.999515461347509
TP 473
TN 284196
FP 119
FN 19
```




```
Classification Report:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00    284315
          1       0.80      0.96      0.87       492

avg / total       1.00      1.00      1.00    284807

The loss is :  22.463499368270703
```

6.5 Optimal Model

Models	Precision	Recall	F1	Accuracy	Custom Loss	Diff vs 1st table	Diff vs 2nd table
Logistic Regression	0.79	0.78	0.79	0.99	45.41	-20.21	-49.74
***Support Vector Machine**	1.00	0.78	0.88	0.99	26.87	-91.59	-1.22
***Random Forest**	0.86	0.94	0.90	0.99	19.69	-16.14	-1.09
Gradient Boosting Tree	0.80	0.96	0.87	0.99	22.54	-59.84	-14.26

All the models have got improvements from tuning hyperparameters especially for Logistic Regression and Gradient Boosting Tree. In conclusion, SVM and Random Forest are more sensitive to the proportion of undersampling compared to tuning hyperparameters. Logistic Regression and GBT are sensitive to both of them.

Random Forest is still the best among the 4 models. Actually the improvement after tuning hyperparameters is trivial for Support Vector Machine (non-linear) and Random Forest. Anyway, better than nothing. It still saves about $ 1.09 per (fraud + misclassification normal) transaction for us.

SVM has a 100% precision which means if a transaction is classified as fraud by SVM then it must be a fraud. Thus, a typical predication process would be like this: predict using SVM first, if result is fraud then classify the transaction as fraud. Otherwise, predict again using Random Forest and return the prediction result. By doing this, we could increase the recall rate as far as possible without doing any harm to precision.

The optimal SVM and Random Forest models are:

SVC(C=1, gamma=0.1)

RandomForestClassifier(n_estimators=200, max_features=1, max_depth=20, min_smaples_split=2, min_samples_leaf=1)

7. Model Deployment

https://github.com/MarcusNEU/INFO7390_2018Spring/blob/master/FinalProject/Model%20Design/pickle_models.ipynb

SVM and Random Forest models have been pickled and uploaded to Amazon S3 for later using in web application:

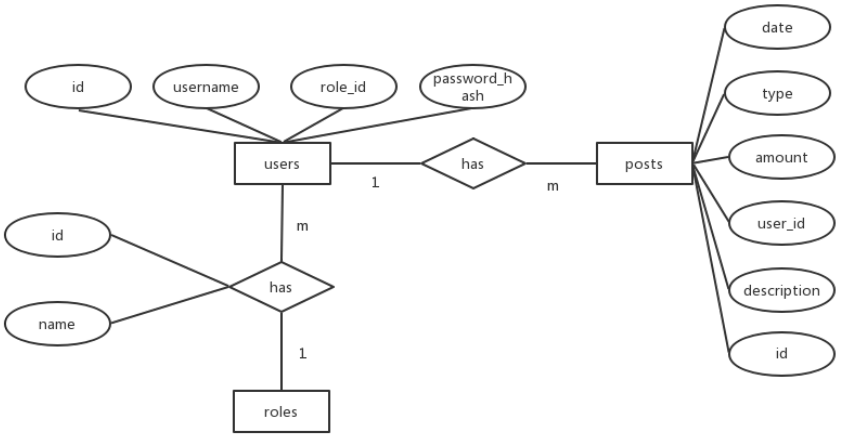
Name	Last modified	Size
random_forest_columns.pkl	Apr 26, 2018 3:34:40 PM GMT-0400	221.0 B
random_forest_model.pkl	Apr 26, 2018 3:34:40 PM GMT-0400	16.7 MB
svm_columns.pkl	Apr 26, 2018 3:35:04 PM GMT-0400	221.0 B
svm_model.pkl	Apr 26, 2018 3:35:04 PM GMT-0400	1.6 MB

8. Web Application Development

In this part the web application for credit card fraud detection is implemented. Main tool for web application is Flask and sqlite.

8.1 Relational Database

A relational database is used to store relationships between tables and data. Three tables are established: users, posts, roles. The relationship between these three tables is shown as follows:



Flask-Migrate is used to keep track of changes to a database schema, and then incremental changes can be applied to the database. The migrate subcommand creates an automatic migration script, use commend:

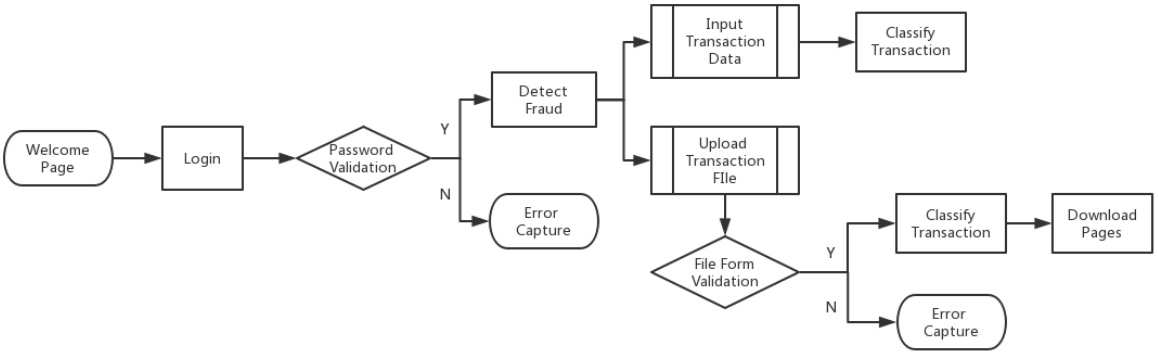
python #.py db migrate -m “migration_script_name ”

Once a migration script has been reviewed and accepted, it can be applied to the database using the db upgrade commend:

python #.py db upgrade

8.2 Application Development

Whole use case of the application will be looked like this:



Welcome page when access http://159.65.231.188:5000

```
@app.route('/', methods=['GET', 'POST'])
def welcome():
    return render_template('welcome.html')
```

At the beginning, a welcome page will present to user, and users can choose login button in the navigation bar at the top of our website.

Login pages when user choose login in navigation bar

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()

        if user is not None and user.verify_password(form.password.data):
            login_user(user, form.remember_me.data)
            print'sucess'
            return redirect(url_for('index'))
```

```
        flash('Wrong password')
    return render_template('login.html', title='Sign In', form=form)
```

After choosing login, a login page will appear and only users exist in database will be allowed to access the detection page. And there are two kinds of user: admin and user. Admin can access the transaction data includes anonymous columns. So an admin can using these special form of data to predict the transaction label as well as inputting transaction data with normal form of columns to predict while customer user can only access transaction data which includes: time, description, amount, type.

Upload page after login and choose upload transaction file

```
@app.route('/choose', methods=['GET', 'POST'])
@login_required
def upload_file():
    if request.method == 'POST':
        file_uploaded = request.files['upload_file']
        preview_parameter = classification.data_processing(file_uploaded)
        output_path = classification.form_download_file(OUTPUT_FOLDER,
preview_parameter[0])
        return render_template('download_test.html',
output_row=preview_parameter[0], total_rows=preview_parameter[1]
                               , output_path=output_path)
    else:
        return render_template('upload_test.html')
```

After receiving a csv file from user, then the application will call the data_processing() function to make a classification and present the final results Fraud/Normal in the web page. If the user wants to download the result. The application will call the form_download_file() function to let the user download result csv file. These two functions are all come from a class named classification, which includes the procession of download models and column index, then classify the transaction data, and every step of the whole thing will be monitored. For each error happened, user will get the error information presented in the website.

Input page after login and choose input transaction data

```
@app.route('/index', methods=['GET', 'POST'])
@login_required
def index():
    form = UserForm()
    if form.validate_on_submit():
        print 'begin detect'
```

```
            print form.amount.data
        return render_template('index.html', form=form)

•
```

This part is for user to input their transaction data, and a special algorithm will use these data to generate 28 anonymous data to predict the result (These steps can't be achieved because of the confidentiality of this algorithm).