Transaction fraud analysis and modelling

Rodrigo Cepeda Marín

1 Introduction

In this document I will analyze the *Credit Card Transactions Fraud Detection* data set obtained from Kaggle with the objective of creating a model able to predict the fraudulence on future operations.

The data is divided in two disjoint data sets, the *train* data set, which will be used in the exploratory data analysis and in the model construction process, and the *test* data set, which will be used to measure the efficacy of the predictive model created.

2 Initial analysis and data treatment

The train data set is formed of 1.2 million credit card records, (where 0.5% are fraudulent and the remaining are legal) containing the following relevant information:

- *trans_date_trans_time*: The date and time of the transaction.
- *merchant*: The business owner to where the transaction was issued.
- *category*: The type of product or service purchased.
- *amt*: The amount of money transferred in the operation.
- *gender*: The gender of the credit card owner.
- *city_pop*: The city population where the transaction was issued
- *dob*: Date of birth of the credit card owner. The combination of *dob* and *trans_date_trans_time* will let us know the age of the credit card owner when the transaction was made.
- *merch_lat*: The latitude of the business where the transaction was issued.
- *merch_long*: The longitude of the business where the transaction was issued.
- *is_fraud*: Whether the transaction is a fraud or not.

The initial function to rearrange the information is the following:

```
def _in_change(dataset, _in_col):
    dataset = dataset.iloc[:, 1:]
    dataset.set_index('trans_num', inplace = True)
    dataset['trans_date_trans_time'] = dataset['trans_date_trans_time'].\
        apply(lambda x: datetime.strptime(x, "%Y-%m-%d %H:%M:%S"))
    dataset['dob'] = dataset['dob'].\
        apply(lambda x: datetime.strptime(x, "%Y-%m-%d"))
    dataset['age_when_trans'] = (dataset['trans_date_trans_time']
                                 - dataset['dob']).dt.days / 365
    #dataset_Y = dataset['is_fraud']
    dataset = dataset[columns_good]
    return dataset

_in_col = ['trans_date_trans_time', 'category', 'amt',
               'gender', 'age_when_trans', 'city_pop',
               'merch_lat', 'merch_long', 'merchant', 'is_fraud']
test_X, train_X = _in_change(_test, _in_col), _in_change(_train, _in_col)
```

After rearranging the data, lets begin with some exploratory analysis. The first piece of information that is worth investigating is the geographical location of the transactions, both legal and fraudulent. For that, we can make use of library `geopandas`, which allows to download maps as .shp files an plot on top of them.

```
us_map = gpd.read_file('us/USA_States.shp')
map_plot = _train[['merch_lat', 'merch_long', 'is_fraud']]
geometry = [Point(xy) for xy in zip(map_plot['merch_long'],
            map_plot['merch_lat'])]
crs = {'init' : 'epsg:4326'}
geo_df = gpd.GeoDataFrame(map_plot, crs = crs,
                          geometry = geometry)

fig, ax = plt.subplots(figsize = (40, 40))
us_map.plot(ax = ax, alpha = 0.4, color = 'grey')
geo_df.plot(ax = ax, markersize = 40, edgecolor='black',
            linewidth=1, color = 'lightblue')
```

First we are going to look at all transactions, regardless of whether they are fraudulent or not:

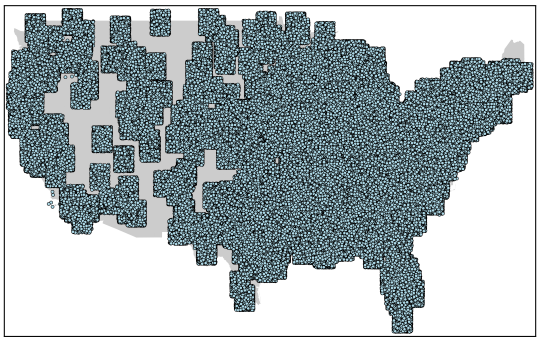


Figure 1: Total transactions in the U.S.

And now, lets view the legal transactions represented as green dots, and the fraudulent transactions, represented as red dots:

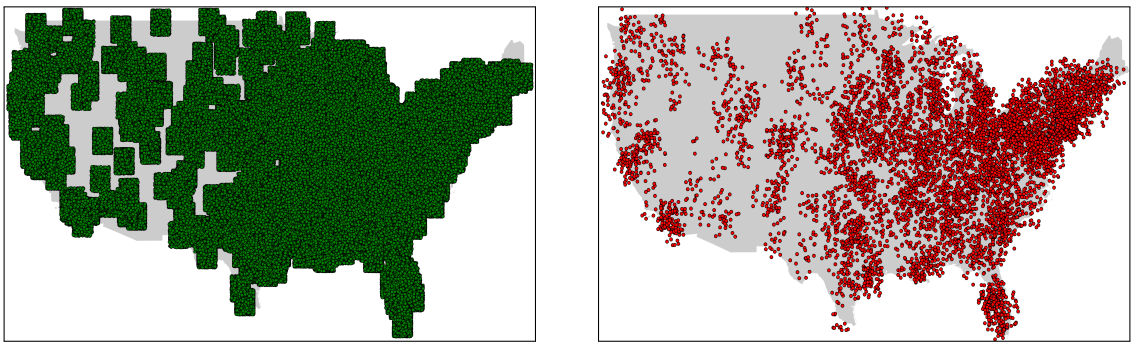
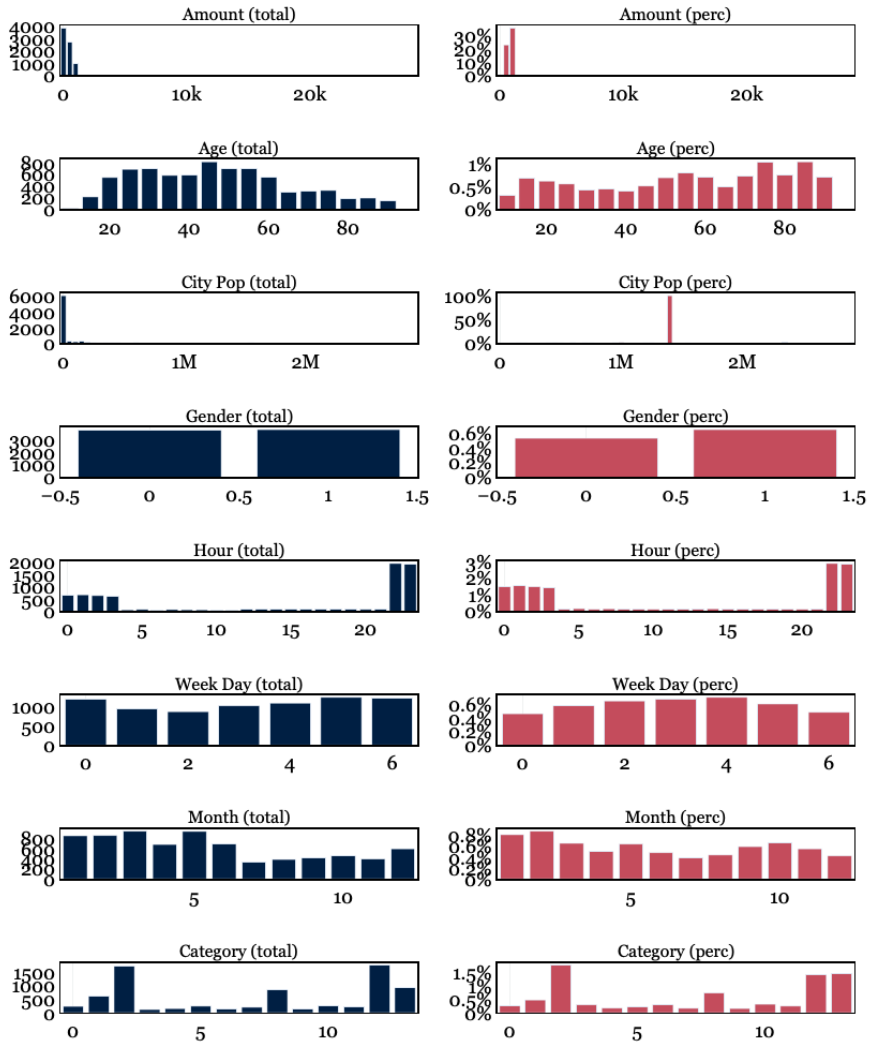


Figure 2: Legal and fraudulent operations in the U.S.

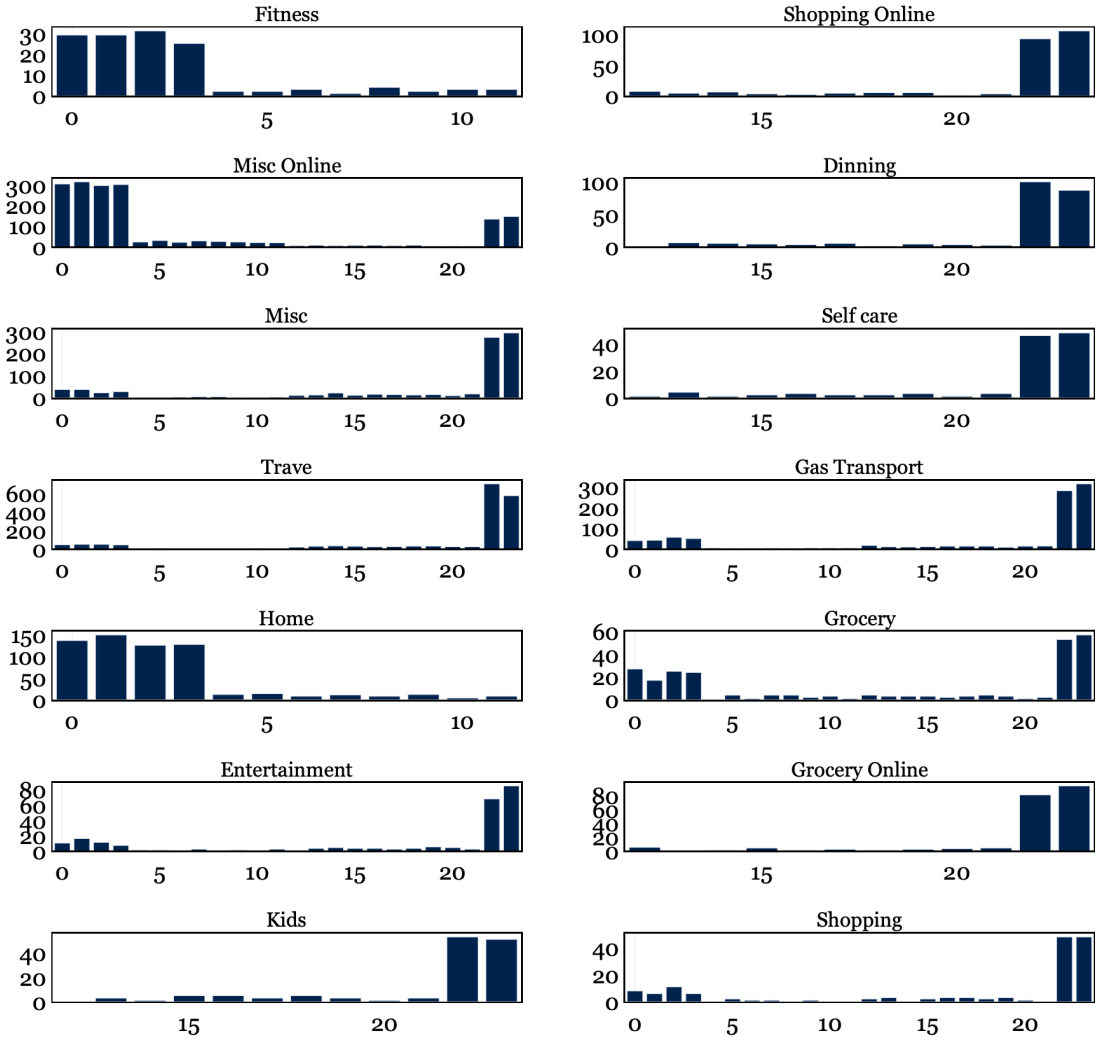
The fraudulent transactions seem to be equally spread throughout the country, there does not seem to be any identifiable pattern to the naked eye within the geographical distribution of the frauds.

Lets continue by analyzing the distribution of fraudulent transactions for *age_when_trans*, *amt*, *gender*, *city_pop*, *category*, hour, weekday and month of the operation. The distributions can offer a first idea of tendencies within the data.



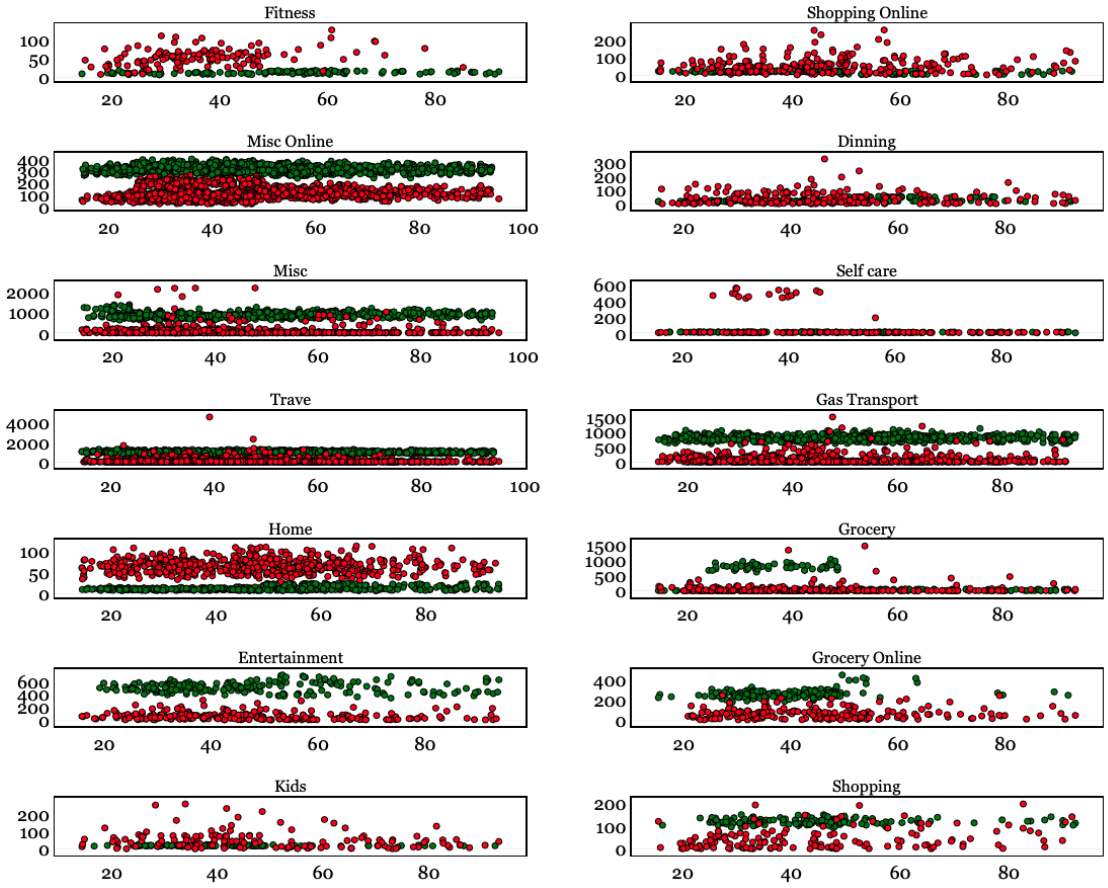

```
four_plot = train_X[['age_when_trans', 'amt', 'gender',
                     'city_pop', 'is_fraud']].copy()
four_plot['Amount'] = four_plot['amt'].apply(lambda x: 500 * np.floor(x / 500))
four_plot['Age'] = four_plot['age_when_trans'].\
                   apply(lambda x: 5 * np.floor(x / 5))
four_plot['City Population'] = four_plot['city_pop'].\
                   apply(lambda x: 50000 * np.floor(x / 50000))
four_plot['hour'] = four_plot['trans_date_trans_time'].dt.hour
four_plot['wday'] = four_plot['trans_date_trans_time'].dt.weekday
four_plot['month'] = four_plot['trans_date_trans_time'].dt.month
fig = make_subplots(rows=4, cols=2,
                    subplot_titles=("Amount (total)", "Amount (perc)",
                                    "Age (total)", "Age (perc)",
                                    "City Pop (total)", "City Pop (perc)",
                                    "Gender (total)", "Gender (perc)",
                                    "Hour (total)", "Hour (perc)", "Week Day
                                    (total)", "Week Day (perc)","Month (total)",
                                    "Month (perc)"), vertical_spacing = 0.085)
```

The distribution by hour show that fraudulent operations have a tendency of being executed during the night. We can also analyze the distribution of fraud operations by hour for each of the categories.




```
fig = make_subplots(rows=7, cols=2,
                    subplot_titles=('Fitness', 'Shopping Online',
                                    'Misc Online','Dinning',
                                    'Misc', 'Self care',
                                    'Trave', 'Gas Transport',
                                    'Home', 'Grocery',
                                    'Entertainment', 'Grocery Online',
                                    'Kids', 'Shopping'), vertical_spacing = 0.085)
_contador = _row = _col = 0
for c in list(np.unique(train_X_cat_hour['category'])):
    _contador = _contador + 1
    if _contador == 1: _row = _row + 1; _col = 1
    if _contador == 2: _contador = 0; _col = 2
    _temp = train_X_cat_hour[train_X_cat_hour['category'] == c]
    fig1 = px.bar(_temp.groupby('hour')['is_fraud'].sum())
    for d in fig1.data:
        fig.add_trace((go.Bar(x = d['x'], y = d['y'], name = d['name'],
                              marker=dict(color = '#01224e'))),
                      row = _row, col = _col)
```

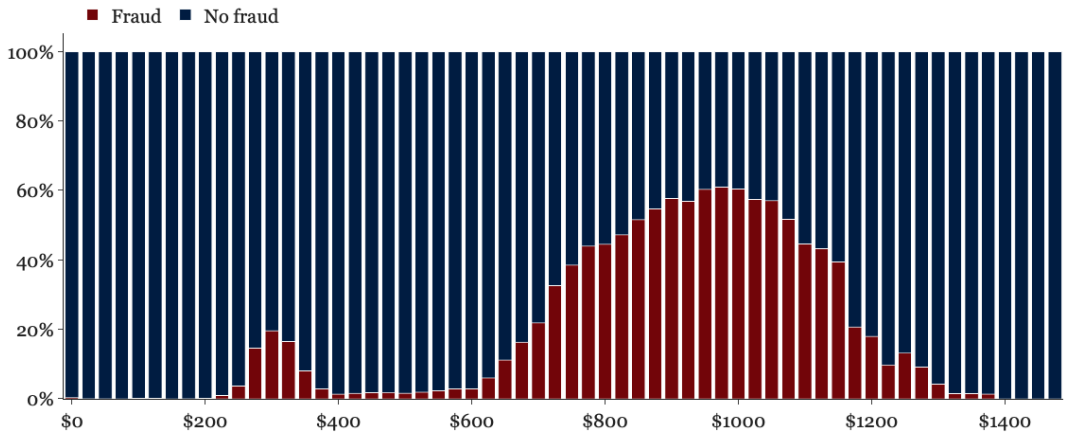
It is also interesting to analyze the fraudulent operations by amount and age for each category. That way we can try to understand shopping patterns depending on the time and target store, where the red dots represent the fraudulent transactions, and a sub sample of legal transactions is represented in green.




```
fig = make_subplots(rows=7, cols=2,
                    subplot_titles=('Fitness', 'Shopping Online',
                                    'Misc Online','Dinning', 'Misc', 'Self care',
                                    'Trave', 'Gas Transport', 'Home', 'Grocery',
                                    'Entertainment', 'Grocery Online',
                                    'Kids', 'Shopping'), vertical_spacing = 0.085)
_contador = _row = _col = 0
for c in list(np.unique(train_X_cat_amt['category'])):
    _contador = _contador + 1
    if _contador == 1: _row = _row + 1; _col = 1
    if _contador == 2: _contador = 0; _col = 2
    _temp = train_X_cat_amt[train_X_cat_amt['category'] == c]
    _temp = _temp[_temp['is_fraud'] == 1]
    fig1 = px.scatter(x = _temp['age_when_trans'], y = _temp['amt'])
    fig.add_trace((go.Scatter(x = fig1.data[0]['x'], y = fig1.data[0]['y'],
                              name = fig1.data[0]['name'], mode = 'markers',
                              marker=dict(color = '#01224e'))),
                              row = _row, col = _col)
```

After plotting and analyzing the data we can extract some initial conclusions:

1. 100% of the fraudulent operations are within the ($0 − $1300] interval. This could be because the limits set on credit cards, that don't allow high transactions in the same day. Also, the different treatment by the law of felony (up to $950) and robbery (more that %950) can be another reason.



2. 85% of the fraudulent transactions happen between 22 at night and 4 in the morning.
3. Fraud across gender is equally distributed, there is no distinction between male and female fraudulent transactions.
4. In regard to age, fraud in older population is considerably smaller in total terms, but it is statistically higher than the average. This is probably correlated to the knowledge of younger generation of internet safety in terms of sharing information online.
5. Categories 2 and 12 have the highest amount of fraudulent transactions. The categories correspond to online shopping and grocery shopping.
6. 94% of the transactions between $200 and $1300 made at night in groceries are fraudulent. Who goes shopping at night?
7. 90% of the transactions between $250 and $1300 in groceries are fraudulent
8. Delaware has a 100% of fraudulent transactions (9 out of 9), which is very likely an error in the data set. After that, the state with the highest fraud rate is Rhode Island, where 2.7% of transactions are non legal.
9. Fraud across age is evenly distributed. Looking at the age-amount graph by category, all the behaviours seem similar and in line with what has been said before; the mid amount transactions have a big tendency to be fraudulent.
10. Last and most important: the data is very imbalanced. Only 0.5% of the transactions are fraud. Building a model with this condition is very complicated because setting every record to non fraud would generate an accuracy of a 99.5%. It is necessary to correct the unevenness of the model.

2.1 Correcting the imbalances in the data set

As mentioned before, the data set is highly imbalanced, with 99.5% of the records of one class. It is not possible to train the model with the current data set as it is, because the model will simply classify all the records in the same class, obtaining almost a 100% accuracy. To solve this issue, two solutions can be used, and we will use both:

1. Under sampling: This consists in randomly sample an amount of records for the most repeated class, reducing then the imbalancement

```
    def _reduce(dataset,d_y, size):
    dataset['is_fraud'] = d_y
    dataset_f = dataset[dataset['is_fraud'] == 1].copy()
    dataset_nf = dataset[dataset['is_fraud'] == 0].copy().sample(size)
    dataset_r = pd.concat([dataset_f, dataset_nf])
    dataset_y = dataset_r['is_fraud']
    dataset_r.drop(columns = ['is_fraud'], inplace = True)
    return dataset_r, dataset_y
    X_train, y_train = _reduce(X_train,y_train, 50000)
```

2. Over sampling: This approach consists in generating synthetic for the least present class until the classes are balanced. The approach taken is named SMOTE.

```
    def _smote(dataset, dataset_Y):
    balance = SMOTE()
    dataset, dataset_Y = balance.fit_resample(dataset, dataset_Y)
    return dataset, dataset_Y
```

After correcting the imbalances in the data set, the train set has 100000 records of which 50000 are fraud and 50000 correspond to legal transactions.

2.2 Principal Components Analysis

In order to reduce the complexity of the problem without loosing information, we can perform the non supervised algorithm of PCA. The first n eigenvectors of the covariance matrix will explain the information good enough while reducing the dimension of the problem.

```
def sep_data(dataset):
    data_Y = dataset['is_fraud']
    data_X = dataset.drop(columns = ['is_fraud'])
    return datapca_X, data_Y
_X, y = sep_data(data)
pca = PCA(n_components = _X.shape[1])
X_pca = pd.DataFrame(pca.fit_transform(_X))
```

3 Modelling

In this section we will compare two classification models, a simple random forest model and a far more complex deep learning structure, to find out the differences in performance between them.

3.1 Random Forest Model

The first model that we will develop is simple random forest model

```
from sklearn.tree import DecisionTreeClassifier
dcstree = DecisionTreeClassifier(random_state=42)
dcstree.fit(X_train, y_train)
```

The model has produced the next results when predicting into the test data set:

271195 (True No Fraud)	5195 (False Fraud)
122 (False No Fraud)	1348 (True Fraud)

As we can see, 91.7% of the frauds and 98.12% of the legal transactions are correctly classified. The amount of false positives is 5195, 3.85 times greater of the true positives classified.

3.2 Deep Learning Structure

The deep learning structure consists on an input and output layers and 5 fully connected deep layers.

```
model = Sequential()
model.add(Input(shape = X_train.shape[1]))
model.add(Dense(256, activation = 'relu'))
model.add(Dense(128, activation = 'sigmoid'))
model.add(Dense(128, activation = 'sigmoid'))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))
model.summary()
model.compile(loss='binary_crossentropy',
              metrics=['acc']) #Utilizamos 'binary_crossentropy' porque
                               #tenemos solo dos clases, 0 y 1.
_nepochs = 4
history = model.fit(X_train,
                    y_train,
                    epochs = _nepochs,
                    #batch_size = _batch_size,
                    verbose = True)
y_pred = model.predict(X_test)
y_pred[y_pred >= 0.5] = 1
y_pred[y_pred != 1] = 0
```

The last layer has a sigmoid activation function on the output layer because the problem is classification based, the outputs are turned in some sense to probabilities with the function $\frac{1}{1+e^{-x}}$ The model has produced the next results when predicting into the test data set:

271782 (True No Fraud)	4608 (False Fraud)
114 (False No Fraud)	1356 (True Fraud)

As we can see, 84.7% of the frauds and 98.77% of the legal transactions are correctly classified. The amount of false positives is 4608, 2.7 times greater of the true positives classified.

4 Conclusion

Comparing both classification approaches we can see the advantages of each of them: the random fores is more accurate predicting fraud but also has a high ratio of false frauds to fraud predictions. Meanwhile, the deep learning structure is slightly less accurate when predicting true frauds but doesn't lead to as many false positives as the random forest approach. Also, it is important to remark that the deep learning structure can be adjusted to fit the problem better using a hyperparamether tuning software such as Keras Tuner. That approach hasn't been taken is this document.