Credit Card Fradulent Detection

Report written in Partial Fulfilment

Of credit Requirement for course

IE 615

Course project in

Data Analytics in Operations Research

Submitted by

1. 193190008 Rahul Khankar
2. 193190014 Yogesh nagar
3. 193190019 Pratik Zaveri

Industrial engineering and Operations Research

Indian Institute of Technology Bombay

Powai, Mumbai, India – 400076

Winter Semester 2019

Table of content

1. Introduction

It is important that credit card companies are able to recognize fraudulent credit card transactions. We have gathered data from kaggle.com for various transactions. We have trained our model using a various categorical and numerical feature

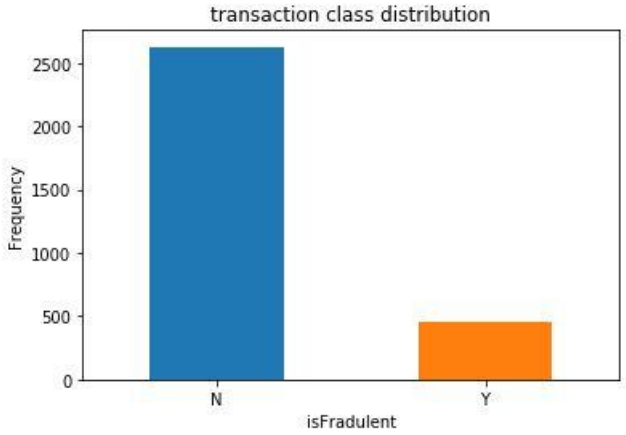
Problem Statement

The Credit Card Fraud Detection Problem includes modelling past credit card. transactions with the knowledge of the ones that turned out to be a fraud. This model is then used to identify whether a new transaction is fraudulent or not. Our aim is to detect fraudulent transactions. We have used the implementation of different Classifications Algorithms

2. Data Preprocessing

2.1 Imbalance learn

We will see how our data is spread between two categories fraudulent and not a fraudulent transaction.



Now we can see that our dataset is highly imbalanced. Data set has most of the transaction Not fraudulent and very less number of fraudulent transaction.

We will use every classification algorithm 2 times. One before Oversampling and other after Oversampling.

Label	Before over sampling	After over sampling
Data of Fraud Transactions	448	2627
Data of Not a Fraud Transaction	2627	2627

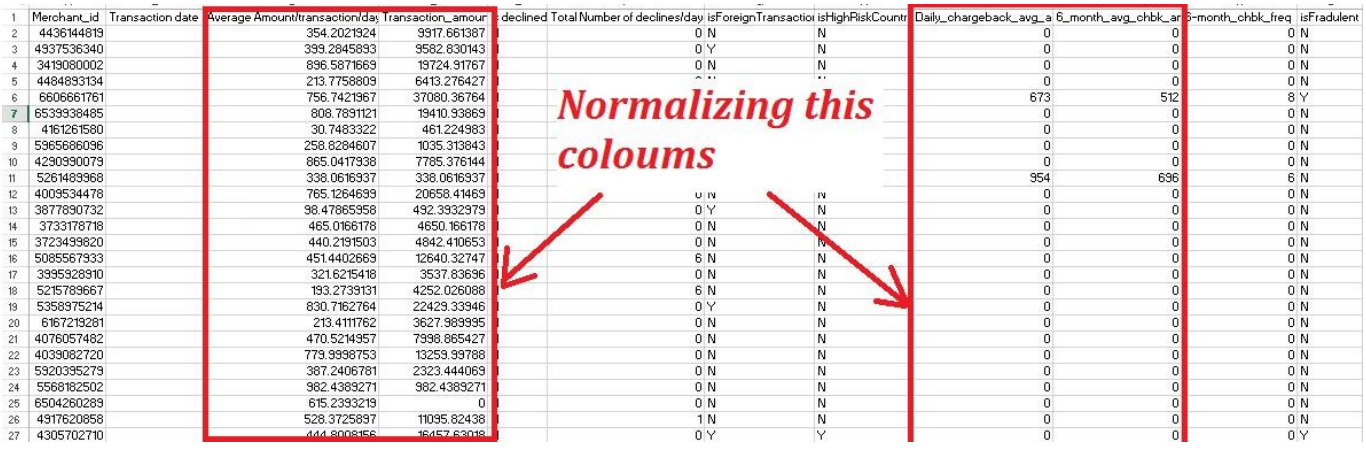
We have split our data into two sets 80% for Training and 20% for the test case.

2.2 Handling Datapoints

a) We don't require Merchant id for fraudulent detection and transaction date column is empty we will drop this two-column

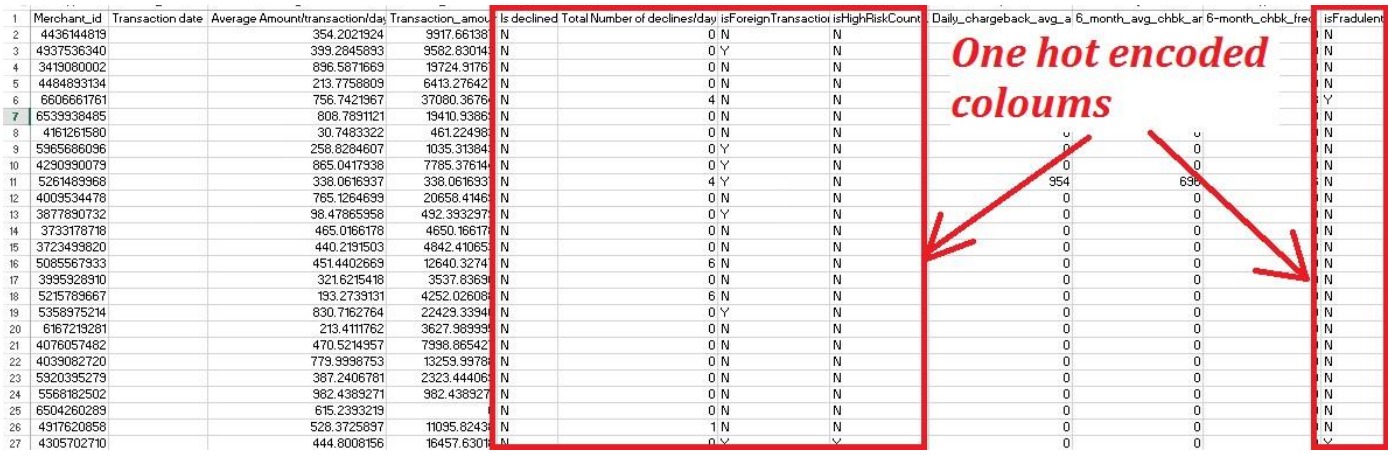
	Merchant_id	Transaction date	Average Amount/transaction/day	Transaction_amour	Is declined	Total Number of declines/day	isForeignTransactio	isHighRiskCountry	Daily_chargeback_avg_a	6_month_avg_chbk_ar	6-month_chbk_freq	isFradulent
2	4436144819		354.2021924	9317.661387	N	0	N	N	0	0	0	N
3	4937536340		399.2845893	9582.830143	N	0	Y	N	0	0	0	N
4	3419080002		896.5871669	19724.91767	N	0	N	N	0	0	0	N
5	4484893134		213.7758809	6413.276427	N	0	N	N	0	0	0	N
6	6606661761		756.7421967	37080.36764	N	4	N	N	673	512	8	Y
7	6539938485		808.7891121	19410.93869	N	0	N	N	0	0	0	N
8	4161261580		30.7483322	461.224983	N	0	N	N	0	0	0	N
9	5965686096		258.8284607	1035.313843	N	0	Y	N	0	0	0	N
10	4290990079		865.0417938	7785.376144	N	0	Y	N	0	0	0	N
11	5261489368		338.0616937	338.0616937	N	4	Y	N	954	696	6	N
12	4009534478		765.1264639	20658.41469	N	0	N	N	0	0	0	N
13	3877890732		98.47865958	492.3932979	N	0	Y	N	0	0	0	N
14	3733178718		465.0166178	4650.166178	N	0	N	N	0	0	0	N
15	3723499820		440.2191503	4842.410653	N	0	N	N	0	0	0	N
16	5085567933		451.4402669	12640.32747	N	6	N	N	0	0	0	N
17	3995928910		321.6215418	3537.83696	N	0	N	N	0	0	0	N
18	5215789667		193.2739131	4252.026088	N	6	N	N	0	0	0	N
19	5358975214		830.7162764	22429.33946	N	0	Y	N	0	0	0	N
20	6167219281		213.4111762	3627.989395	N	0	N	N	0	0	0	N
21	4076057482		470.5214957	7998.865427	N	0	N	N	0	0	0	N
22	4039082720		779.9998753	13259.99788	N	0	N	N	0	0	0	N
23	5320395279		387.2406781	2323.444069	N	0	N	N	0	0	0	N
24	5568182502		982.4389271	982.4389271	N	0	N	N	0	0	0	N
25	6504260289		615.2393219	0	N	0	N	N	0	0	0	N
26	4917620858		528.3725897	11095.82438	N	1	N	N	0	0	0	N
27	4305702710		444.8008156	16457.63018	N	0	Y	Y	0	0	0	Y

b) Since some columns have a wide variation in their values we have normalized them between 0 to 1 for fast calculations.



	Merchant_id	Transaction date	Average Amount/transaction/day	Transaction_amour	declined	Total Number of declines/day	isForeignTransactio	isHighRiskCountr	Daily_chargeback_avg_a	6_month_avg_chbk_ar	6-month_chbk_freq	isFradulent
2	4436144819		354.2021924	9317.661387		0	N	N	0	0	0	N
3	4937536340		399.2845893	9582.830143		0	Y	N	0	0	0	N
4	3419080002		896.5871669	19724.91767		0	N	N	0	0	0	N
5	4484893134		213.7758809	6413.276427		0	N	N	0	0	0	N
6	6606661761		756.7421967	37080.36764		4	N	N	673	512	8	Y
7	6539938485		808.7891121	19410.93869		0	N	N	0	0	0	N
8	4161261580		30.7483322	461.224983		0	N	N	0	0	0	N
9	5965686096		258.8284607	1035.313843		0	Y	N	0	0	0	N
10	4290990079		865.0417938	7785.376144		0	Y	N	0	0	0	N
11	5261489368		338.0616937	338.0616937		4	Y	N	954	696	6	N
12	4009534478		765.1264639	20658.41469		0	N	N	0	0	0	N
13	3877890732		98.47865958	492.3932979		0	Y	N	0	0	0	N
14	3733178718		465.0166178	4650.166178		0	N	N	0	0	0	N
15	3723499820		440.2191503	4842.410653		0	N	N	0	0	0	N
16	5085567933		451.4402669	12640.32747		6	N	N	0	0	0	N
17	3995928910		321.6215418	3537.83696		0	N	N	0	0	0	N
18	5215789667		193.2739131	4252.026088		6	N	N	0	0	0	N
19	5358975214		830.7162764	22429.33946		0	Y	N	0	0	0	N
20	6167219281		213.4111762	3627.989395		0	N	N	0	0	0	N
21	4076057482		470.5214957	7998.865427		0	N	N	0	0	0	N
22	4039082720		779.9998753	13259.99788		0	N	N	0	0	0	N
23	5320395279		387.2406781	2323.444069		0	N	N	0	0	0	N
24	5568182502		982.4389271	982.4389271		0	N	N	0	0	0	N
25	6504260289		615.2393219	0		0	N	N	0	0	0	N
26	4917620858		528.3725897	11095.82438		1	N	N	0	0	0	N
27	4305702710		444.8008156	16457.63018		0	Y	Y	0	0	0	Y

c) Some of the features in the dataset have categorical, we have converted this categorical features in numerical by one hot encoding.



	Merchant_id	Transaction date	Average Amount/transaction/day	Transaction_amou	Is declined	Total Number of declines/day	isForeignTransactio	isHighRiskCount	Daily_chargeback_avg_a	6_month_avg_chbk_ar	6-month_chbk_fre	isFradulent
2	4436144819		354.2021924	9917.661381	N	0	N	N				N
3	4937536340		399.2845893	9582.830143	N	0	Y	N				N
4	3419080002		896.5871669	19724.9176	N	0	N	N				N
5	4484893134		213.7758809	6413.27642	N	0	N	N				N
6	6606661761		756.7421967	37080.3676	N	4	N	N				Y
7	6539938485		808.7891121	19410.9386	N	0	N	N				N
8	4161261580		30.7483322	461.224983	N	0	N	N	0	0		N
9	5365686096		258.8284607	1035.31384	N	0	Y	N	0	0		N
10	4290990079		865.0417938	7785.37614	N	0	Y	N	0	0		N
11	5261489968		338.0616937	338.061693	N	4	Y	N	954	696		N
12	4009534478		765.1264699	20658.4146	N	0	N	N	0	0		N
13	3877890732		98.47865958	492.393297	N	0	Y	N	0	0		N
14	3733178718		465.0166178	4650.16617	N	0	N	N	0	0		N
15	3723439820		440.2191503	4842.41065	N	0	N	N	0	0		N
16	5085567933		451.4402669	12640.3274	N	6	N	N	0	0		N
17	3995928910		321.6215418	3537.8369	N	0	N	N	0	0		N
18	5215789667		193.2739131	4252.02608	N	6	N	N	0	0		N
19	5358975214		830.7162764	22429.3394	N	0	Y	N	0	0		N
20	6167219281		213.4111762	3627.98393	N	0	N	N	0	0		N
21	4076057482		470.5214957	7998.86542	N	0	N	N	0	0		N
22	4039082720		779.9998753	13259.9978	N	0	N	N	0	0		N
23	5920395279		387.2406781	2323.44406	N	0	N	N	0	0		N
24	5568182502		982.4389271	982.438927	N	0	N	N	0	0		N
25	6504260289		615.2393219	0	N	0	N	N	0	0		N
26	4917620858		528.3725897	11095.8243	N	1	N	N	0	0		N
27	4305702710		444.8008156	16457.6301	N	0	Y	Y	0	0		Y

3. Confusion matrix

Recall is the ratio of correctly identified fraud cases to total fraud cases.
Precision is the ratio of correctly predicted fraud cases to total predicted fraud cases.
Combination of this precision and recall gives F1 score as an output measure of accuracy.
Confusion Matrix provides as the comparison that the what amount of data are correctly and wrongly classified. i.e. True +ve or False +ve.

4. Classification Algorithms:

4.1 Random Forest

Algorithm:

1. Assume a number of cases in the training set are N. Then, Sample of these N cases is taken at random but with replacement.
2. If there are M input variables or features, a number m<M is specified such that at each node, m variables are selected at random out of M. The best split on this m is used to split the node. The value of m is held constant while we grow a forest.
3. Each tree is grown to the largest extent possible and there is no pruning.
4. Predict new data by aggregating the predictions of the n trees (i.e. majority votes for classification, the average for regression)

Result:

a) **Before Oversampling**:

(i) Confusion Matrix

Predicted Class→ **Actual Class ↓**	**Not a Fraud** **Transaction**	**Fraud** **Transactions**
Not a Fraud Transaction	516	2
Fraud Transactions	11	86

(ii) Classification report

Label	Precision	Re-call	F1-Score
0	0.98	1.00	0.99
1	0.98	0.89	0.93

(iii) Accuracy: 97.8%

b) After Oversampling :

(i) Confusion matrix

Predicted Class→ **Actual Class ↓**	**Not a Fraud** **Transaction**	**Fraud Transactions**
Not a Fraud Transaction	518	0
Fraud Transactions	1	96

(ii) Classification Report:

Label	Precision	Re-call	F1-Score
0	1.00	1.00	1.00
1	1.00	0.99	0.99

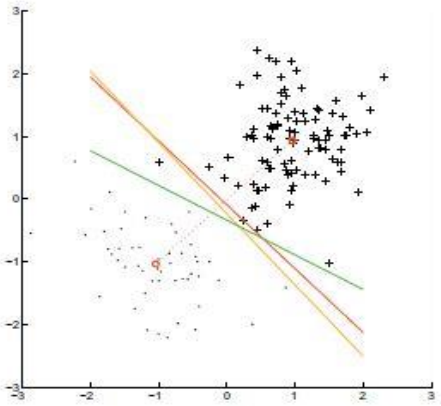
(iii) Accuracy: 99.83 %

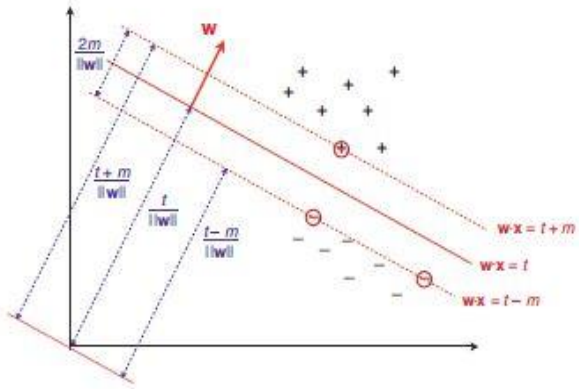
As we can observe in the confusion Metrix that after the pre-processing on the training data set the number of wrong predictions in the test data set are reduced.

4.2 Support Vector Machine (SVM)

- SVM is a supervised learning algorithm. This means that SVM trains on a set of labelled data. SVM studies the labelled training data and then classifies any new input data depending on what it learnt in the training phase.

- Linearly separable data have infinitely many decision boundaries that separate the classes, but some of them are better than others.





(i) Linear Seperator **(ii) SVM**

Algorithm:

In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class.

Optimisation problem:

$$\frac{argmin}{w,t} \quad \frac{1}{2} \left|\left|w^2\right|\right|$$

Subject to: Yi (w· xi −t) ≥ 1

Result

(a) Before Oversampling:

(i) Confusion matrix

Predicted Class→ **Actual Class ↓**	**Not a Fraud** **Transaction**	**Fraud Transactions**
Not a Fraud Transaction	522	1
Fraud Transactions	53	39

(ii) Classification Report:

Label	Precision	Re-call	F1-Score
0	0.91	1.00	0.95
1	0.97	0.42	0.59

(iii) Accuracy: 91%

(b) After Oversampling:

(i) Confusion matrix

Predicted Class→ **Actual Class ↓**	**Not a Fraud** **Transaction**	**Fraud Transactions**
Not a Fraud Transaction	447	76
Fraud Transactions	31	61

(ii) Classification Report:

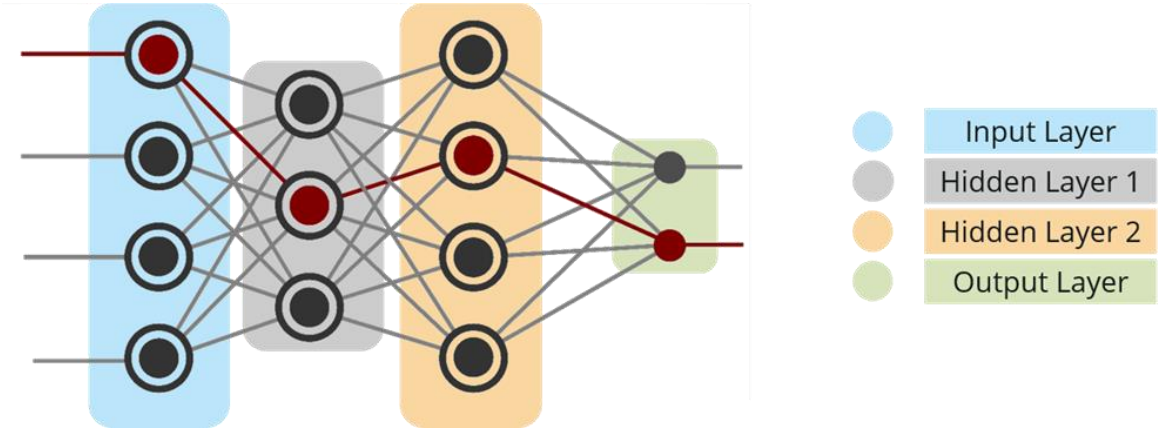
Label	Precision	Re-call	F1-Score
0	0.94	0.85	0.89
1	0.45	0.66	0.53

(iii) Accuracy: 83%

4.3 Multilayer Neural Network

Limitations of Single Layer Perceptron:-

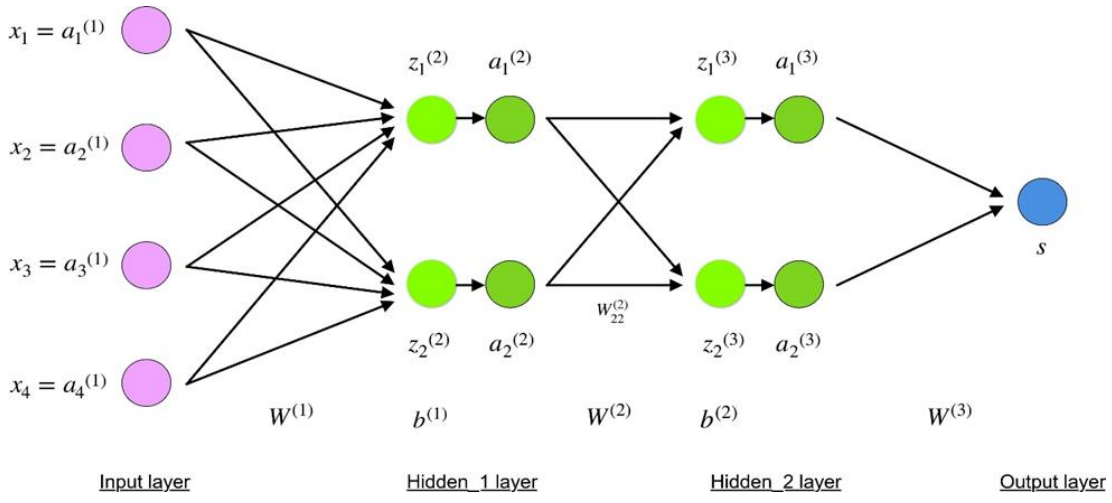
- Single-Layer Perceptron cannot classify non-linearly separable data points.
- Complex problems, that involve a lot of parameters cannot be solved by Single-Layer perceptron.



The neural network consists of neurons for the **input layer**, neurons for the **hidden layers** and 1 neuron for the **output layer**.

We have used 3 hidden layers, 1 input layer and 1 output layer. We have used the Relu activation function

Simple neural network illustration.



Input layer

The neurons, colored in purple, shows the input data. These are user provided values. The first set of activations (a) are the input values.

$$x_i = a_i^{(1)}, i \in 1,2,3,4$$ .....(Equation for input x(i))

Hidden layers

The final values at the hidden neurons, coloured in green, are computed by weighted inputs in layer l and activations in layer l. For layer 2 and 3 the equations are:

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

(Equations for z^2 and a^2).

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

(Equations for z^3 and a^3)

W^2 and W^3 are the weights in layer 2 and 3 while b^2 and b^3 are the biases in those layers.

Activations a^2 and a^3 are computed using an activation function f. Typically, this function f is non-linear (e.g. sigmoid, ReLU, tanh)

We have used **Relu Activation function.**

Relu = Max (0,X)

Let's pick layer 2 and its parameters as an example. The same operations can be applied to any layer in the network.

W^1 is a weight matrix of shape (n, m) where n is the number of output neurons (neurons in the next layer) and m is the number of input neurons (neurons in the previous layer).

$$W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \end{bmatrix}$$

(Equation for W^1)

The first number in any weight's subscript is the index of the neuron in the next and the second number matches the index of the neuron in previous layer

x is the input vector of shape (m, 1) where m is the number of input neurons.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

(Equation for x)

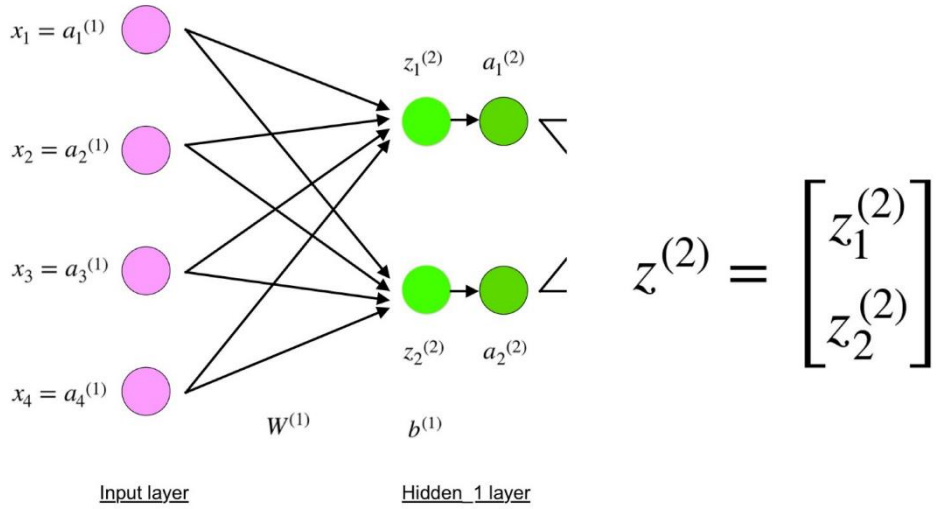
b^1 is a bias vector of shape (n , 1) where n is the number of neurons in the current layer.

$$b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

Following the equation for z^2, we can use the above definitions of W^1, x and b^1 to derive "Equation for z^2".

$$z^{(2)} = \begin{bmatrix} W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + W_{14}^{(1)}x_4 \\ W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + W_{24}^{(1)}x_4 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

(Equation for z^2)



$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \end{bmatrix}$$

Output layer :

The final part of a neural network is the output layer which produces the predicated value. In our simple example, it is presented as a single neuron, coloured in blue and evaluated as follows

$$s = W^{(3)} a^{(3)}$$

(Equation for outputs)

Backpropagation and computing gradients:

backpropagation aims to minimize the cost function by adjusting network's weights and biases. The level of adjustment is determined by the gradients of the cost function with respect to those parameters.
Compute gradients happens using a technique called chain rule.
For a single weight (w_jk) ^l, the gradient as follows.

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} \qquad chain\ rule$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad by\ definition$$

$$m\ -\ number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}a_k^{l-1} \qquad final\ value$$

Similar set of equations can be applied to (b_j)^l.

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} \qquad chain\ rule$$

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}1 \qquad final\ value$$

The common part in both equations is often called “local gradient” and is expressed as follows:

The “local gradient” can easily be determined using the chain rule.

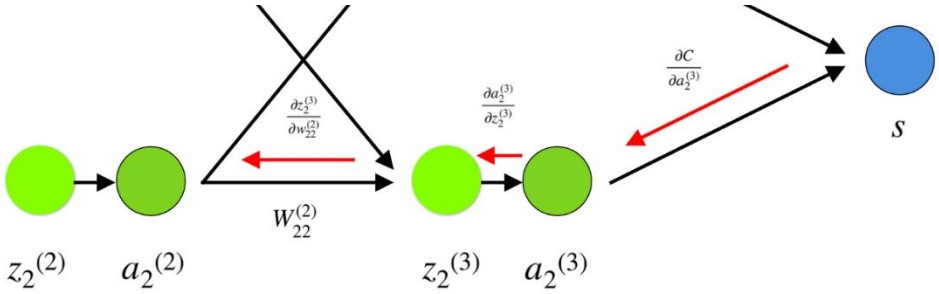
$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \qquad local\ gradient$$

The gradients allow us to optimize the model's parameters:

$$
\begin{aligned}
&while\ (termination\ condition\ not\ met)\\
&\quad w := w - \epsilon \frac{\partial C}{\partial w}\\
&\quad b := b - \epsilon \frac{\partial C}{\partial b}\\
&end
\end{aligned}
$$

(Algorithm for optimizing weights and biases (also called "Gradient descent"))

- Initial values of w and b are randomly chosen.
- Epsilon (e) is the learning rate. It determines the gradient's influence.
- w and b are matrix representations of the weights and biases. Derivative of C in w or b can be calculated using partial derivatives of C in the individual weights or biases.
- Termination condition is met once the cost function is minimized.



Weight *(w_22)²* connects *(a_2)²* and *(z_2)²*, so computing the gradient requires applying the chain rule through *(z_2)³* and *(a_2)³:*

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} \cdot f'(z_2^{(3)}) \cdot a_2^{(2)}$$

Calculating the final value of derivative of C in $(a\_2)^3$ requires knowledge of the function C. Since C is dependent on $(a\_2)^3$, calculating the derivative should be fairly straightforward.

Result

(a) Before oversampling

(i) Confusion Matrix

Predicted Class→ Actual Class ↓	Not a Fraud Transaction	Fraud Transactions
Not a Fraud Transaction	525	4
Fraud Transactions	4	82

(ii) Classification report

Label	Precision	Re-call	F1-Score
0	0.97	0.97	0.99
1	0.95	0.95	0.95

(iii) Accuracy: 99%

(b) After oversampling :

(iv) Confusion Matrix

Predicted Class→ Actual Class ↓	Not a Fraud Transaction	Fraud Transactions
Not a Fraud Transaction	441	88
Fraud Transactions	0	86

(v) Classification report

Label	Precision	Re-call	F1-Score
0	1	0.83	0.91
1	0.49	1	0.66

(vi) Accuracy: 86%

4.4 Logistic Regression

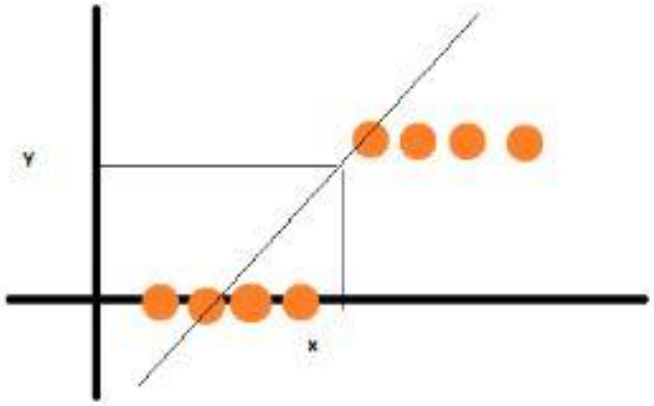
The linear model can work well for regression but fails for classification. Consider a case of two classes, we can label one of the classes with 0 and the other with 1 and use linear regression. it will work and a most linear algorithm will spit out weights for the user. But there are some problems with this approach:

A linear model does not output probabilities, but it considers the classes as numbers (0 and 1) and fits the best hyperplane that minimizes the distances between the points and the hyperplane. So it simply interpolates between the points, and we cannot interpret it as probabilities.

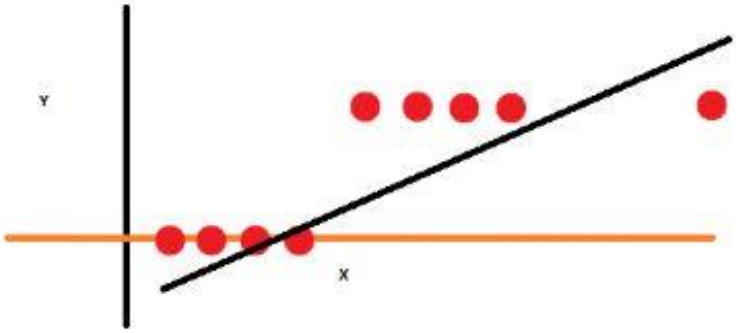
A linear model also extrapolates and gives you values below zero and above one. This is a good sign that there might be a smarter approach to classification.

Consider a problem with Linear Classifier

Suppose we have to classify **Y= {0,1}** and **X** are **a data point**. It is a binary classification. Let's try it with a linear classifier



The linear classifier has done the job. It is really a good fit but what happens if I add a new data to the given data set.

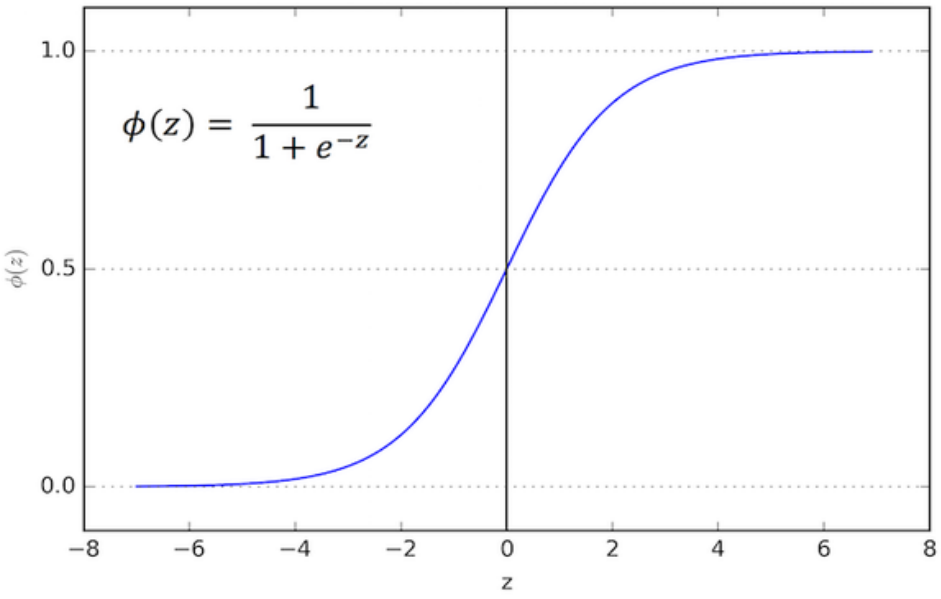


It's really a bad solution. Linear classifier didn't work.

the solution is **Logistic Regression**:

Intuitively, it also doesn't make sense for h(x) to take values larger than 1 or smaller than 0 when we know that y ∈ {0, 1}.To fix this, let us take another hypotheses h(x). We will choose hypothesis as follows :

$$h_\theta(x) = \frac{1}{1 + e^{-\theta.x}}$$



$$P(y = 1|x, \theta) = h_\theta(x)$$

$$P(y = 0|x, \theta) = 1 - h_\theta(x)$$

if we concatenate the above equations it can be rewritten as follows:

$$P(y|x, \theta) = (h_\theta(x)^y) * (1 - h_\theta(x)^{1-y})$$

As in above equation,

when y =0, p(y|x;theta) = (1-h(x)) and
when y =1 p(y|x;theta) = h(x)

How to update parameter

likelihood of the parameters are given by

$$L(\theta) = \prod_{i=1}^{m} p(y^i|x^i, \theta)$$

$$L(\theta) = \prod_{i=1}^{m} (h_\theta(x^i)^{y^i} * (1 - h_\theta(x^i)^{1-y^i}$$

and if we **Maximize log-likelihood,** it is given by :

$$l(\theta) = Log(L(\theta))$$

$$l(\theta) = \sum_{i=1}^{m} y^i * \log(h_\theta(x^i) + (1 - y^i) * (1 - \log(h_\theta(x^i))$$

we will use the **Gradient Descent algorithm**, we know that h(x) is given by the sigmoid function.

$$h_\theta(x) = \frac{1}{1 + e^{-\theta.x}}$$

Let us take it as g(z) for calculation simplicity

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = \frac{1}{(1+e^{-z})^2} * (e^{-z})$$

$$g'(z) = g(z) * (1 - g(z))$$

if apply gradient descent by taking the partial derivative of log-likelihood with respect to theta, then the partial derivative of likelihood becomes,

$$\frac{\partial\, l(\theta)}{\partial(\theta_j)} = \big(y - h_\theta(x)\big) * x_j$$

Parameter updation:

$$\theta_j := \theta_j + \propto * (\, y^i - h_\theta\big(x^i\big) * x_j^i$$

Result :

(a) Before Oversampling:

(i) Confusion matrix

Predicted Class→ Actual Class ↓	**Not a Fraud Transaction**	**Fraud Transactions**
Not a Fraud Transaction	525	2
Fraud Transactions	42	46

(ii) Classification report

Label	**Precision**	**Re-call**	**F1-Score**
0	0.93	1	0.96
1	0.96	0.52	0.68

(b) After Oversampling:

(i) Confusion matrix

Predicted Class→ Actual Class ↓	**Not a Fraud Transaction**	**Fraud Transactions**
Not a Fraud Transaction	474	53
Fraud Transactions	11	77

(ii) Classification report

Label	**Precision**	**Re-call**	**F1-Score**
0	0.98	0.9	0.94
1	0.59	0.88	0.71

5. Final Result :

Classification Method	Accuracy before oversampling	Accuracy after oversampling
Random Forest	**0.98**	**0.9983**
MLNN	**0.99**	**0.86**
Support Vector Machine	**0.91**	**0.83**
Logistic Regression	**0.93**	**0.90**

Classification Method	F1 Score before oversampling	F1 Score after oversampling
Random Forest	**0.98**	**0.99**
MLNN	**0.99**	**0.87**
Support Vector Machine	**0.90**	**0.84**
Logistic Regression	**0.92**	**0.90**

6. Conclusion :

Finally we concluded that random forest is best classifier among all four classifier because best accuracy (0.9983) and F1 score (0.99) is given by Random forest in both cases (i.e.) with and without use of oversampling. In random forest accuracy and F1 score is increased with oversampling but in other Classifiers accuracy and F1 Score is decreased.

We studied about it on various websites and trained models on some other datasets. From that we reached to conclusion that it is case of over fitting models of models, since Oversampling replicates minority class events

in Random forest Oversampling improved accuracy because RandomForest is a technique of Ensemble learning and formed from many weak learner (Decision tree).

But in other Classifiers Oversampling lead to Overfitting of models.

7. Reference

(i) **Dealing with Imbalanced Data**
https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb

(ii) **Lecture of Prof. Sudeshna Sarkar, IIT Kharagpur**
https://www.youtube.com/watch?v=BRMS3T11Cdw&list=PL3pGy4HtqwD2a57wl7Cl7tmfxfk7JWJ9Y

(iii) **Lectures of Andrew NG on Coursera**
https://www.coursera.org/learn/machine-learning/home/welcome

(iv) **Dataset**
https://www.kaggle.com/#

(v) **Conclusion**
https://stats.stackexchange.com/questions/234016/opinions-about-oversampling-in-general-and-the-smote-algorithm-in-particular

https://www.datacamp.com/community/tutorials/diving-deep-imbalanced-data