Machine Learning Engineer Nanodegree

Capstone Project "Customer Churn Prediction for Wireless Providers"

Ufuk Ozturk

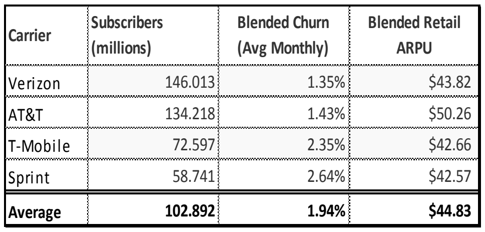
02/16/2018

I. Definition

Project Overview

Customer churn – when subscribers jump from network to network in search of bargains – is one of the biggest challenges confronting a telecom company.

Common causes of churn include high prices, poor service, poor connection quality, new competitors and outdated technology. The churn rate is used as an indicator of the health and loyalty of a company's subscriber base and the lower the churn rate, the better the outlook is for the company.



Carrier	Subscribers (millions)	Blended Churn (Avg Monthly)	Blended Retail ARPU
Verizon	146.013	1.35%	$43.82
AT&T	134.218	1.43%	$50.26
T-Mobile	72.597	2.35%	$42.66
Sprint	58.741	2.64%	$42.57
Average	**102.892**	**1.94%**	**$44.83**

Figure 1. Top 4 x US Wireless Carrier Metrics Q1 2017[1]

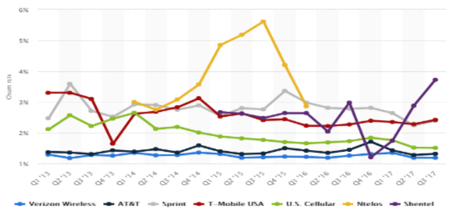


Figure 2. Average monthly churn rate for carriers in the US (2013-17)[2]

The monthly average revenue per user (ARPU) is $45 and the average length of postpaid customer relationships was approximately 51 months in 2011[3]. That means the average monthly loss from the churn is approx. $90M.

Typical wireless industry standard cost of customer acquisition is ranging $315 (Sprint[4]) & $417(Bell & Telus[5]). As per the wireless provider's perspective, the retention of customers is the key, because, the cost of retaining existing customers is much cheaper than the acquisition of new customers.
The primary dataset for this project is provided by sgi.com on University of Tartu website[6].

---

1 https://www.fiercewireless.com/wireless/how-verizon-at-t-t-mobile-sprint-and-more-stacked-up-q1-2017-top-7-carriers

2 https://www.statista.com/statistics/283511/average-monthly-churn-rate-top-wireless-carriers-us/

3 http://money.cnn.com/2012/03/26/technology/cell-phone-customers/index.htm

4 https://www.entrepreneur.com/article/225415

5 http://business.financialpost.com/technology/big-telecoms-are-spending-more-cash-to-keep-customers-but-some-tactics-raise-concerns

6 https://courses.cs.ut.ee/MTAT.03.319/2017_fall/uploads/Main/edw_cdr

Losing customers is not good for any business and identifying unhappy customers early on gives a chance to take the action. Given these challenging telecom industry dynamics, managing the customer base to reduce churn should be among any senior telecom executive's highest priorities[7].

IBM Watson sample dataset[8] is widely used in the churn prediction studies. Matt Dancho from the Business Science approached this problem with R using Keras, LIME and Correlation Analysis[9]. A neural network based customer profiling methodology for churn prediction was presented by John Hadden[10] and over the last decade there has been increasing interest for relevant studies in telecommunication industry.[11] [12] [13]

Problem Statement

Customer churn is a serious problem for all mobile operators. Early identification of customers from the risk group could help retain them in the operator's network. Using machine learning algorithms can help identify which customers are most likely to churn, thus providing the telecom operator with valuable data to turn the situation around.

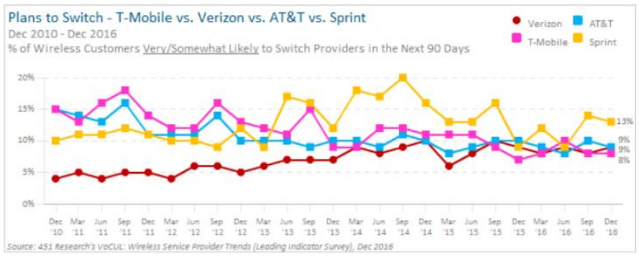


Figure 3. Customers most likely to switch in the next 90 days (survey)[14]

In this project, I'll be using an Artificial Neural Network (ANN) technique to identify the customers who are about to churn. ANN was proposed based on human brain tissue and neural system, and can be used to simulate neural activities of information processing in the human brain. In neural networks each attribute is associated with a weight and combinations of weighted attributes participate in the prediction task.

[7] https://www.mckinsey.com/industries/telecommunications/our-insights/reducing-churn-in-telecom-through-advanced-analytics

[8] https://www.ibm.com/communities/analytics/watson-analytics-blog/guide-to-sample-datasets/

[9] http://www.business-science.io/business/2017/11/28/customer_churn_analysis_keras.html

[10] https://dspace.lib.cranfield.ac.uk/bitstream/handle/1826/3508/Hadden_J_2008.pdf?sequence=3

[11] http://www.academia.edu/26326189/Telecommunication_subscribers_churn_prediction_model_using_machine_learning

[12] https://ijcsi.org/papers/IJCSI-10-2-1-165-172.pdf

[13] https://hrcak.srce.hr/file/92211

[14] http://techblog.comsoc.org/2017/02/23/451-alliance-wireless-service-provider-trends-t-mobile-leads/

Based on the dataset, this method will find patterns which can identify possible churners. The model will calculate the score for each customer with the probability of churn and address the top X ones.

The weights are constantly updated during the learning process. Given a customer dataset and a set of predictor variables the neural network tries to calculate a combination of the inputs and to output the probability that the customer is a churner.

ANN's prediction accuracy is generally high and very robust when training examples contain errors or noisy data.

In that case, the company could analyze the data, prioritize the customers based on their net profitability and target their most valuable customers while factoring in the cost of the incentives.

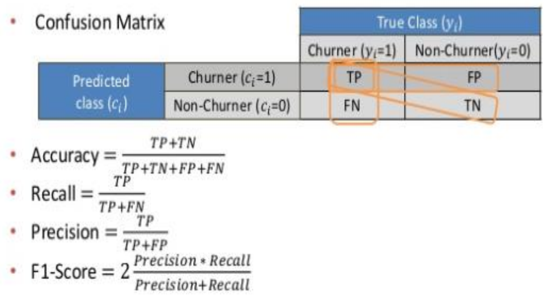
Metrics

The churn prediction is the classification problem which is making a decision on which class the new instance belong to base on the features.

Confusion matrix to describe the performance of a model which is very useful because it will provide the True positive (TP)/False positive (FP)/True negative (TN) and the False negative (FN) values.

Mobile operators prefer models with high **sensitivity** rather than models with high **specificity** because the cost associated with the incorrect classification of churners is higher than the cost associated with the incorrect classification of a non-churner.

A compromise between high sensitivity combined with reasonable specificity should be always made so mobile operators can effectively manage their budget to achieve high customer retention.



- Confusion Matrix

		True Class (y_i)	
		Churner (y_i=1)	Non-Churner(y_i=0)
Predicted class (c_i)	Churner (c_i=1)	TP	FP
	Non-Churner (c_i=0)	FN	TN

- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
- Recall = $\frac{TP}{TP+FN}$
- Precision = $\frac{TP}{TP+FP}$
- F1-Score = $2\frac{Precision * Recall}{Precision+Recall}$

Figure 4. The churn prediction metrics

The metrics I will use to evaluate the model;

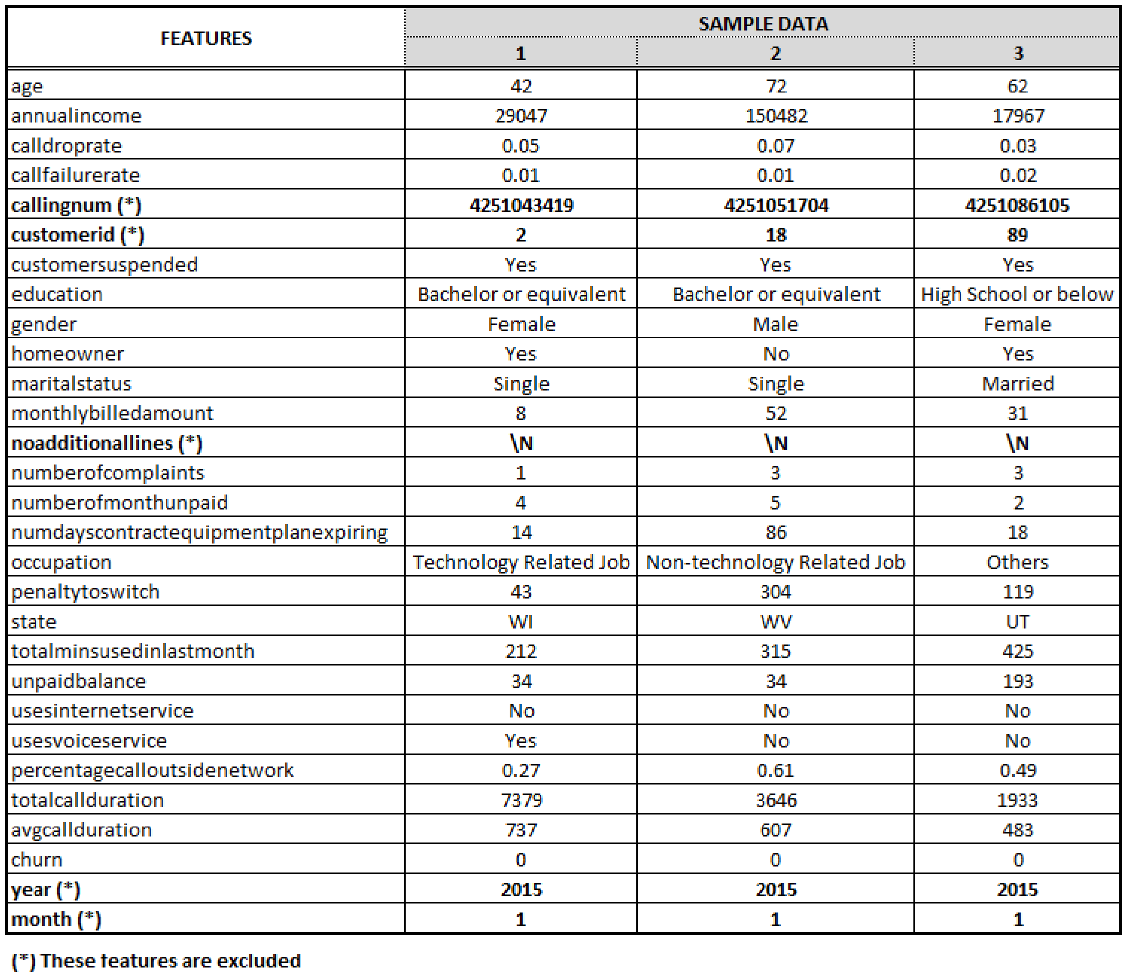
Recall/Sensitivity (True Positive rate): In this churn problem; an important question to ask, when the customer churns, how often does my model predict that correctly?
Precision (Positive prediction value): Another question, when a model predicts the customer will churn, how often does that customer actually churn?
Specificity (True negative rate): corresponds to the proportion of non-churners which are correctly identified.
F1 score: It is the weighted average of precision and recall.
If we predicted that everyone would churn that would give us the sensitivity of 1 and the specificity of 0.
Predicting that no one will churn would give us the sensitivity of 0 and specificity of 1.

II. Analysis

Data Exploration

The dataset is provided by sgi.com on University of Tartu[15] website. It is from the wireless operator's operational data and each row represents a customer's monthly status for 2015 Jan thru Mar. There are 20468 records for the 9525 unique customers and 29 attributes for each customer such as education, annual income, gender, state and the call drop rates.
The data also contains a label "churn" to indicate if the customer churned or stayed with the company. The dataset doesn't include any unusual properties or outliers.

Here are some samples taken from the dataset with their corresponding labels.



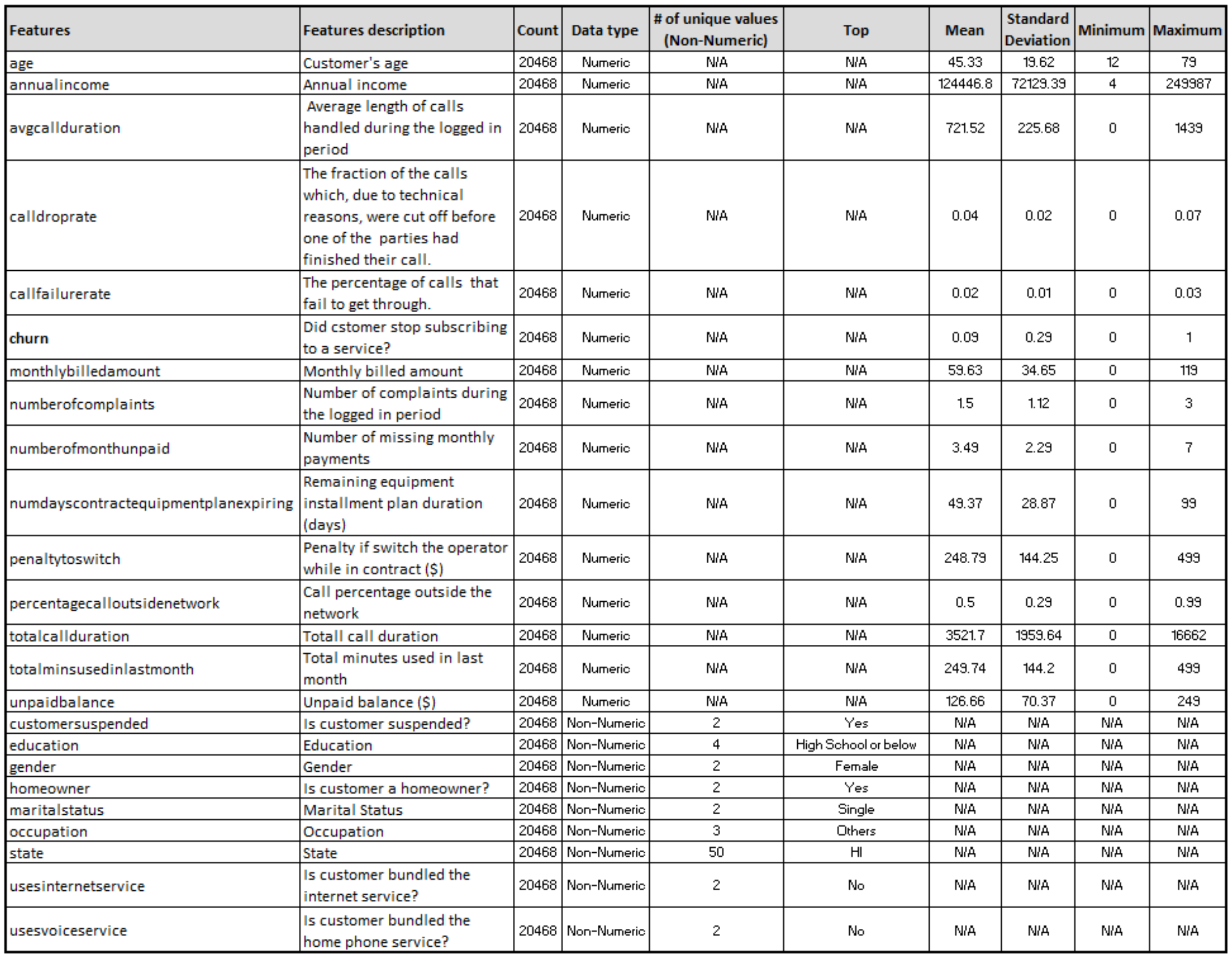
FEATURES	**SAMPLE DATA**		
	1	**2**	**3**
age	42	72	62
annualincome	29047	150482	17967
calldroprate	0.05	0.07	0.03
callfailurerate	0.01	0.01	0.02
callingnum (*)	**4251043419**	**4251051704**	**4251086105**
customerid (*)	**2**	**18**	**89**
customersuspended	Yes	Yes	Yes
education	Bachelor or equivalent	Bachelor or equivalent	High School or below
gender	Female	Male	Female
homeowner	Yes	No	Yes
maritalstatus	Single	Single	Married
monthlybilledamount	8	52	31
noadditionallines (*)	**\N**	**\N**	**\N**
numberofcomplaints	1	3	3
numberofmonthunpaid	4	5	2
numdayscontractequipmentplanexpiring	14	86	18
occupation	Technology Related Job	Non-technology Related Job	Others
penaltytoswitch	43	304	119
state	WI	WV	UT
totalminsusedinlastmonth	212	315	425
unpaidbalance	34	34	193
usesinternetservice	No	No	No
usesvoiceservice	Yes	No	No
percentagecalloutsidenetwork	0.27	0.61	0.49
totalcallduration	7379	3646	1933
avgcallduration	737	607	483
churn	0	0	0
year (*)	**2015**	**2015**	**2015**
month (*)	**1**	**1**	**1**

(*) These features are excluded

Figure 5. The churn dataset features

Simple statistics about the dataset are presented here.

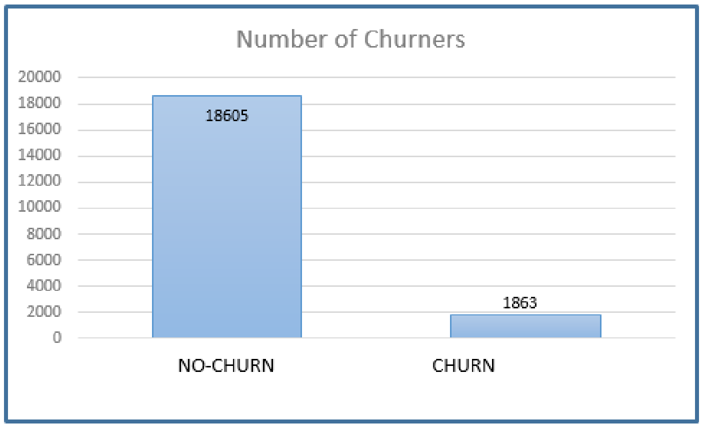
[15] https://courses.cs.ut.ee/MTAT.03.319/2017_fall/uploads/Main/edw_cdr



Features	Features description	Count	Data type	# of unique values (Non-Numeric)	Top	Mean	Standard Deviation	Minimum	Maximum
age	Customer's age	20468	Numeric	N/A	N/A	45.33	19.62	12	79
annualincome	Annual income	20468	Numeric	N/A	N/A	124446.8	72129.39	4	249987
avgcallduration	Average length of calls handled during the logged in period	20468	Numeric	N/A	N/A	721.52	225.68	0	1439
calldroprate	The fraction of the calls which, due to technical reasons, were cut off before one of the parties had finished their call.	20468	Numeric	N/A	N/A	0.04	0.02	0	0.07
callfailurerate	The percentage of calls that fail to get through.	20468	Numeric	N/A	N/A	0.02	0.01	0	0.03
churn	Did cstomer stop subscribing to a service?	20468	Numeric	N/A	N/A	0.09	0.29	0	1
monthlybilledamount	Monthly billed amount	20468	Numeric	N/A	N/A	59.63	34.65	0	119
numberofcomplaints	Number of complaints during the logged in period	20468	Numeric	N/A	N/A	1.5	1.12	0	3
numberofmonthunpaid	Number of missing monthly payments	20468	Numeric	N/A	N/A	3.49	2.29	0	7
numdayscontractequipmentplanexpiring	Remaining equipment installment plan duration (days)	20468	Numeric	N/A	N/A	49.37	28.87	0	99
penaltytoswitch	Penalty if switch the operator while in contract ($)	20468	Numeric	N/A	N/A	248.79	144.25	0	499
percentagecalloutsidenetwork	Call percentage outside the network	20468	Numeric	N/A	N/A	0.5	0.29	0	0.99
totalcallduration	Totall call duration	20468	Numeric	N/A	N/A	3521.7	1959.64	0	16662
totalminsusedinlastmonth	Total minutes used in last month	20468	Numeric	N/A	N/A	249.74	144.2	0	499
unpaidbalance	Unpaid balance ($)	20468	Numeric	N/A	N/A	126.66	70.37	0	249
customersuspended	Is customer suspended?	20468	Non-Numeric	2	Yes	N/A	N/A	N/A	N/A
education	Education	20468	Non-Numeric	4	High School or below	N/A	N/A	N/A	N/A
gender	Gender	20468	Non-Numeric	2	Female	N/A	N/A	N/A	N/A
homeowner	Is customer a homeowner?	20468	Non-Numeric	2	Yes	N/A	N/A	N/A	N/A
maritalstatus	Marital Status	20468	Non-Numeric	2	Single	N/A	N/A	N/A	N/A
occupation	Occupation	20468	Non-Numeric	3	Others	N/A	N/A	N/A	N/A
state	State	20468	Non-Numeric	50	HI	N/A	N/A	N/A	N/A
usesinternetservice	Is customer bundled the internet service?	20468	Non-Numeric	2	No	N/A	N/A	N/A	N/A
usesvoiceservice	Is customer bundled the home phone service?	20468	Non-Numeric	2	No	N/A	N/A	N/A	N/A

Figure 6. The churn dataset statistics

The number of instances in each class;



These five features has been excluded from the analysis which they shouldn't affect churn;

- Customer's phone number and unique ID ('callingnum' & 'customerid')
- The date ('year' & 'month')
- The number of additional lines "empty column" ('noadditionallines')

Exploratory Visualization

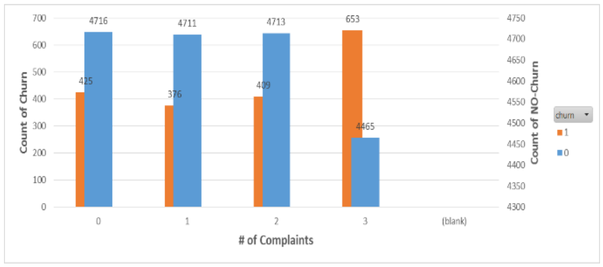
Nearly one-fifth of U.S. postpaid subscribers are considering switching carriers, according to data from Jefferies Equity Research Americas (2017-survey). And their primary concerns are price and coverage[16]. In our dataset, there are number of features related with both concerns.

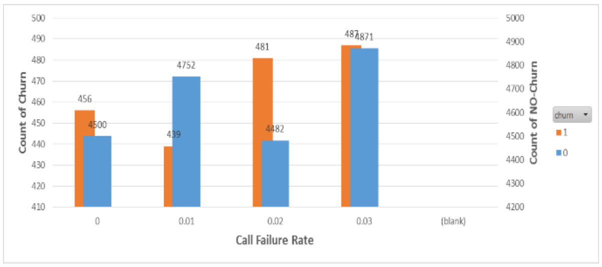
The charts below shows the cost and coverage related features vs churn rates.

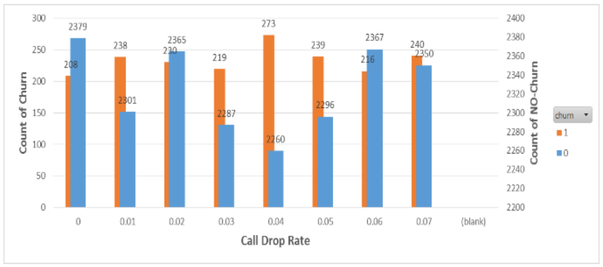
Poor Customer experience related features vs Churn rate;

- *Number of complaints during the logged in period (numberofcomplaints)*
- *The percentage of calls that fail to get through (callfailurerate)*
- *The fraction of the calls which, due to technical reasons, were cut off before one of the parties had finished their call (calldroprate)*

** Please note that the orange (1) represents 'churned' customers and the blue (0) represents non-churners.





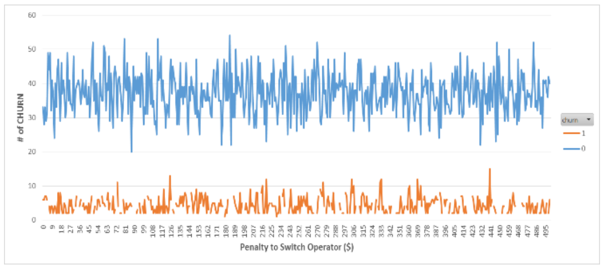


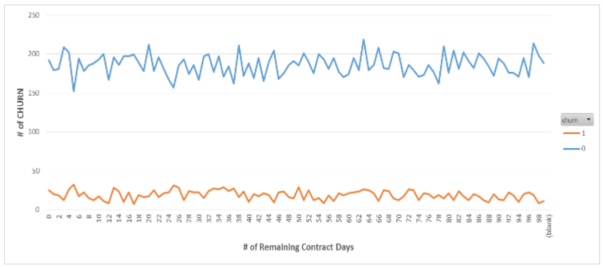
16 https://www.fiercewireless.com/wireless/nearly-one-fifth-u-s-postpaid-subs-considering-switching-jefferies

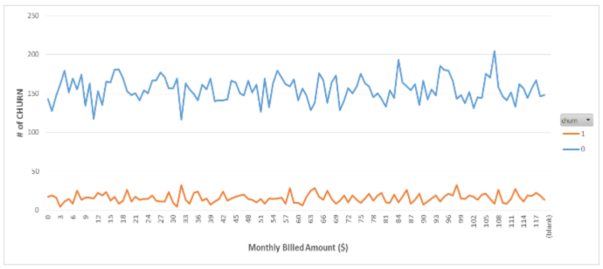
Cost related features vs Churn rate;

- *Penalty amount if switch the operator while in contract (penaltytoswitch)*
- *Remaining equipment installment plan duration-days (numdayscontractequipmentplanexpiring)*
- *Monthly billed amount (monthlybilledamount)*

** Please note that the orange (1) represents 'churned' customers and the blue (0) represents non-churners.







As we can see from these charts, none of these features can explain the churn problem alone:

There is no difference on the churn rate between a lowest paying customer and a highest paying customer "Monthly Billed Amount vs. churn rate". If we look at the penalty to switch operator feature the churn rate is very close for both $400+ and $10 penalty. The number of remaining contract days feature shows the similar behavior. The churner vs. non-churner ratio is almost the same for the lowest and highest call drop and failure rates.

Actually the "Number of Complaints" is the only feature which shows an increase on churn rate. It looks like 1 or 2 interactions with customer support is relatively ok but the customers are 50% more likely to be a "switch-risk" when they have to contact customer support more than twice in a month.

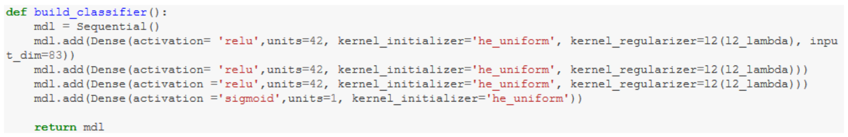
Algorithms and Techniques

I'll be using a Backpropagation based feedforward Artificial Neural Network to predict customer churn.

In the two-class classification problems, as in churn prediction, our goal is to assign previously unseen patterns to their respective classes (churn / no-churn) based on previous data from each class. Any labels that we can generate which correlate to data, can be used to train a neural network. After the algorithm has found the best pattern it can, it will use that pattern to make predictions for unlabeled testing data.

Neural networks are a class of powerful machine learning algorithms. They are based on solid statistical foundations and have been applied successfully in classification problems.

I will import the Sequential model to initialize NN and dense module to add Hidden Layers.



```
def build_classifier():
    mdl = Sequential()
    mdl.add(Dense(activation= 'relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda), inpu
t_dim=83))
    mdl.add(Dense(activation= 'relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda)))
    mdl.add(Dense(activation ='relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda)))
    mdl.add(Dense(activation ='sigmoid',units=1, kernel_initializer='he_uniform'))

    return mdl
```

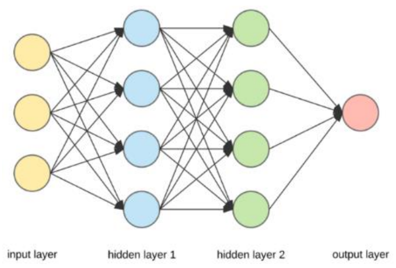


Figure 7. Neural Network

The Input Layer

The shape of the dataset will be 20468 row x 83 column (feature) after the cleaning and encoding the dataset.

The number of neurons in that layer is equal to the number of features (83) in the dataset.

Activation Functions

The **'sigmoid'** activation function and one output neuron is selected for this binary classification (2 class) problem for the **output layer**. The sigmoid takes a number and squashes it into a range between 0 and 1 and it is used in the output layer where our end goal is to predict probability. It converts large negative numbers to 0 and large positive numbers to 1.

The '**ReLu**' (Rectified Linear units) activation function selected for the **hidden layers**. This means that when the input x < 0 the output is 0 and if x > 0 the output is x. This activation makes the network converge much faster and its major benefit is the reduced likelihood of the gradient to vanish[17]. But ReLu tends to blow up activation (there is no mechanism to constrain the output of the neuron) and it should be used within Hidden layers of a Neural Network Model.

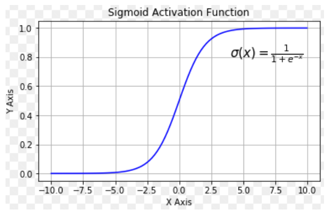


Figure 8. "Sigmoid" function –output layer

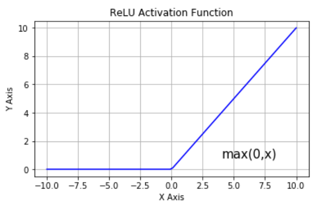


Figure 9. "RELU" function –input and hidden layers

The Hidden Layers

Neural networks with **two hidden layers** can represent functions with any kind of shape[18]. There are many rule-of-thumb methods for determining the correct **number of neurons to use in the hidden layers**, such as the following:

[17] https://stats.stackexchange.com/questions/126238/what-are-the-advantages-of-relu-over-sigmoid-function-in-deep-neural-networks

[18] https://web.archive.org/web/20140721050413/http://www.heatonresearch.com/node/707

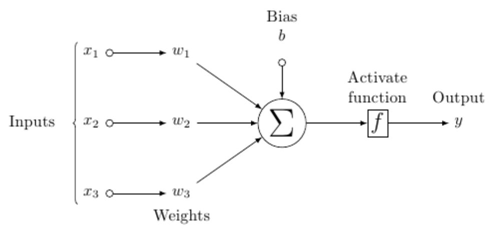


Figure 10. Neural Network diagram

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer units

42 hidden neurons has been selected by trial and error: (input layer + output layer) /2: $\frac{83+1}{2} = 42$

Network Initialization (kernel_initializer)

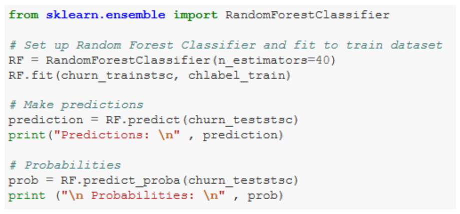
This parameter is used for assigning initial weights to the various layers within the network. Initializing the weights in an appropriate way can be significantly influential to how easily the network learns from the training set. The key idea behind the **'he_uniform' initialization** scheme is to make it easier for a signal to pass through the layer during forward as well as backward propagation, for a linear activation (*this also works nicely for sigmoid activations because the interval where they are unsaturated is roughly linear as well*). This initialization more suitable for ReLU activations, compensating for the fact that this activation is zero for half of the possible input space[19].

L2 regularization (kernel_regularizer)

There are several regularizes that we can apply to our networks to keep overfitting in check. Arguably the most popular out of them is L2 regularization (sometimes also called weight decay), which takes a more direct approach than dropout for regularizing. A regularize, in this sense, aims to decrease complexity of the model while maintaining parameter count the same. L2 regularization does so by penalizing weights with large magnitudes, by minimizing their L2 norm, using a hyperparameter λ to specify the relative importance of minimizing the norm to minimizing the loss on the training set.

Benchmark

Random Forest Classifier has been used to compare the performance of artificial neural network for the churn prediction project.

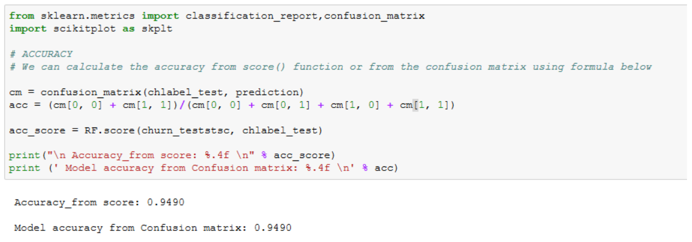

```
from sklearn.ensemble import RandomForestClassifier

# Set up Random Forest Classifier and fit to train dataset
RF = RandomForestClassifier(n_estimators=40)
RF.fit(churn_trainstsc, chlabel_train)

# Make predictions
prediction = RF.predict(churn_teststsc)
print("Predictions: \n" , prediction)

# Probabilities
prob = RF.predict_proba(churn_teststsc)
print ("\n Probabilities: \n" , prob)
```

Evaluation "Benchmark Model"

The same metrics used to evaluate both Neural Network and Random Forest Classifier models:

- "Accuracy" & "Confusion matrix" & "Recall/Sensitivity" & "Precision" & "F1 score"

[19] https://cambridgespark.com/content/tutorials/neural-networks-tuning-techniques/index.html



```
from sklearn.metrics import classification_report,confusion_matrix
import scikitplot as skplt

# ACCURACY
# We can calculate the accuracy from score() function or from the confusion matrix using formula below

cm = confusion_matrix(chlabel_test, prediction)
acc = (cm[0, 0] + cm[1, 1])/(cm[0, 0] + cm[0, 1] + cm[1, 0] + cm[1, 1])

acc_score = RF.score(churn_teststsc, chlabel_test)

print("\n Accuracy_from score: %.4f \n" % acc_score)
print (' Model accuracy from Confusion matrix: %.4f \n' % acc)
```

```
 Accuracy_from score: 0.9490

 Model accuracy from Confusion matrix: 0.9490
```

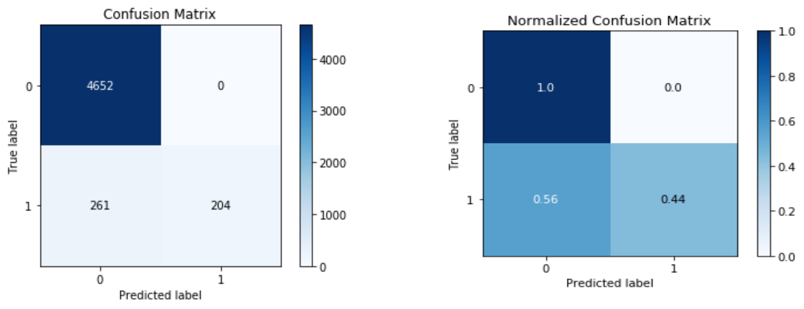
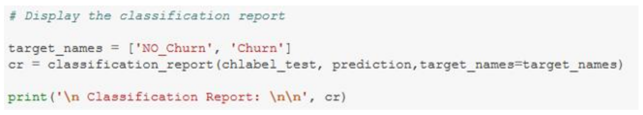
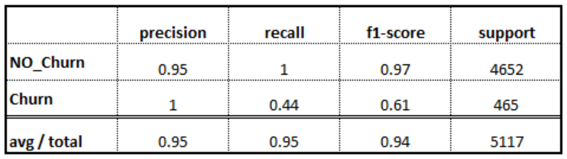


Figure 11. Random Forest Classifier model - Confusion matrix



```
# Display the classification report

target_names = ['NO_Churn', 'Churn']
cr = classification_report(chlabel_test, prediction,target_names=target_names)

print('\n Classification Report: \n\n', cr)
```



	precision	recall	f1-score	support
NO_Churn	0.95	1	0.97	4652
Churn	1	0.44	0.61	465
avg / total	0.95	0.95	0.94	5117

Figure 12. Random Forest Classifier model – Classification report

III. Methodology

Data Preprocessing

- **Data Cleaning**

These five features has been excluded from the analysis which they shouldn't affect churn;

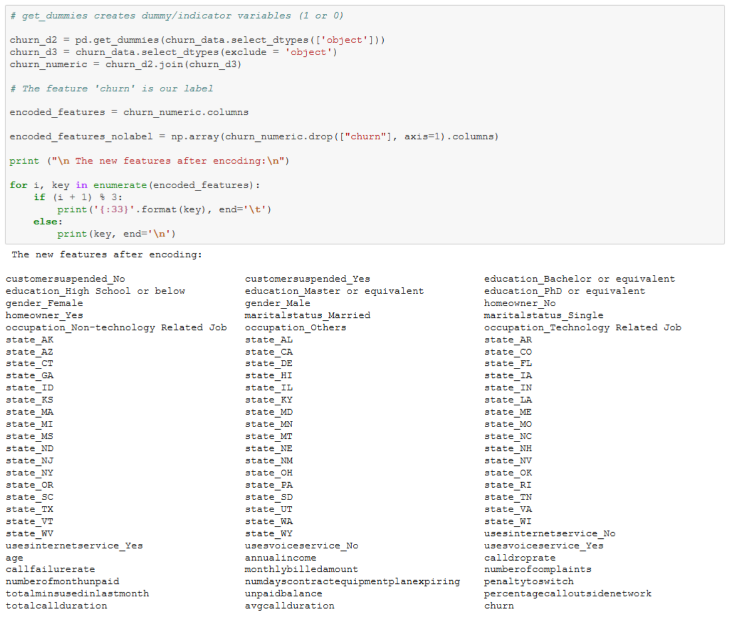
- Customer's phone number and unique ID ('callingnum' & 'customerid')
- The date ('year' & 'month')
- The number of additional lines "empty column" ('noadditionallines')

- **One Hot Encoding**

The 'churn' dataset contains 'object' type data as well as numeric data. The non-numeric data need to be encoded, as in mathematical equations we need to use only numerical values. That is, non-numerical 'object' type data transformed to numerical labels.

These are the object type features in this dataset;
- *"customersuspended" & "education" & "gender" & "homeowner" & "maritalstatus" & "occupation" & "state" & "usesinternetservice" & "usesvoiceservice"*

Pandas' get_dummies function has been used for encoding.



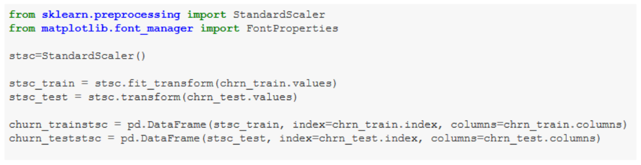
```
# get_dummies creates dummy/indicator variables (1 or 0)

churn_d2 = pd.get_dummies(churn_data.select_dtypes(['object']))
churn_d3 = churn_data.select_dtypes(exclude = 'object')
churn_numeric = churn_d2.join(churn_d3)

# The feature 'churn' is our label

encoded_features = churn_numeric.columns

encoded_features_nolabel = np.array(churn_numeric.drop(["churn"], axis=1).columns)

print ("\n The new features after encoding:\n")

for i, key in enumerate(encoded_features):
    if (i + 1) % 3:
        print('{:33}'.format(key), end='\t')
    else:
        print(key, end='\n')
```

```
 The new features after encoding:

customersuspended_No                  customersuspended_Yes                 education_Bachelor or equivalent
education_High School or below        education_Master or equivalent        education_PhD or equivalent
gender_Female                         gender_Male                           homeowner_No
homeowner_Yes                         maritalstatus_Married                 maritalstatus_Single
occupation_Non-technology Related Job occupation_Others                     occupation_Technology Related Job
state_AK                              state_AL                              state_AR
state_AZ                              state_CA                              state_CO
state_CT                              state_DE                              state_FL
state_GA                              state_HI                              state_IA
state_ID                              state_IL                              state_IN
state_KS                              state_KY                              state_LA
state_MA                              state_MD                              state_ME
state_MI                              state_MN                              state_MO
state_MS                              state_MT                              state_NC
state_ND                              state_NE                              state_NH
state_NJ                              state_NM                              state_NV
state_NY                              state_OH                              state_OK
state_OR                              state_PA                              state_RI
state_SC                              state_SD                              state_TN
state_TX                              state_UT                              state_VA
state_VT                              state_WA                              state_WI
state_WV                              state_WY                              usesinternetservice_No
usesinternetservice_Yes               usesvoiceservice_No                   usesvoiceservice_Yes
age                                   annualincome                          calldroprate
callfailurerate                       monthlybilledamount                   numberofcomplaints
numberofmonthunpaid                   numdayscontractequipmentplanexpiring  penaltytoswitch
totalminsusedinlastmonth              unpaidbalance                         percentagecalloutsidenetwork
totalcallduration                     avgcallduration                       churn
```

- **Feature Scaling**

Feature scaling is used to avoid dominance of one independent variable on others. Different from human's brain, when machine learning algorithms do the classification, all the features are expressed and calculated by the same coordinate system.

So when these algorithms' input is unscaled features, large scale data has more influence on the weight.



```
from sklearn.preprocessing import StandardScaler
from matplotlib.font_manager import FontProperties

stsc=StandardScaler()

stsc_train = stsc.fit_transform(chrn_train.values)
stsc_test = stsc.transform(chrn_test.values)

churn_trainstsc = pd.DataFrame(stsc_train, index=chrn_train.index, columns=chrn_train.columns)
churn_teststsc = pd.DataFrame(stsc_test, index=chrn_test.index, columns=chrn_test.columns)
```

In this project the "**StandardScaler**" method of feature scaling that are implemented.

The StandardScaler assumes the data is normally distributed within each feature and will scale them such that the distribution is now centered on zero, with a standard deviation of 1 when we are comparing measurements that have different units.

The mean and standard deviation are calculated for the feature and then the feature is scaled based on:

$$\frac{x_i - mean(x)}{stdev(x)}$$

StandardScaler that implements the Transformer API to compute the mean and standard deviation on a training set so as to be able to reapply the same transformation on the testing set.

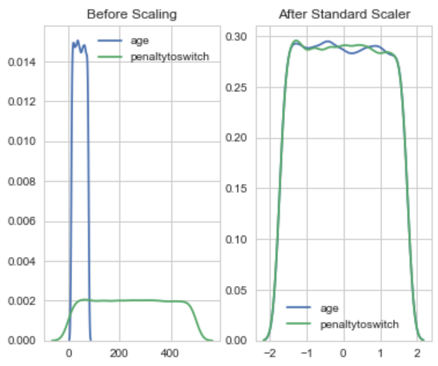
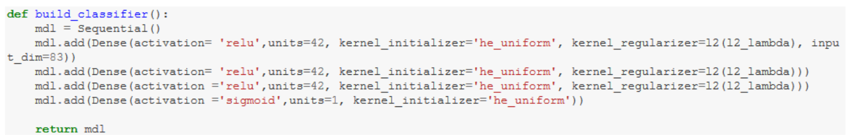


Figure 13. Feature Scaling – StandardScaler

Implementation

The Artificial Neural Network model parameters has been described in the **Algorithms and Techniques** section.

- The ‘*ReLu*’ (Rectified Linear units) activation function selected for the hidden layers.
- The ‘*sigmoid*’ activation function and one output neuron is selected for this binary classification (2 class) problem for the output layer.
- The ‘*he_uniform*’ kernel_initializer is used for assigning initial weights to the various layers within the network scheme.
- The ‘*L2 regularization’* (0.0001 factor) applied to our networks to keep overfitting in check.

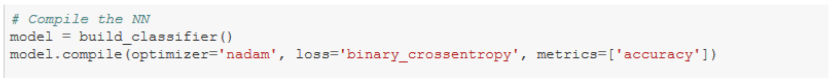


```
def build_classifier():
    mdl = Sequential()
    mdl.add(Dense(activation= 'relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda), inpu
t_dim=83))
    mdl.add(Dense(activation= 'relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda)))
    mdl.add(Dense(activation ='relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda)))
    mdl.add(Dense(activation ='sigmoid',units=1, kernel_initializer='he_uniform'))

    return mdl
```

Compiling the Model

Optimizer

After initialized the weights with “he-uniform” initializer then the optimizer algorithm has been used to find the optimal set of weights.



```
# Compile the NN
model = build_classifier()
model.compile(optimizer='nadam', loss='binary_crossentropy', metrics=['accuracy'])
```

The "NADAM" optimizer has been selected for this project based on "Tuning the ANN" between Rmsprop & Adam and Nadam.

RMSprop contributes the exponentially decaying average of past squared gradients, while momentum accounts for the exponentially decaying average of past gradients.

Adam is similar to RMSprop with momentum. And the Nesterov accelerated gradient (NAG) is superior to vanilla momentum. Nadam (Nesterov-accelerated Adaptive Moment Estimation) thus combines Adam and NAG instead of classical momentum[20].

Finally, binary_crossentropy has been used as the logarithmic loss function during training. This is the preferred loss function for binary classification problems. The accuracy metrics will be collected when the model is trained.

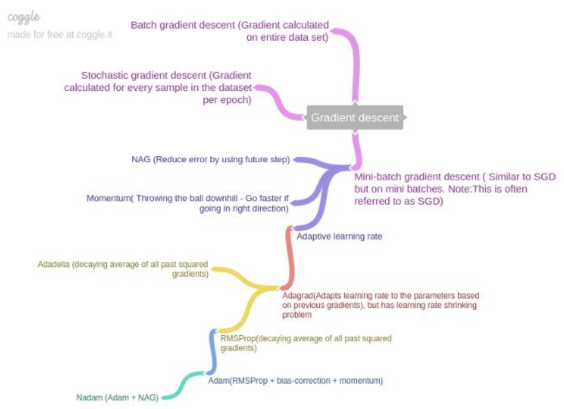
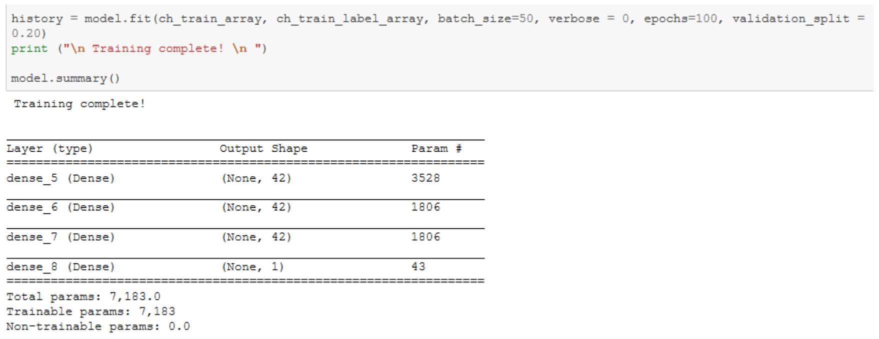

coggle
made for free at coggle.it
Batch gradient descent (Gradient calculated on entire data set)
Stochastic gradient descent (Gradient calculated for every sample in the dataset per epoch)
Gradient descent
NAG (Reduce error by using future step)
Mini-batch gradient descent (Similar to SGD but on mini batches. Note:This is often referred to as SGD)
Momentum(Throwing the ball downhill - Go faster if going in right direction)
Adaptive learning rate
Adadelta (decaying average of all past squared gradients)
Adagrad(Adapts learning rate to the parameters based on previous gradients), but has learning rate shrinking problem
RMSProp(decaying average of all past squared gradients)
Adam(RMSProp + bias-correction + momentum)
Nadam (Adam + NAG)

Figure 14. Optimizer map[21]

Fitting the Model

The 'Automatic Verification Dataset' has been used by setting the validation split argument. With this parameter, a portion (20%) of the training data separated into a validation dataset and evaluate the performance of the model on that validation dataset each epoch.

The "epochs" set to 100 & the "batch size" value set to 50. These values has been selected after tuning the model.



```
history = model.fit(ch_train_array, ch_train_label_array, batch_size=50, verbose = 0, epochs=100, validation_split =
0.20)
print ("\n Training complete! \n ")

model.summary()
```

```
 Training complete!

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_5 (Dense)              (None, 42)                3528      
_________________________________________________________________
dense_6 (Dense)              (None, 42)                1806      
_________________________________________________________________
dense_7 (Dense)              (None, 42)                1806      
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 43        
=================================================================
Total params: 7,183.0
Trainable params: 7,183
Non-trainable params: 0.0
```

20 http://ruder.io/optimizing-gradient-descent/

21 http://forums.fast.ai/t/how-do-we-decide-the-optimizer-used-for-training/1829/6

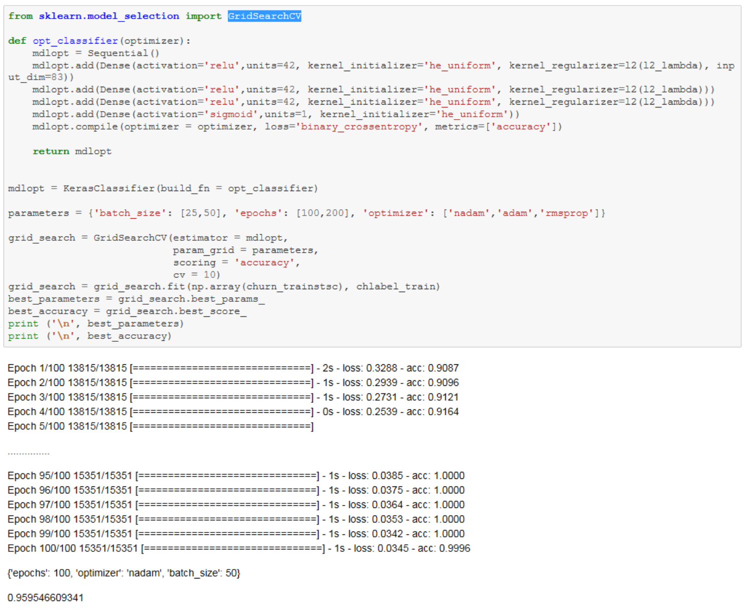
Refinement

Tuning the ANN

GridSearchCV with 10- fold cross validation has been used to methodically train and evaluate the model for each and every combination of parameters below to select the 'best parameters' maximizing the cross-validation score.

'batch_size':	[25, 50]
'epochs':	[100,200]
'optimizer':	['nadam','adam','rmsprop']

This method is computationally expensive and takes around 12 hour on my machine.



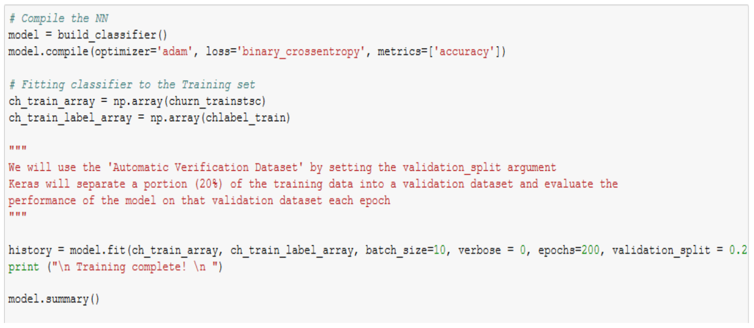
```
from sklearn.model_selection import GridSearchCV

def opt_classifier(optimizer):
    mdlopt = Sequential()
    mdlopt.add(Dense(activation='relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda), inp
ut_dim=83))
    mdlopt.add(Dense(activation='relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda)))
    mdlopt.add(Dense(activation='relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda)))
    mdlopt.add(Dense(activation='sigmoid',units=1, kernel_initializer='he_uniform'))
    mdlopt.compile(optimizer = optimizer, loss='binary_crossentropy', metrics=['accuracy'])

    return mdlopt


mdlopt = KerasClassifier(build_fn = opt_classifier)

parameters = {'batch_size': [25,50], 'epochs': [100,200], 'optimizer': ['nadam','adam','rmsprop']}

grid_search = GridSearchCV(estimator = mdlopt,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10)
grid_search = grid_search.fit(np.array(churn_trainstsc), chlabel_train)
best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_
print ('\n', best_parameters)
print ('\n', best_accuracy)
```

```
Epoch 1/100 13815/13815 [==============================] - 2s - loss: 0.3288 - acc: 0.9087
Epoch 2/100 13815/13815 [==============================] - 1s - loss: 0.2939 - acc: 0.9096
Epoch 3/100 13815/13815 [==============================] - 1s - loss: 0.2731 - acc: 0.9121
Epoch 4/100 13815/13815 [==============================] - 0s - loss: 0.2539 - acc: 0.9164
Epoch 5/100 13815/13815 [==============================]

...............

Epoch 95/100 15351/15351 [==============================] - 1s - loss: 0.0385 - acc: 1.0000
Epoch 96/100 15351/15351 [==============================] - 1s - loss: 0.0375 - acc: 1.0000
Epoch 97/100 15351/15351 [==============================] - 1s - loss: 0.0364 - acc: 1.0000
Epoch 98/100 15351/15351 [==============================] - 1s - loss: 0.0353 - acc: 1.0000
Epoch 99/100 15351/15351 [==============================] - 1s - loss: 0.0342 - acc: 1.0000
Epoch 100/100 15351/15351 [==============================] - 1s - loss: 0.0345 - acc: 0.9996

{'epochs': 100, 'optimizer': 'nadam', 'batch_size': 50}

0.959546609341
```

The initial parameter selections changed after the tuning.

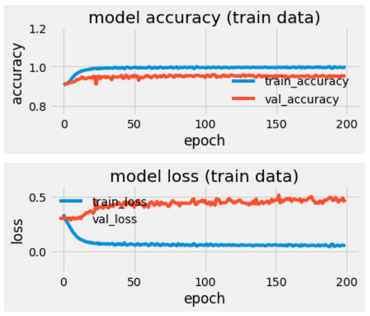
<u>**initial parameters:**</u>

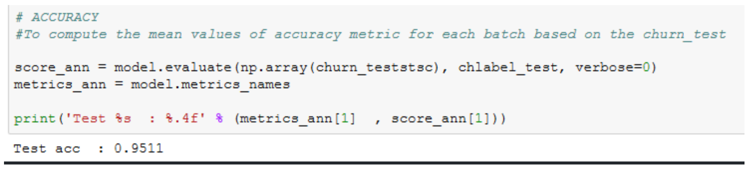
batch_size:	10
epochs:	200
optimizer:	adam



```
# Compile the NN
model = build_classifier()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Fitting classifier to the Training set
ch_train_array = np.array(churn_trainstsc)
ch_train_label_array = np.array(chlabel_train)

"""
We will use the 'Automatic Verification Dataset' by setting the validation_split argument
Keras will separate a portion (20%) of the training data into a validation dataset and evaluate the
performance of the model on that validation dataset each epoch
"""

history = model.fit(ch_train_array, ch_train_label_array, batch_size=10, verbose = 0, epochs=200, validation_split = 0.2
print ("\n Training complete! \n ")

model.summary()
```

BEFORE TUNING



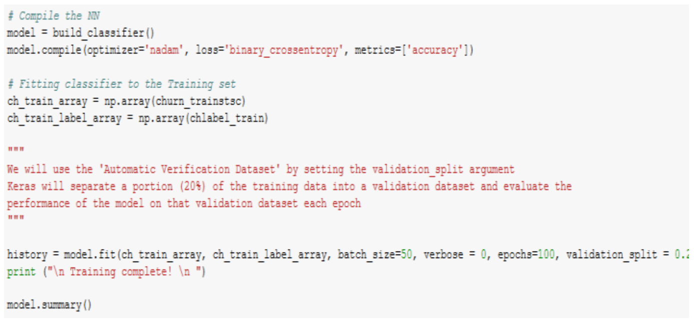


```
# ACCURACY
#To compute the mean values of accuracy metric for each batch based on the churn_test

score_ann = model.evaluate(np.array(churn_teststsc), chlabel_test, verbose=0)
metrics_ann = model.metrics_names

print('Test %s  : %.4f' % (metrics_ann[1]  , score_ann[1]))
```

```
Test acc  : 0.9511
```

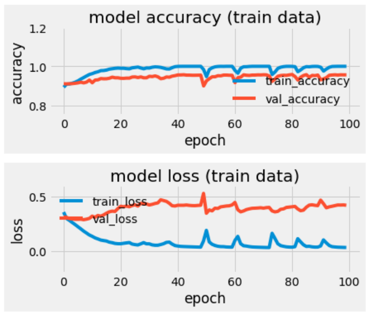
best parameters:

batch_size:	50
epochs:	100
optimizer:	nadam



```
# Compile the NN
model = build_classifier()
model.compile(optimizer='nadam', loss='binary_crossentropy', metrics=['accuracy'])

# Fitting classifier to the Training set
ch_train_array = np.array(churn_trainstsc)
ch_train_label_array = np.array(chlabel_train)

"""
We will use the 'Automatic Verification Dataset' by setting the validation_split argument
Keras will separate a portion (20%) of the training data into a validation dataset and evaluate the
performance of the model on that validation dataset each epoch
"""

history = model.fit(ch_train_array, ch_train_label_array, batch_size=50, verbose = 0, epochs=100, validation_split = 0.2
print ("\n Training complete! \n ")

model.summary()
```

AFTER TUNING



IV. Results

Model Evaluation and Validation

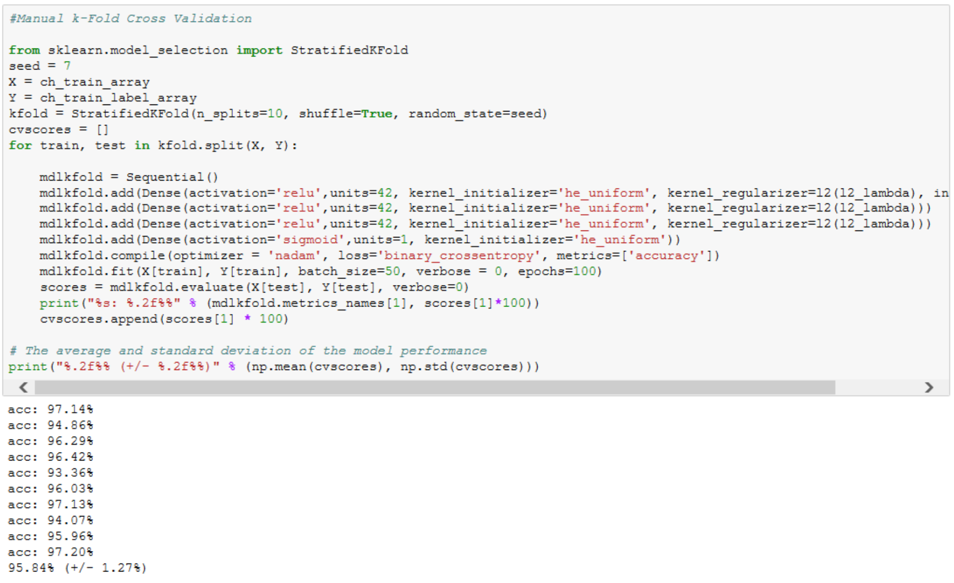
Manual k-Fold Cross Validation

The k-fold cross validation[22] provides a robust estimate of the performance of a model.

It does this by splitting the training dataset into k subsets and takes turns training models on all subsets except one which is held out, and evaluating model performance on the held out validation dataset.

The process is repeated until all subsets are given an opportunity to be the held out validation set.

The performance measure is then averaged across all models that are created.



```
#Manual k-Fold Cross Validation

from sklearn.model_selection import StratifiedKFold
seed = 7
X = ch_train_array
Y = ch_train_label_array
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
cvscores = []
for train, test in kfold.split(X, Y):

    mdlkfold = Sequential()
    mdlkfold.add(Dense(activation='relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda), in
    mdlkfold.add(Dense(activation='relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda)))
    mdlkfold.add(Dense(activation='relu',units=42, kernel_initializer='he_uniform', kernel_regularizer=l2(l2_lambda)))
    mdlkfold.add(Dense(activation='sigmoid',units=1, kernel_initializer='he_uniform'))
    mdlkfold.compile(optimizer = 'nadam', loss='binary_crossentropy', metrics=['accuracy'])
    mdlkfold.fit(X[train], Y[train], batch_size=50, verbose = 0, epochs=100)
    scores = mdlkfold.evaluate(X[test], Y[test], verbose=0)
    print("%s: %.2f%%" % (mdlkfold.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)

# The average and standard deviation of the model performance
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

```
acc: 97.14%
acc: 94.86%
acc: 96.29%
acc: 96.42%
acc: 93.36%
acc: 96.03%
acc: 97.13%
acc: 94.07%
acc: 95.96%
acc: 97.20%
95.84% (+/- 1.27%)
```

22 https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/

The 10-fold StratifiedKFold (*the algorithm attempts to balance the number of instances of each class in each fold*) has been used to evaluate the churn model.

StratifiedKFold created and evaluated 10 models using the 10 splits of the data and the next table shows all of the models' performance scores. The average and standard deviation of the model performance is also shown to provide a robust estimate of model accuracy.

Fold #	Accuracy %
1	97.14%
2	94.86%
3	96.29%
4	96.42%
5	93.36%
6	96.03%
7	97.13%
8	94.07%
9	95.96%
10	97.20%
Average	95.84%
Standard Deviation	+/- 1.27%

Metrics

These metrics has been used to evaluate the model;

- "Confusion matrix" to describe the performance of a model which is very useful because it will provide the True positive (TP)/False positive (FP)/True negative (TN) and the False negative (FN) values.
  The x-axis: If a customer churned or not.
  The y-axis: If the classifier said a customer would churned or not.

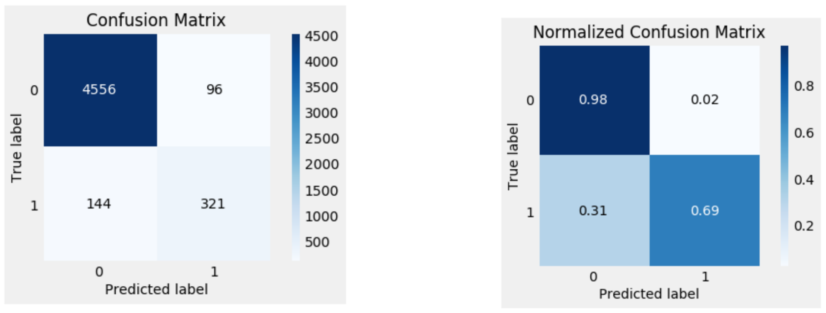
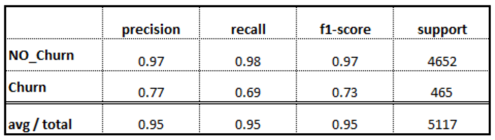


Figure 15. Neural Network model - Confusion matrix

- "Recall/Sensitivity": In this churn problem; an important question to ask, when the customer churns, how often does my model predict that correctly?
- "Precision": Another question, when a model predicts the customer will churn, how often does that customer actually churn?
- "F1 score": It is the weighted average of precision and recall.



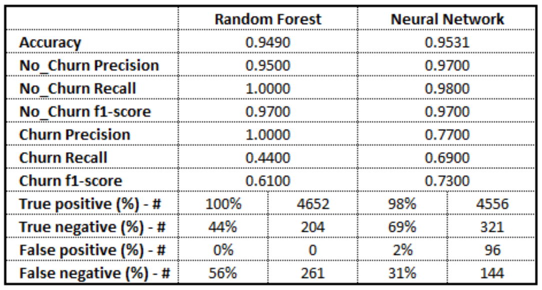
	precision	recall	f1-score	support
NO_Churn	0.97	0.98	0.97	4652
Churn	0.77	0.69	0.73	465
avg / total	0.95	0.95	0.95	5117

Figure 16. Neural Network model – Classification report

Justification

In this project, I tried to provide a solution to overcome the churn problem for the wireless operator. The results conducted in this project show that the neural network model (ANN) outperforms the Random Forest (RF) model in most of the metrics.

The performance comparison of artificial neural network with random forest classifier for the churn prediction problem.



	Random Forest		Neural Network	
Accuracy	0.9490		0.9531	
No_Churn Precision	0.9500		0.9700	
No_Churn Recall	1.0000		0.9800	
No_Churn f1-score	0.9700		0.9700	
Churn Precision	1.0000		0.7700	
Churn Recall	0.4400		0.6900	
Churn f1-score	0.6100		0.7300	
True positive (%) - #	100%	4652	98%	4556
True negative (%) - #	44%	204	69%	321
False positive (%) - #	0%	0	2%	96
False negative (%) - #	56%	261	31%	144

As per the wireless provider's perspective, acquiring new customer is far more costly than keeping a customer. In this churn problem, the most important question to ask; when the customer churns, how often does the model predict that correctly?

The RF model correctly predicts 44% of the churned customers. The NN's score is 69%. If we'd use the NN model, the company could take an action to keep these additional 25% "unhappy" customers before they can decide to leave the company. According to the survey in 2013[23] the typical industry standard cost of customer acquisition is $315 for wireless companies. Actually this number is way more than that right now. This is the cost of false negatives.

On the other hand, NN model's False Positive rate is 2 percent but the rate of the Random Forest is 0. The false positive means that the customer is happy, but the model mistakenly predicted churn. That is, ANN model incorrectly predicted 96 more customers than the RF model. A quick google search shows that the average retention incentive cost is around 100$.

We can formulize the cost function

$$COST = 315\$ * FN + 100\$ * (FP + TN) \quad (*)$$

[23] https://www.entrepreneur.com/article/225415

Note: *True Positive ignored because these are correctly identified "happy" customers*

If we apply this formula to the RF and ANN Confusion matrix values then the cost for both models in this project;

Random Forest Cost:

$COST_{RF} = 315\$ * 261 + 100\$ * (0 + 204) = \ 102.615\$$ (*)

NN Cost:

$COST_{NN} = 315\$ * 144 + 100\$ * (96 + 321) = \ 87.060\$$ (*)

We can also think that the spending 100$ (False Positive) for a customer who is not 100% loyal is not completely wasting money. If we also adjust the cost of acquiring a new customer to recent numbers then the difference between these 2 models would be way more than 15%.

V. Conclusion

Feature Importance

Feature importance measure is used to evaluate the importance of each feature where higher score means more important. It is very important knowing which features in a model are influencing its effectiveness. This is also very valuable to improve our model's actionability.

If we know which customer is predicted to churn then we can act before the customer decides to leave.

But if we understand why our customers are likely to churn –which features affect churn decision mostly- then we can act according to this feature's nature before it occurs.

We can plot feature importance to see which features were most useful in our model. RF automatically provide estimates of feature importance from a trained predictive model[24]

When training a tree, it can be computed how much each feature decreases the weighted impurity in a tree. For a forest, the impurity decrease from each feature can be averaged and the features are ranked according to this measure.

[24] http://blog.datadive.net/selecting-good-features-part-iii-random-forests/

```
# Random Forest Model_feature Importance

fig = plt.figure(figsize=(15, 15))
ax = fig.add_subplot(111)

churn_feat = pd.DataFrame(RF.feature_importances_, columns=["feat_importance"])
churn_feat["labels"] = encoded_features_nolabel

churn_feat.sort_values("feat_importance", inplace=True, ascending=False)
display(churn_feat.head(10))

# Plot the feature importance for all features

bar_width=0.5
index = np.arange(len(RF.feature_importances_))
rects = plt.barh(index , churn_feat["feat_importance"], bar_width, alpha=0.4, color='b', label='Main')
plt.yticks(index, churn_feat["labels"])
plt.title('RF Classifier Feature Importance')
plt.show()
```

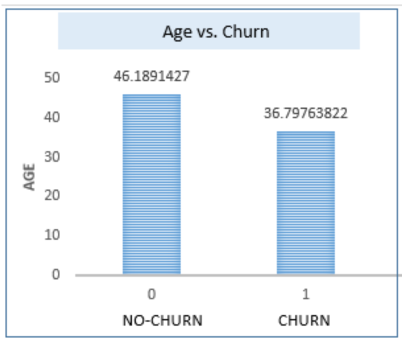
	feat_importance	labels
69	0.08	age
79	0.07	unpaidbalance
70	0.07	annualincome
78	0.06	totalminsusedinlastmonth
73	0.06	monthlybilledamount
77	0.06	penaltytoswitch
76	0.06	numdayscontractequipmentplanexpiring
80	0.06	percentagecalloutsidenetwork
82	0.04	avgcallduration
81	0.04	totalcallduration

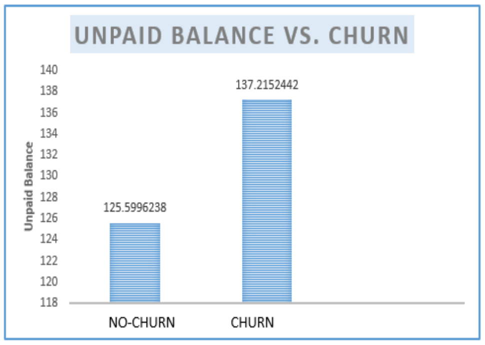
The churn is a very complicated metric.

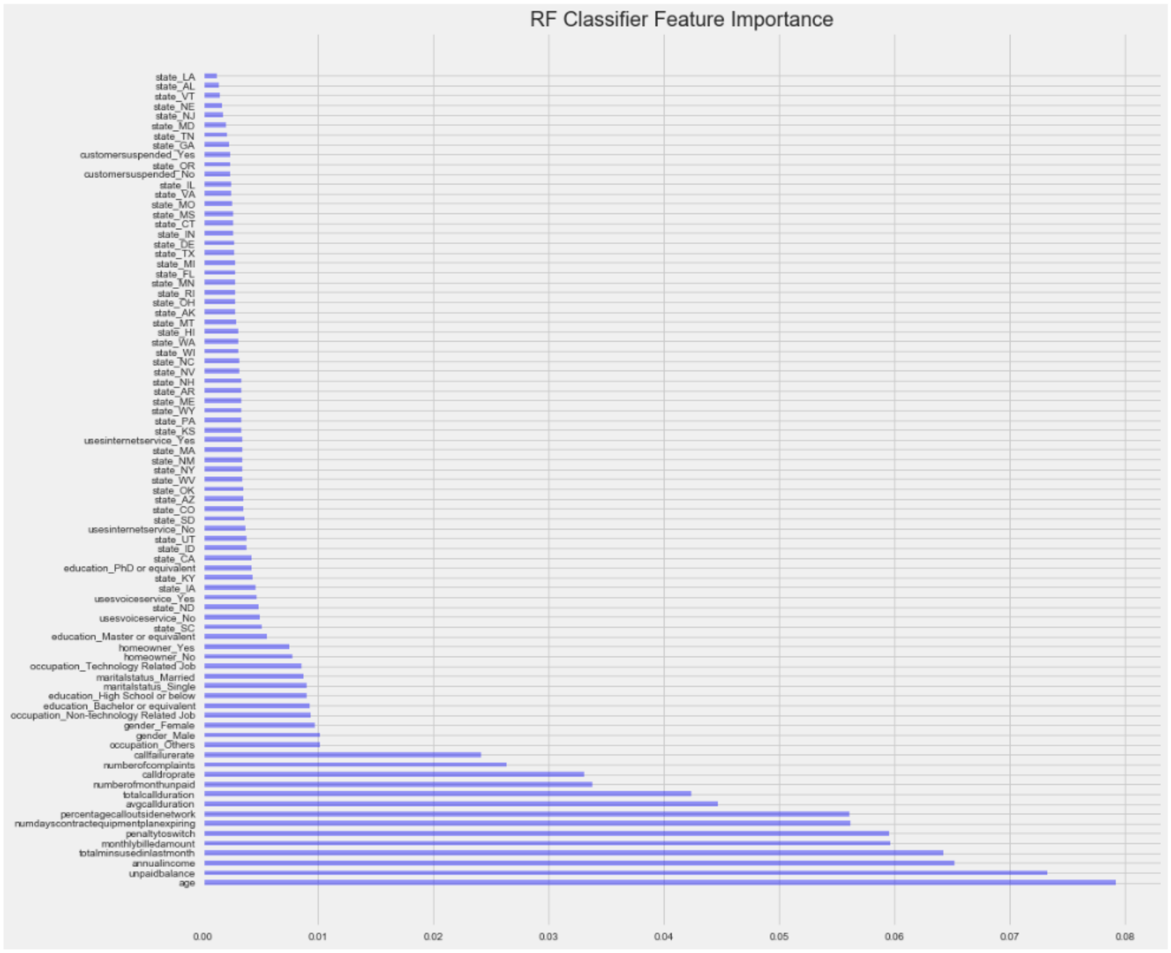
As we can see from the feature importance ranking table for our model, there is no one single feature is essential for distinguishing the one category from the other.

This suggests that we can't singling out some features while ignoring the others.

But the ‘age’ and ‘unpaid balance’ features are in the top of our ranking and if we go back and plot these two features alone then we can see the younger customers and higher unpaid balanced customers more likely to churn;







Reflection

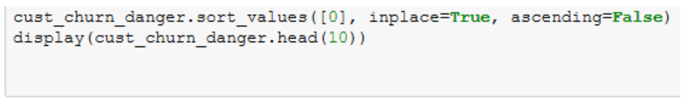
I work in the wireless telecom industry for more than 15 years and to reduce churn is more important than ever right now, particularly the telecom industry's growing competitive environment.

The output of this project to provide a churn probability score for each customer for the wireless operator. To summarize, these steps taken to predict the churn probability of our wireless customers:

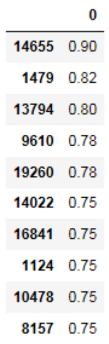
- Define the question of interest (which customers are likely to churn?)
- Import, explore and clean the data
- Preprocess the data (encoding string variables & feature scaling)
- Divide the dataset into train and test subsets
- Build and evaluate a benchmark model (Random Forest)
- Build a model (Neural Network "Sequential module" for initializing and "Dense module" to add hidden layers)
- Compile and fit the model

- Fine tune the model parameters
- Predict the test set labels and evaluate the model
- Visualize the most important features and customers in danger to churn

These customers are our top 10 contenders close to churn based on our model's probability results



```
cust_churn_danger.sort_values([0], inplace=True, ascending=False)
display(cust_churn_danger.head(10))
```



	0
14655	0.90
1479	0.82
13794	0.80
9610	0.78
19260	0.78
14022	0.75
16841	0.75
1124	0.75
10478	0.75
8157	0.75

Improvement

The next phase would be to iterate over the steps (data cleaning, pre-processing, fine-tuning, etc.) in an effort to increase the accuracy and decrease the cost of 'false negative' of our model. Using a more complete, larger and up-to-date dataset, the addition of new features, the tuning of parameters would boost our model. And then we can use the model in a production environment where it can be maintained and constantly updated. In the production environment, these scores can help to build an incentive policy and customer care representatives can act according to this policy and customer's score.