Neural Network in Keras

Jason Yingling

May 5, 2018

1 Introduction

Customer churn rate is the rate at which customers leave a company. For example, if a company has a churn rate of 25%, then the lifetime of its customers is approximately 4 years. Many companies that operate on a subscription model commonly use customer attrition as a metric to analyze the health of their business. The churn rate differs between industries but for this paper, we will focus on churn for a telecommunication company provided by IBM. The telecommunication industry spends millions of dollars to acquire new customers and most new customers are acquired from other providers.

Telecommunications companies have two approaches to dealing with churn, broad and personalized. The broad approach is based on advertising to improve brand image, leading to retention of customers. The personalized approach is a new avenue that has come about due to improvements in the availability and analytical techniques of data. This approach identifies customers that are most likely to leave the company and conducts marketing campaigns designed to convince them to stay. A company that only uses the broad approach does not have direct control or the optimization ability that the personalized approach provides.

Neural networks have emerged as useful tools for classification problems. Recent research has shown that neural networks provide an alternative to conventional classification methods. Neural networks have several advantages that make them worth further investigation. Being data driven, neural networks are able to adapt to data without any explicit specification by the modeler. Therefore there are no limitations due to distribution, functional form, etc. that other classification systems require. Stinchcombe and White (1989) showed that neural networks are able to approximate any function within bounds of accuracy. This is important, because classification problems seek to identify a functional relationship between the object being classified and the associated variables. Lastly, neural networks are able to estimate posterior probabilities, which helps establish classification rules [].

2 Data Description

The data for predicting churn for a telecommunication company is provided by IBM. The dataset consists of 7042 customers and 19 variables:

- customers who left the company within the last month (Churn)
- Services the customer has signed up for: phone, multiple lines, internet, online security, device protection, tech support, online backup, streaming TV and Movies
- Customer account information: How long they have been with the company, payment method, paperless billing, monthly charges, and total charges
- Demographic info: gender, partner, dependents, senior citizen

3 Data Preparation

Selection of a set of appropriate input features is an important issue for neural networks, as it is with any other model. However, unlike traditional methods, overall accuracy will not be noticeably affected by the inclusion of additional variables. Neural networks are able to effectively diminish the impact of the presence of a variable unrelated to the problem by the adjustment of the synapse weights within the neural network. The representation of relationships between variables as weights in the neural network also comes with a deficiency. If variables are of different sizes, the neural network will not be able to distinguish the degree of change caused by simple numerical difference rather than correlation. To alleviate this problem, all quantitative data will need to be normalized by a technique such as min-max normalization, or z-score normalization.

Although neural networks do not require normality, highly skewed data can affect neural networks. Therefore, it is beneficial to check the degree of skewedness for all quantitative data. After checking the skew of our three quantitative variables, we only needed to use the Box-Cox transformation on "Total Charge" variable.

Neural networks also struggle to handle categorical data. Because neural networks are simply a collection of weights, categorical data cannot be represented accurately. To handle this issue, we use "One hot encoding" on the categorical variables. One hot encoding is a process by which categorical variables are given a boolean column for each category within each categorical variable.

4 Design Decisions for Neural Network

We make the following design decisions:

Topology: We used several topologies to try and see which design worked best to classify churn.

- Model1: 2 hidden layers, 128-64 nodes, 100 epochs

- Model2: 2 hidden layers, 128-64 nodes, 200 epochs
- Model3: 2 hidden layers, 128-64 nodes, 200 epochs 20% dropout
- Model4: 2 hidden layers, 128-64 nodes, 500 epochs 50% dropout
- Model5: 4 hidden layers, 200-200-200-200 nodes, 500 epochs, 50 % dropout

Dropout is used as a regularization technique to help reduce over-fitting the training data. Dropout refers to dropping out hidden or visible nodes in a neural network. At each stage, individual nodes are dropped out of the network with probability p; incoming and outgoing synapse connections are also removed.

Activation Function: We use the ReLU function

$$f(x) = max(0, x)$$

as an activation function for all hidden neurons. The derivative of the ReLU function is:

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

For the output neuron, we use the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$

4.1 Output Encoding

Each output of the neural network corresponds to a combination of the values of its output units. The neural network has to decide whether a customer will leave or stay. In this case, we set all output values greater than 0.5 to "yes" and all output values less than or equal to 0.5 to "no".

4.2 Loss Function

We use the binary, cross-entropy loss function:

$$J = -\frac{1}{n} \sum_{i=1}^{n} y^{(i)} \log(f(x^{(i)}) + (1 - y^{(i)}) \log(1 - f(x^{(i)}))$$

where, $X = x^{(1)}, ..., x^{(n)}$ is the set of inputs and $Y = y^{(1)}, ..., y^{(n)}$ corresponds to either 0 or 1. $f(x)$ represents the output of the neural network given input x.

4.3 Training (Vectorization)

The full algorithm for training a neural network is notationally tedious. Vectorizing the equations condenses the notation and also provides a more computationally efficient way to train a neural network. The algorithm for a subset, size m of the training data is:

- Input a set of training values
- for each training observation x: set the input activation: Feed-forward: For each layer, compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = f(z^{x,l})$.

  Output error: $\delta^{x,L} = \Delta_a J_x \odot f'(z^{x,L})$

  Back-propagate error: for each layer compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot f'(z^{x,l})$
- Gradient descent: for each layer, update the weights according to the loss function and the bias according to $b^l \to b^l - \frac{m}{n} \sum \delta^{x,l}$

(Nielson, 2015)

5 Feature Selection

Variable importance is measured differently, depending on the model being utilized. Generally, features are ranked according to their contribution to the predictive power of the model in relation to each other. This importance can easily be used to assess which variables have the most impact on the model and which variables may be omitted.

In neural networks, variables are combined as inputs into one or more hidden layers, then are converted to weights. These weights are then used as predictions of the final output. These predictions can be very accurate, but the accuracy comes with the loss of ability to interpret the variables. Because the variable relationships are represented in the neural network as weights, neural networks turn into a "black box" model.

5.1 Tuning ANN

We are able to further improve Neural Networks through "hyper-parameter" optimization. Hyper-parameters are values set prior to the learning process. The initialization of hyper-parameters prior to learning means they are not derived via training. Therefore, they could potentially change the accuracy of the model, but must be changed prior to a new training cycle. In this paper, we used grid search. Grid search creates a wrapper function that sets the various combination of hyper-parameters that the programmer wants to test, and then the wrapping function will apply these over the pre-built neural network.

Grid search took over 6 hours on the computer that the code was run on. Grid search is very inefficient, but is embarrassingly parallel. So the time to find the optimal hyper-parameters has the potential to be greatly reduced.

6 Results

To create a standard basis to compare the neural networks with, we ran the data through a logistic regression. The results were 80% prediction accuracy, 12.2% false positive and 7.1% false negative.

Churn Prediction			
Stastic	Model1	Model2	Model3
Accuracy	74.8%	74.8%	78.4%
False (+)	10.8%	11.7%	5.5%
False (-)	14.6%	13.4%	16.1%

Churn Prediction			
Stastic	Model4	Model5	Logistic
Accuracy	78.2%	78.7%	80.0%
False (+)	5.1%	8.2%	12.2%
False (-)	16.7%	13%	7.1%

As we can see, the first attempt at the the neural network showed the lowest accuracy. Since there is no standard methodology for improving a neural network, optimization becomes more art than science. The best one can do is continuously change different hyper-parameters and different topologies to see if there are any improvements. As we progress through the different models, we see minor improvements after each modification. Therefore we may have reason to believe that the model can stand to be improved further by even more neurons per layer or a deeper network.

7 Conclusion

Preliminary, less precise results for smaller neural network were improved after changing the topology and hyper-parameters, and adding dropout. Simply increasing the epochs showed minor improvements, but introducing dropout showed immediate increase in prediction accuracy. Lastly, when we increased the network dimensions, the model continued to improve, leading to the possibility that further expansion would lead to better results. Although we were unable to achieve the results that a simple logistic regression made, we were successfully able to show that neural networks are a viable alternative for classification problems. In the presence of non-linear relationships between the data, the neural network could potentially outperform the logistic regression.

8 Bibliography

B. Amirikian and H. Nishimura, What size network is good for generalization of a specific task of interest?, Neural Networks, vol. 7, no. 2,pp. 321329, 1994.

E. B. Baum, What size net gives valid generalization?, Neural Comput. , vol. 1, pp. 151160, 1989

E. Barnard, Optimization for training neural nets, IEEE Trans. Neural Networks, vol. 3, pp. 232240, 1992

T. G. Dietterich, Overfitting and undercomputing in machine learning, Comput. Surv., vol. 27, no. 3, pp. 326327, 1995

Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep learning. Cambridge,MA: MIT Press

Li, Z., Gong, B., Yang, T. (2016). Improved dropout for shallow and deep learning. In D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, R. Garnett (Eds.), Advancesin neural information processing systems (pp. 19). N.p.: Preproceedings.

K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, Neural Networks, vol. 2, pp. 359366, 1989

Kingma, D., Ba, J. (2014). Adam: Amethod for stochastic optimization. arXiv 1412.6980.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In Proceedings of the 24th International Conference on Machine Learning (pp. 473480) N.p.: International Machine Learning Society.

L. Prechelt, A quantitative study of experimental evaluation of neural network algorithms: Current research practice, Neural Networks, vol. 9, no. 3, pp. 457462, 1996

M. D. Richard and R. Lippmann, Neural network classifiers estimate Bayesian a posteriori probabilities, Neural Comput., vol. 3, pp. 461483, 1991.

J. W. Shavlik, R. J. Mooney, and G. G. Towell, Symbolic and neural learning algorithms: An empirical comparison, Mach. Learn. , vol. 6, pp. 111144, 1991

```
# Part 1 - Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Importing the dataset
dataset = pd.read_csv('telecomchurn.csv')
# Remove rows with entry errors
dataset.drop(dataset.index[[5218,3331,4380,753,3826,
                            1082,488,1340,6754,6670,936]],inplace=True)
dataset.TotalCharges = pd.DataFrame(dataset.TotalCharges, dtype='float')
dataset.SeniorCitizen = pd.DataFrame(dataset.SeniorCitizen, dtype='object')

X = dataset.drop(columns=['customerID','Churn'])
y = dataset.Churn
y = y.map(dict(Yes=1, No=0)).values
```

```
#Investigate data

dataset.isnull().sum() #checking for total null values

f,ax=plt.subplots(1,2,figsize=(18,8))
dataset['Churn'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax
ax[0].set_title('Churn')
ax[0].set_ylabel('')
sns.countplot('Churn',data=dataset,ax=ax[1])
ax[1].set_title('Churn')
plt.show()


f,ax=plt.subplots(1,3,figsize=(18,8))
sns.distplot(dataset.TotalCharges,ax=ax[0])
ax[0].set_title('Total Amount Charged')
sns.distplot(dataset.MonthlyCharges,ax=ax[1])
ax[1].set_title('Monthly Charged')
sns.distplot(dataset.MonthlyCharges,ax=ax[2])
plt.show()


f,ax=plt.subplots(1,3,figsize=(18,8))
sns.distplot(dataset.tenure,ax=ax[0])
ax[0].set_title('Tenure')
sns.distplot(dataset.MonthlyCharges,ax=ax[1])
ax[1].set_title('Monthly Charges')
sns.distplot(dataset.TotalCharges,ax=ax[2])
ax[2].set_title('Total Charges')
plt.show()


f,ax=plt.subplots(1,2,figsize=(18,8))
dataset['Contract'].value_counts().plot.bar(ax=ax[0])
ax[0].set_title('Number Of Customers By Contract')
ax[0].set_ylabel('Count')
sns.countplot('Contract',hue='Churn',data=dataset,ax=ax[1])
ax[1].set_title('Contract:Leave vs Stay')
plt.show()


f,ax=plt.subplots(1,2,figsize=(18,8))
dataset['PaymentMethod'].value_counts().plot.bar(ax=ax[0])
ax[0].set_title('Number Of Customers By Payment Method')
```

```
ax[0].set_ylabel('Count')
sns.countplot('PaymentMethod',hue='Churn',data=dataset,ax=ax[1])
ax[1].set_title('Payment Method:Leave vs Stay')
plt.show()

f,ax=plt.subplots(1,2,figsize=(18,8))
dataset['InternetService'].value_counts().plot.bar(ax=ax[0])
ax[0].set_title('Number Of Customers By Internet Service')
ax[0].set_ylabel('Count')
sns.countplot('InternetService',hue='Churn',data=dataset,ax=ax[1])
ax[1].set_title('Internet Service:Leave vs Stay')
plt.show()


f,ax=plt.subplots(1,2,figsize=(18,8))
sns.violinplot("PaymentMethod","tenure", hue="Churn", data=dataset,split=True,ax=
ax[0].set_title('PaymentMethod and tenure vs Churn')
ax[0].set_yticks(range(0,100,10))
sns.violinplot("PaymentMethod","tenure", hue="Churn", data=dataset,split=True,ax=
ax[1].set_title('PaymentMethod and Tenure vs Churn')
ax[1].set_yticks(range(0,100,10))
plt.show()



f,ax=plt.subplots(2,2,figsize=(20,15))
sns.countplot('MultipleLines',data=dataset,ax=ax[0,0])
ax[0,0].set_title('No. Of Customers MultipleLines')
sns.countplot('MultipleLines',hue='gender',data=dataset,ax=ax[0,1])
ax[0,1].set_title('Male-Female Split for MultipleLines')
sns.countplot('MultipleLines',hue='Churn',data=dataset,ax=ax[1,0])
ax[1,0].set_title('MultipleLines vs Churn')
sns.countplot('MultipleLines',hue='PaymentMethod',data=dataset,ax=ax[1,1])
ax[1,1].set_title('MultipleLines vs PaymentMethod')
plt.subplots_adjust(wspace=0.2,hspace=0.5)
plt.show()


f,ax=plt.subplots(1,3,figsize=(20,8))
sns.distplot(dataset[dataset['Contract']=='Month-to-month'].MonthlyCharges,ax=ax
ax[0].set_title('Charges by Month-to-month')
sns.distplot(dataset[dataset['Contract']=='One year'].MonthlyCharges,ax=ax[1])
ax[1].set_title('Charges by One year')
sns.distplot(dataset[dataset['Contract']=='Two year'].MonthlyCharges,ax=ax[2])
ax[2].set_title('Charges by Two year')
plt.show()
```

```
f,ax=plt.subplots(1,3,figsize=(20,8))
sns.distplot(dataset[dataset['Contract']=='Month-to-month'].TotalCharges,ax=ax[0
ax[0].set_title('Charges by Month-to-month')
sns.distplot(dataset[dataset['Contract']=='One year'].TotalCharges,ax=ax[1])
ax[1].set_title('Charges by One year')
sns.distplot(dataset[dataset['Contract']=='Two year'].TotalCharges,ax=ax[2])
ax[2].set_title('Charges by Two year')
plt.show()

sns.heatmap(dataset.corr(),annot=True,cmap='RdYlGn',linewidths=0.2) #data.corr()-
fig=plt.gcf()
plt.show()



# Check the skew of all numerical features

from scipy import stats
from scipy.stats import norm, skew

numeric_feats = dataset.dtypes[dataset.dtypes != "object"].index

skewed_feats = dataset[numeric_feats].apply(lambda x: skew(x.dropna())).sort_val

skewness = pd.DataFrame({'Skew' :skewed_feats})
print(skewness)

fig = plt.figure()
res = stats.probplot(dataset['TotalCharges'], plot=plt)
plt.show()

#Transform TotalCharges
dataset['TotalCharges'] = stats.boxcox(dataset['TotalCharges'])[0]

sns.distplot(dataset['TotalCharges'] , fit=norm);

fig = plt.figure()
res = stats.probplot(dataset['TotalCharges'], plot=plt)
plt.show()

# Encoding categorical data
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

X_1 = X.select_dtypes(include=[object])
```

```
le = preprocessing.LabelEncoder()
X_1 = X_1.apply(le.fit_transform)
X_3 = X.select_dtypes(include=[float,int])

X = X_1.join(X_3).values

onehotencoder = OneHotEncoder(categorical_features = [0,1,2,3,4,5,6,7,8,9,10,11,
X = onehotencoder.fit_transform(X).toarray()


# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rando

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
# Initialising the ANN
classifier = Sequential()
# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 128, kernel_initializer = 'uniform', activation = '
#Adding the second hidden layer
classifier.add(Dense(units = 64, kernel_initializer = 'uniform', activation = 'r
# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sig
# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = [
# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 32, epochs = 100)

# Part 3 - Making predictions and evaluating the model
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
print((cm[0,0]+cm[1,1])/(len(y_pred)))

# Importing the Keras libraries and packages

# Initialising the ANN
classifier2 = Sequential()
# Adding the input layer and the first hidden layer
classifier2.add(Dense(units = 128, kernel_initializer = 'uniform', activation =
#Adding the second hidden layer
classifier2.add(Dense(units = 64, kernel_initializer = 'uniform', activation = '
# Adding the output layer
classifier2.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 's
# Compiling the ANN
classifier2.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
# Fitting the ANN to the Training set
classifier2.fit(X_train, y_train, batch_size = 32, epochs = 200)
# Part 3 - Making predictions and evaluating the model
# Predicting the Test set results
y_pred2 = classifier2.predict(X_test)
y_pred2 = (y_pred2 > 0.5)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm2 = confusion_matrix(y_test, y_pred2)
print((cm2[0,0]+cm2[1,1])/(len(y_pred2)))

# Importing the Keras libraries and packages

# Initialising the ANN and add dropout
classifier3 = Sequential()
classifier3.add(Dense(units = 128, kernel_initializer = 'uniform', activation =
classifier3.add(Dropout(p = 0.2))
classifier3.add(Dense(units = 64, kernel_initializer = 'uniform', activation = '
classifier3.add(Dropout(p = 0.2))
classifier3.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 's
classifier3.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
# Fitting the ANN to the Training set
classifier3.fit(X_train, y_train, batch_size = 32, epochs = 200)
# Part 3 - Making predictions and evaluating the model
# Predicting the Test set results
y_pred3 = classifier3.predict(X_test)
y_pred3 = (y_pred3 > 0.5)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm3 = confusion_matrix(y_test, y_pred3)
```

```
print((cm3[0,0]+cm3[1,1])/(len(y_pred)))

# Importing the Keras libraries and packages

# Initialising the ANN
classifier4 = Sequential()
# Adding the input layer and the first hidden layer
classifier4.add(Dense(units = 128, kernel_initializer = 'uniform', activation =
classifier4.add(Dropout(p = 0.5))
classifier4.add(Dense(units = 64, kernel_initializer = 'uniform', activation = '
classifier4.add(Dropout(p = 0.5))
# Adding the output layer
classifier4.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 's
# Compiling the ANN
classifier4.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
# Fitting the ANN to the Training set
classifier4.fit(X_train, y_train, batch_size = 32, epochs = 500)
# Part 3 - Making predictions and evaluating the model
# Predicting the Test set results
y_pred4 = classifier4.predict(X_test)
y_pred4 = (y_pred4 > 0.5)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm4 = confusion_matrix(y_test, y_pred4)
print((cm4[0,0]+cm4[1,1])/(len(y_pred4)))

# Importing the Keras libraries and packages

# Initialising the ANN
classifier5 = Sequential()
# Adding the input layer and the first hidden layer
classifier5.add(Dense(units = 200, kernel_initializer = 'uniform', activation =
classifier5.add(Dropout(p = 0.5))
classifier5.add(Dense(units = 200, kernel_initializer = 'uniform', activation =
classifier5.add(Dropout(p = 0.5))
classifier5.add(Dense(units = 200, kernel_initializer = 'uniform', activation =
classifier5.add(Dropout(p = 0.5))
classifier5.add(Dense(units = 200, kernel_initializer = 'uniform', activation =
classifier5.add(Dropout(p = 0.5))
# Adding the output layer
classifier5.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 's
# Compiling the ANN
classifier5.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
# Fitting the ANN to the Training set
classifier5.fit(X_train, y_train, batch_size = 32, epochs = 500)
```

```
# Part 3 - Making predictions and evaluating the model
# Predicting the Test set results
y_pred5 = classifier5.predict(X_test)
y_pred5 = (y_pred5 > 0.5)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm5 = confusion_matrix(y_test, y_pred5)
print((cm5[0,0]+cm5[1,1])/(len(y_pred)))

# Importing the Keras libraries and packages

# Tuning the ANN
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(units = 64, kernel_initializer = 'uniform', activation =
    classifier.add(Dense(units = 32, kernel_initializer = 'uniform', activation =
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation =
    classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy', metr
    return classifier
classifier = KerasClassifier(build_fn = build_classifier)
parameters = {'batch_size': [25, 32],
              'epochs': [100,200,300,500],
              'optimizer': ['adam', 'rmsprop']}
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10)
grid_search = grid_search.fit(X_train, y_train)
best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_

# Tuning the ANN
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(units = 128, kernel_initializer = 'uniform', activation
    classifier.add(Dropout(p = 0.2))
    classifier.add(Dense(units = 64, kernel_initializer = 'uniform', activation =
```

```
    classifier.add(Dropout(p = 0.2))
    classifier.add(Dense(units = 64, kernel_initializer = 'uniform', activation =
    classifier.add(Dropout(p = 0.2))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation =
    classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy', metr
    return classifier
classifier = KerasClassifier(build_fn = build_classifier)
parameters = {'batch_size': [25, 32],
              'epochs': [100,200,300,500],
              'optimizer': ['adam', 'rmsprop']}
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10)
grid_search = grid_search.fit(X_train, y_train)
best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_


# Tuning the ANN
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(units = 128, kernel_initializer = 'uniform', activation
    classifier.add(Dropout(p = 0.5))
    classifier.add(Dense(units = 64, kernel_initializer = 'uniform', activation =
    classifier.add(Dropout(p = 0.5))
    classifier.add(Dense(units = 64, kernel_initializer = 'uniform', activation =
    classifier.add(Dropout(p = 0.5))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation =
    classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy', metr
    return classifier
classifier = KerasClassifier(build_fn = build_classifier)
parameters = {'batch_size': [25, 32],
              'epochs': [100,200,300,500],
              'optimizer': ['adam', 'rmsprop']}
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10)
grid_search = grid_search.fit(X_train, y_train)
best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_
```

```
# Make Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
model = LogisticRegression()
model.fit(X_train, y_train)
prediction3=model.predict(X_test)
metrics.accuracy_score(prediction3, y_test)
lcm = confusion_matrix(prediction3, y_test)
```