Customer Retention NLP-ML Model

A machine learning model is a mathematical representation of the output of the training process. Machine learning is the study of different algorithms that can improve automatically through experience & old data and build the model. A machine learning model is similar to computer software designed to recognise patterns or behaviours based on previous experience or data. The learning algorithm discovers patterns within the training data and outputs an ML model that captures these patterns and makes predictions on new data.

In this case, we are looking at a model built on the predicted customer retention based on their engagement manager's comments about the case. We have used Natural Language Processing and Machine Learning to obtain our results. We are testing different machine learning algorithms of differing complexities to find what is the best-suited model in this scenario. We start with Naive Bayes and then move to a Random Forest model, and then we use a boosting method called XGBoost. We compare the accuracies of these algorithms and understand how each model excels and outdoes the other.

Naive Bayes Algorithm

A Naive Bayes classifier is a machine learning algorithm to classify or filter data. It is primarily used in natural language processing, categorising news articles into topics, filtering spam mail, and sentiment analysis. The Naive Bayes classifier can classify, filter, or categorise text data. Below is sample use cases.

- Categorising news articles
- Filtering spam mail
- Analysing positive/negative sentiment

The Naive Bayes classifier has three algorithms: Gaussian Naive Bayes, multinomial Naive Bayes, and Bernoulli Naive Bayes. It is pretty complicated to understand all algorithms deeply. One of the simpler supervised Bayesian Network models, the Naive Bayes algorithm, is a probabilistic classifier based on Bayes' Theorem (which you might remember from high school statistics). But its simplicity doesn't make it a poor choice. It can produce highly accurate predictions even when the dataset is not very large (just a few thousand samples).

Random Forest Classifier

Random forest is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy-to-use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests create decision trees on randomly selected data samples, get a prediction from each tree and selects the best solution employing voting. It also provides a pretty good indicator of the feature's importance. Random forest has a variety of applications, such as recommendation engines, image classification and feature selection. It can classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects essential features in a dataset.

XGBoost.

XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core. It is an optimised distributed gradient boosting library. But what is boosting?
Boosting is a sequential technique which works on the principle of an ensemble. It combines a set of weak learners and delivers improved prediction accuracy. At any instant t, the model outcomes are weighed based on the outcomes of the previous instant t-1. The outcomes predicted correctly are given a lower weight, and the ones miss-classified are weighted higher. Note that a weak learner is one which is slightly better than random guessing. For example, a decision tree whose predictions are slightly better than 50%.

Now that we know all three algorithms let us understand their project application.

In this case, we are classifying customer comments to predict whether they will pay the renewal amount. I have taken a dataset from my company's retention team, and the engagement managers who work there try to retain the customer. Each manager updates the comments about a specific customer, whether he is likely to pay or what the update is regarding that particular case.
Here the comments are made into a corpus, and we create a bag of words model. The X variable is the bag of words output, and the y variable is whether the customer paid or not. From the bag of words model, we match the outcome (y variable) and train all three of our models accordingly. First, we use the Naive Bayes conditional probability method to determine which binary output it will fall to. Then we apply the random forest classifier and boost it later on for better results. Either 'Paid' or "Not Paid" is the binary output. This model behaves like a sentiment analysis model, and we can link the comment to whether the customer will renew or not.

Here, I have built the model, imported my company's dataset, and applied all three models to it to figure out which model performs better and gives us better accuracy. Below we can see the code.

Code:

```
#importing data from google sheets
from google.colab import auth
auth.authenticate_user()

import gspread
from google.auth import default
creds, _ = default()

gc = gspread.authorize(creds)

worksheet = gc.open('NLP_Comments').sheet1

# get_all_values gives a list of rows.
rows = worksheet.get_all_values()

# Convert to a DataFrame and render.
import pandas as pd
dataset=pd.DataFrame.from_records(rows)
```

```
#Converting first row to header
dataset.columns = dataset.iloc[0]
dataset = dataset[1:]
```

```
dataset['Comments'],dataset['Status']
```

```
(1                                                   Upgradation
 2      disconnecting the calls\nNR, call back on 7th ...
 3      She had paid for m2 and have been moved to thi...
 4      Stopped attending classes due to exams, wants ...
 5      Did not get connected//Connect on Saturday//No...
                              ...
 592                                                 Refund case
 593                 started on 2nd Oct, not up for renewal
 594                 started on 2nd Oct, not up for renewal
 595            the mother has not done the payment \nNR,NR
 596                                  Enrolled for M2 on 31/10
 Name: Comments, Length: 596, dtype: object, 1          Paid
 2      Not Paid
 3          Paid
 4      Not Paid
 5      Not Paid
          ...
 592    Not Paid
 593        Paid
 594        Paid
 595        Paid
 596        Paid
 Name: Status, Length: 596, dtype: object)
```

```
pip install stop_words
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting stop_words
  Downloading stop-words-2018.7.23.tar.gz (31 kB)
Building wheels for collected packages: stop-words
  Building wheel for stop-words (setup.py) ... done
  Created wheel for stop-words: filename=stop_words-2018.7.23-py3-none-any.whl size=32911 sha256=a5a761812120ad3f0683de27fa87722b592a1c62c7fd1fec1bb5cf9c73c355c1
  Stored in directory: /root/.cache/pip/wheels/fb/86/b2/277b10b1ce9f73ce15059bf6975d4547cc4ec3feeb651978e9
Successfully built stop-words
Installing collected packages: stop-words
Successfully installed stop-words-2018.7.23
```

```
import nltk
nltk.download('punkt')
from nltk.corpus import stopwords
nltk.download('stopwords')
print(stopwords.words('english'))


from nltk.tokenize import word_tokenize
from stop_words import get_stop_words
stop_words = list(get_stop_words('en'))         #About 900 stopwords
nltk_words = list(stopwords.words('english')) #About 150 stopwords
stop_words.extend(nltk_words)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself',
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
#removing stopwords from the comments
corpus=dataset['Comments'].apply(lambda words: ' '.join(word.lower() for word in words.split() if word not in stop_words))
print(corpus)
```

```
1                                    upgradation
2              disconnecting calls nr, call back 7th 10am
3                        she paid m2 moved batch recently
4       stopped attending classes due exams, wants joi...
5       did get connected//connect saturday//no commun...
                              ...
592                                  refund case
593                          started 2nd oct, renewal
594                          started 2nd oct, renewal
595                         mother done payment nr,nr
596                                 enrolled m2 31/10
Name: Comments, Length: 596, dtype: object
```

```
# Creating the Bag of Words model
from sklearn.feature_extraction.text import CountVectorizer

# To extract max 1500 features
cv = CountVectorizer(max_features = 1500)

# X contains corpus (dependent variable)
X = cv.fit_transform(corpus).toarray()

# y contains answers if comments is positive or negative
y = dataset.iloc[:, -1].values
```

Naive Bayes

```
# 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 = 0.25)
```

```
# Slicing dataset to match the target dataset
X_train=X_train[:, :937]
X_test=X_test[:, :937]
```

```
# fitting naive bayes to the training set
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix

classifier = GaussianNB();
classifier.fit(X_train, y_train)

# predicting test set results
y_pred = classifier.predict(X_test)

# making the confusion matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[52, 41],
       [13, 43]])
```

```
print("Naive Bayes score: ",classifier.score(X_test, y_test))
```

```
Naive Bayes score:  0.6375838926174496
```

Random Forest

```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier

# n_estimators can be said as number of trees
model = RandomForestClassifier(n_estimators = 501,
                    criterion = 'entropy')

model.fit(X_train, y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=501)
```

```python
# Predicting the Test set results
y_pred = model.predict(X_test)

y_pred
```

```
array(['Paid', 'Paid', 'Not Paid', 'Not Paid', 'Not Paid', 'Paid', 'Paid',
       'Not Paid', 'Paid', 'Not Paid', 'Not Paid', 'Paid', 'Paid', 'Paid',
       'Paid', 'Not Paid', 'Not Paid', 'Paid', 'Not Paid', 'Not Paid',
       'Paid', 'Paid', 'Not Paid', 'Not Paid', 'Not Paid', 'Not Paid',
       'Paid', 'Not Paid', 'Paid', 'Paid', 'Paid', 'Paid', 'Not Paid',
       'Paid', 'Paid', 'Not Paid', 'Not Paid', 'Not Paid', 'Paid',
       'Not Paid', 'Not Paid', 'Not Paid', 'Not Paid', 'Not Paid', 'Paid',
       'Paid', 'Not Paid', 'Not Paid', 'Not Paid', 'Paid', 'Paid', 'Paid',
       'Not Paid', 'Not Paid', 'Not Paid', 'Not Paid', 'Not Paid',
       'Not Paid', 'Not Paid', 'Paid', 'Not Paid', 'Paid', 'Paid', 'Paid',
       'Paid', 'Not Paid', 'Paid', 'Paid', 'Not Paid', 'Paid', 'Not Paid',
       'Paid', 'Not Paid', 'Not Paid', 'Paid', 'Paid', 'Not Paid',
       'Not Paid', 'Paid', 'Not Paid', 'Not Paid', 'Paid', 'Paid', 'Paid',
       'Not Paid', 'Not Paid', 'Not Paid', 'Paid', 'Not Paid', 'Paid',
       'Not Paid', 'Paid', 'Not Paid', 'Not Paid', 'Not Paid', 'Paid',
       'Not Paid', 'Paid', 'Paid', 'Not Paid', 'Paid', 'Not Paid', 'Paid',
       'Paid', 'Paid', 'Not Paid', 'Not Paid', 'Paid', 'Paid', 'Not Paid',
       'Paid', 'Not Paid', 'Paid', 'Not Paid', 'Paid', 'Not Paid',
       'Not Paid', 'Not Paid', 'Paid', 'Paid', 'Not Paid', 'Paid',
       'Not Paid', 'Not Paid', 'Paid', 'Not Paid', 'Not Paid', 'Not Paid',
       'Not Paid', 'Not Paid', 'Not Paid', 'Not Paid', 'Paid', 'Not Paid',
       'Paid', 'Paid', 'Paid', 'Paid', 'Not Paid', 'Paid', 'Not Paid',
       'Paid', 'Not Paid', 'Paid', 'Not Paid', 'Paid', 'Paid', 'Paid',
       'Paid'], dtype=object)
```

```python
from sklearn.metrics import classification_report, confusion_matrix#
print("Classification report - \n",classification_report(y_test,y_pred))
```

```
Classification report - 
               precision    recall  f1-score   support

    Not Paid       0.86      0.72      0.78        93
        Paid       0.63      0.80      0.71        56

    accuracy                           0.75       149
   macro avg       0.75      0.76      0.75       149
weighted avg       0.77      0.75      0.76       149
```

```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

cm
```

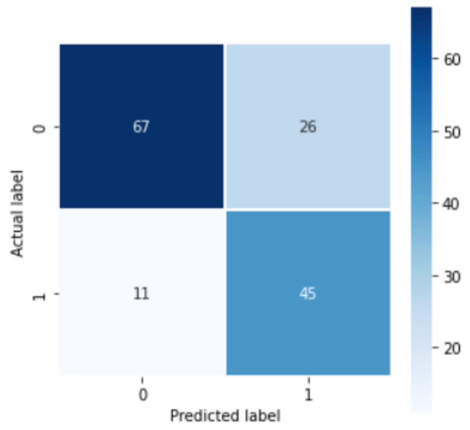
```
array([[67, 26],
       [11, 45]])
```

```python
#Accuracy score
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.7516778523489933
```

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,square = True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 48.29999999999998, 'Predicted label')
```



XGBoost

```
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

categorical_pipeline = Pipeline(
    steps=[
        ("impute", SimpleImputer(strategy="most_frequent")),
        ("oh-encode", OneHotEncoder(handle_unknown="ignore", sparse=False)),
    ]
)
```

```
from sklearn.preprocessing import StandardScaler

numeric_pipeline = Pipeline(
    steps=[("impute", SimpleImputer(strategy="mean")),
           ("scale", StandardScaler())]
)
```

```
# X contains corpus (dependent variable)
X = cv.fit_transform(corpus).toarray()

# y contains answers if comments is positive or negative
y = dataset.iloc[:, -1]
```

```python
from sklearn.compose import ColumnTransformer

full_processor = ColumnTransformer(
    transformers=[
        ("numeric", numeric_pipeline, num_cols),
        ("categorical", categorical_pipeline, cat_cols),
    ]
)
```

```python
import xgboost as xgb

xgb_cl = xgb.XGBClassifier()

print(type(xgb_cl))
```

```
<class 'xgboost.sklearn.XGBClassifier'>
```

```python
# Apply preprocessing
X_processed=X
y_processed = SimpleImputer(strategy="most_frequent").fit_transform(
    y.values.reshape(-1, 1)
)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_processed, y_processed, stratify=y_processed, random_state=55
)
```

```python
from sklearn.metrics import accuracy_score

# Init classifier
xgb_cl = xgb.XGBClassifier()

# Fit
xgb_cl.fit(X_train, y_train)

# Predict
preds = xgb_cl.predict(X_test)

# Score
accuracy_score(y_test, preds)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/
  y = column_or_1d(y, warn=True)
0.8791946308724832
```

Conclusion

As we can see from the above code, the Naive Bayes Model returned an accuracy of 64%; the random forest classifier returned an accuracy of 75%. After boosting with the XGBosst algorithm, we achieved an accuracy of 88%. We consistently improved the model's accuracy by using the proper models and applying the needed preprocessing. There are many models out there that can be applied to each use case. It is up to the analyst to determine which fits best in a particular case.

We could predict using these Machine Learning algorithms whether that particular customer would pay or not. This helps a company to build estimation reports and predict revenue for the coming months. Machine Learning can perform incredible tasks and reduce human intervention in iterative processes like checking the comments of each manager and checking case by case. Overall, the application of these tools and the ceiling of it are left to the user to figure out. Great things can be achieved when creativity and technologically advanced tools like this amalgamate. The power is in the hands of the analyst.