FlyNava Technologies
Airlines. Innovation

Machine Learning model to classify news Headlines as relevant and non-relevant

Relevant

Non-Relevant

Shivani Bhakal, IIT Roorkee

Introduction:

This project helps the users to classify the news headlines as relevant and non-relevant news to their company. Relevant news contains headlines which give information about new launch of flights or the increasing capacity or the information about new routes etc.

Table of Contents

Data Extraction:

- I extracted the 4 news sites containing news with specific keywords. These news headlines are mostly relevant and gives useful information in the context of company.
  The below 4 news sites are given the top priority for news headlines in demo.flynava.com.
  1. Etihad
  2. Qatar
  3. Ethiopian
  4. Emirates

  keywords=["launches","launch","capacity","routes","destination","offers","deploy","airbus",
  "increases","frequency","flights","book","pay","travel","dubai","route","price","rate","fare"]
  I used the find_all () function to find if words like in keywords list exists or not.If there are any such
- I then combined the code of all 4 news sites into one single code that will extract the relevant news from these sites in one go and saves into csv file.
- I got around 408 news headlines from these sites. Other than these news headlines, I got the news headlines from the collection named *JUP_DB_Google_Trigger.*

Data pre-processing:

- After extraction of the data, I assigned output values to all the news headlines.
  Output: 1 (Relevant news or useful news)
  Eg: Jet Airways to launch new daily flight from Delhi to Muscat

  Qatar Airways plans to launch first fully foreign-owned airline in India

  Output: 0 (Non-relevant news)
  Eg: Qatar Airways Debuts New 'No Borders' Video

  Emirates, Norwegian Step up Free Wi-Fi

- Now in my dataset, I have two fields named ‘Title’ and ‘Output’ and their corresponding values.
  I made use of *NLTK* library and removed all *stopwords* and managed to keep only *English words* in my dataset. For this I used *sentence* and *word tokenizer.*

- I then divided my dataset into Train and Test set with Test containing 1098 entries and Test set c ontains 98 entries.

- After that, I removed some of the common words from the Train as well as Test dataset that might affect my model, because these words will be in both Train as well as Test dataset.
  Remove list contains these common words:
  rem=["airways","etihad","emirates","jet","ethiopian","airlines","qatar","travelpulse","arabia","ai r","capa","centre","aviation"]

For this, I made a function named remove_unwanted(), that will take old_titles and new_titles as arguments.

- X_train and X_test contains all titles free from unwanted words, stopwords and contains only alp hanumeric-al values while X_train_target and X_test_target contains the Output values in form of numpy array.

Modelling:

- I used the CountVectorizer module from sklearn package. This convert a collection of text docum ents to a matrix of token counts.

fit_transform(raw_documents[, y])	Learn the vocabulary dictionary and return term-document matrix.

- TfidfTransformer module is used to transform a count matrix to a normalized tf or tf-idf represen tation.Tf means term-frequency while tf-idf means term-frequency times inverse document-frequ ency. This is a common term weighting scheme in information retrieval that has also found good u se in document classification.

- After that, I trained Naive Bayes classifier on my training data. For this, I used MultinomialNB module. Naive Bayes classifier for multinomial models
  The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.
  I chose Naïve Bayes as my classifier because when assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.

- Then I built a Pipeline, by building a pipeline, we can write less code and do all of the above, by building a pipeline as follows:
  The names ‘vect’, ‘tfidf’ and ‘clf’ are arbitrary but will be used later.
  We will be using the 'text_clf' going forward.

```
text_clf = Pipeline ([('vect', CountVectorizer ()), ('tfidf', TfidfTransformer()), ('clf', MultinomialNB())])
```

- I also used the SGDclassifier that takes linear SVM, which is the best for document classifier for pro blems where sparsity is high and features/instances are also very high.
- Later I will calculate the accuracy of both of these algorithm and takes the one which has highest a ccuracy.

Analysis of results:

- On calculating the performance of Training Support Vector Machines - SVM and NB Classifier, it was found that the SVM has better results in terms of accuracy.

```
In [63]:
# Performance of NB Classifier
import numpy as np
predicted = text_clf.predict(X_test)
np.mean(predicted == X_test_target)

Out[63]: 0.87755102040816324
```

```
In [30]:
# Training Support Vector Machines - SVM and calculating its performance
import numpy as np
from sklearn.linear_model import SGDClassifier
text_clf_svm = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                         ('clf-svm', SGDClassifier(loss='hinge', penalty='l2',alpha=1e-3, n_iter=5, random_state=42))])

text_clf_svm = text_clf_svm.fit(X_train, X_train_target)
predicted_svm = text_clf_svm.predict(X_test)
np.mean(predicted_svm == X_test_target)

Out[30]: 0.90816326530612246
```

We can see that **svm** gave 90.81% accurate result as compared to **NB classifier**.

Below is the result that **svm classifier** predicted.

```
In [60]:
X_test_target=df1['Output'].values # changes the column to numpy array.
X_test_target

Out[60]: array([0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
                0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
                0, 0, 1, 0, 0, 0], dtype=int64)
```

Fig: 1 shows the array of output values of test data.

```
In [31]:
predicted_svm

Out[31]: array([0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
                0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
                0, 1, 0, 0, 0, 0], dtype=int64)
```

Fig: 2 shows the array of output values of predicted data.

Automation for test data:

- This model will take the test data from the collection named *JUP_DB_Google_Trigger* where the Last_Updated_date is equal to current date. So that for every day we can predict that whether the news headlines are relevant or non-relevant.