FreePizza

November 1, 2018

1 Take-home challenge: Free Pizza! - Lorent Caravaku

There is a dataset where each row corresponds to a request of a Reddit user, that has triggered a free pizza action (or not). Given all the information provided, we will train a model that will predict if a certain request is likely to receive a free pizza or not.

We are going to use the dataset data.json provided. We are going to use a (2/3;1/3) ratio to split the dataset, so we can test our model.

The assumptions that we can make: The selection of user that will be offered a Pizza is based only on the text content and not on others information such as the date, the users votes, etc.

First we need to import all libraries.

```
In [90]: import json
         import pandas as pd
         import matplotlib.pyplot as plt
         from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.model_selection import train_test_split, cross_val_score
         from sklearn.linear_model import LogisticRegression
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.metrics import accuracy_score
         from sklearn import metrics
         from nltk.corpus import stopwords
         from nltk.stem import WordNetLemmatizer
         from nltk import word_tokenize
         from nltk import pos_tag
         from string import punctuation
```

We declare some variables/constants that we will need later.

```
In [91]: DATASET_FILE = 'data.json'
         STOPWORDS = 'stopwords.json'

         lmt = WordNetLemmatizer()
         nb = MultinomialNB()
         lr = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial')
```

We import the dataset.

```
In [92]: # Opening Dataset (data.json)
         with open(DATASET_FILE) as fin:
             data = json.load(fin)
```

We import some additional english stopwords that we will remove from the dataset. Stop-words are non-content words that primarily has only grammatical function.

```
In [93]: # Opening stopwords.json file
         with open(STOPWORDS) as fin:
             stopwords01 = json.load(fin)
```

We create a combined stoplist with three different sources of stopwords.

```
In [94]: # Creating Stopwords set
         stopwords01_ = set(stopwords01['en'])
         stopwords02_ = set(stopwords.words('english'))
         stopwords03_ = set(punctuation)
         stoplist_combined = set.union(stopwords01_, stopwords02_, stopwords03_)
```

"penn2morphy" converts penn treebank tags to WordNet.

```
In [95]: def penn2morphy(penntag):
             """ Converts Penn Treebank tags to WordNet. """
             # Input: str - peen tag
             # Output: str - wordNet
             morphy_tag = {'NN':'n', 'JJ':'a',
                           'VB':'v', 'RB':'r'}
             try:
                 return morphy_tag[penntag[:2]]
             except:
                 return 'n'
```

"lemmatize" maps the different forms of the same word to the same root word.

```
In [96]: def lemmatize(text):
             """ Finding the root words """
             # Input: str - text
             # Output: (lowercase) str - text.
             return [lmt.lemmatize(word.lower(), pos=penn2morphy(tag))
                     for word, tag in pos_tag(word_tokenize(text))]
```

"prepocess" will remove stopwords and lemmatize

```
In [97]: def preprocess(text):
             """ Removing stopwords and digits """
             # Input: str - document/sentence
             # Output: list(str) - list of lemmas
             return [word for word in lemmatize(text)
                     if word not in stoplist_combined
                     and not word.isdigit()]
```

Creating Dataframe from json formated data

```
In [98]: # Creating Dataframe from json formated data
         df = pd.io.json.json_normalize(data)
```

Keeping only relevant information (based on the assumptions made)

```
In [99]: # Keeping only relevant information (based on the assumptions made)
         df_train = df[['request_id', 'request_title',
                        'request_text_edit_aware',
                        'requester_received_pizza']]
```

Spliting data into two subsets : training dataset and test dataset

```
In [100]: # Spliting data into two subsets : training dataset and test dataset
          train, test = train_test_split(df_train, test_size=0.3)
```

Vectorizing Datasets

```
In [101]: # Vectorizing Datasets
          count_vect = CountVectorizer(analyzer=preprocess)
          train_set = count_vect.fit_transform(train['request_text_edit_aware'])
          train_tags = train['requester_received_pizza']
          test_set = count_vect.transform(test['request_text_edit_aware'])
          test_tags = test['requester_received_pizza']
```

Training NB and LR model

```
In [102]: # Training NB model
          nb_clf = nb.fit(train_set, train_tags)
          lr_clf = lr.fit(train_set, train_tags)
```

Testing the trained model

```
In [103]: # Testing the trained model
          predictions_test_nb = nb_clf.predict(test_set)
          predictions_test_lr = lr_clf.predict(test_set)
```

```
In [104]: # Printing accuracy
          print('\n-- Accuracy --')
          print('MultinomialNB = {}'.format(
                  accuracy_score(predictions_test_nb, test_tags) * 100)
               )
          print('LogisticRegression = {}'.format(
                  accuracy_score(predictions_test_lr, test_tags) * 100)
               )
```

```

-- Accuracy --
MultinomialNB = 75.4950495049505
LogisticRegression = 71.69966996699671
```

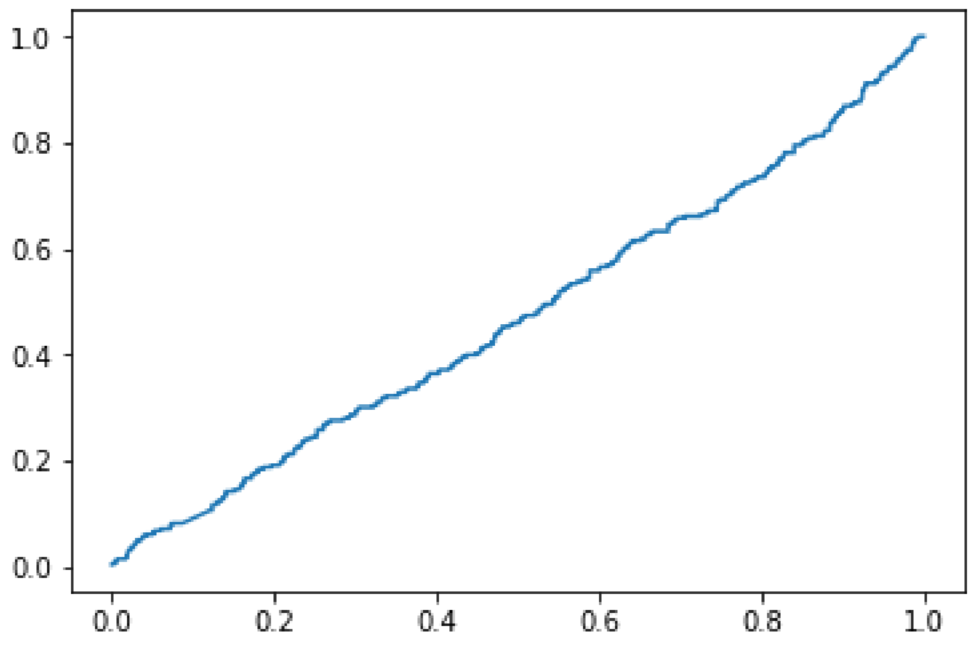
Cross Validation

```
In [105]: print('\n-- Cross Validation Accuracy --')
          scores_nb = cross_val_score(nb, train_set, train_tags, cv=3)
          print('MultinomialNB = ', scores_nb)
          scores_lr = cross_val_score(lr, train_set, train_tags, cv=3)
          print('LogisticRegression = ', scores_lr)
```

```
-- Cross Validation Accuracy --
MultinomialNB =  [ 0.70625663  0.69565217  0.71125265]
LogisticRegression =  [ 0.6892895   0.68610817  0.69214437]
```

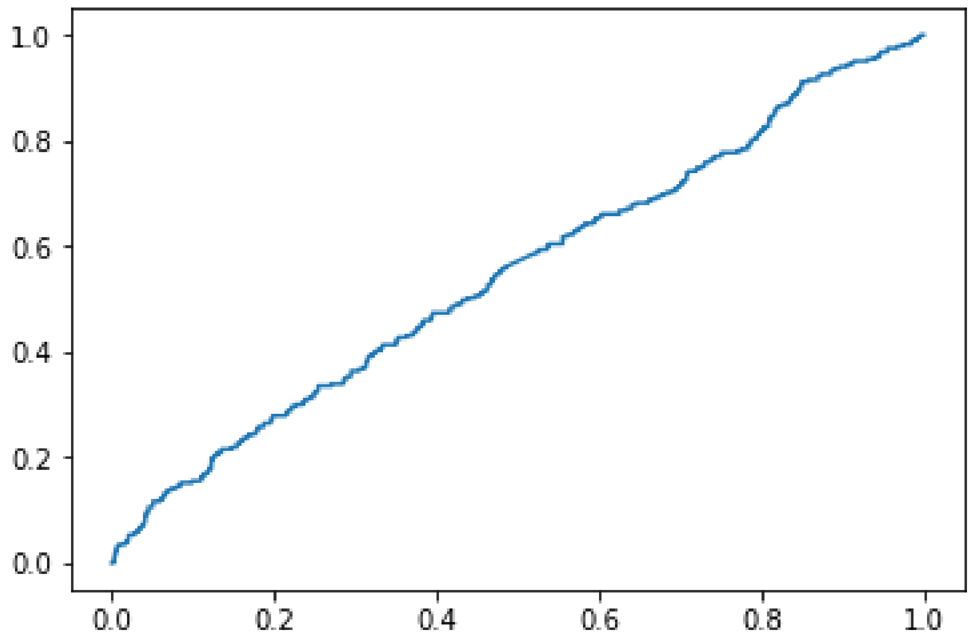
Plot MultinomialNB model ROC

```
In [106]: # Ploting ROC: Receiver Operating Characteristic
          fpr_nb, tpr_nb, thresh_nb = metrics.roc_curve(test_tags, nb_clf.predict_proba(test_set
          plt.plot(fpr_nb, tpr_nb)
          plt.show()
```



Plot LogisticRegression model ROC

```
In [107]: # Ploting ROC: Receiver Operating Characteristic
          fpr_lr, tpr_lr, thresh_lr = metrics.roc_curve(test_tags, lr_clf.predict_proba(test_set
          plt.plot(fpr_lr, tpr_lr)
          plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0