NLP: Predicting Upvotes Based on Headline

Introduction

Hacker News is a community where users can submit articles, and other users can upvote those articles. The articles with the most upvotes make it to the front page, where they're more visible to the community.

Goal

In this project, I'll be predicting the number of upvotes articles received, based on their headlines. Because upvotes are an indicator of popularity, I'll discover which types of articles tend to be the most popular.

Data

The data set consists of submissions users made to Hacker News from 2006 to 2015. Developer Arnaud Drizard used the Hacker News API to scrape the data, which can be found in one of his GitHub repositories (https://github.com/arnauddri/hn). I've sampled 3000 rows from the data randomly, and removed all of the extraneous columns. I will solely be working with the following four columns:

- `submission_time` - When the article was submitted
- `upvotes` - The number of upvotes the article received
- `url` - The base URL of the article
- `headline` - The article's headline

In [1]:
```
import pandas as pd
import numpy as np

submissions = pd.read_csv("sel_hn_stories.csv")
submissions.columns = ["submission_time", "upvotes", "url", "headline"]
submissions = submissions.dropna()
submissions.head()
```

Out[1]:

	submission_time	upvotes	url	headline
0	2010-02-17T16:57:59Z	1	blog.jonasbandi.net	Software: Sadly we did adopt from the construc...
1	2014-02-04T02:36:30Z	1	blogs.wsj.com	Google’s Stock Split Means More Control for L...
2	2011-10-26T07:11:29Z	1	threatpost.com	SSL DOS attack tool released exploiting negoti...
3	2011-04-03T15:43:44Z	67	algorithm.com.au	Immutability and Blocks Lambdas and Closures
4	2013-01-13T16:49:20Z	1	winmacsofts.com	Comment optimiser la vitesse de Wordpress?

Data Preparation

My goal is to train a linear regression algorithm that predicts the number of upvotes a headline would receive. To do this, I'll need to convert each headline to a numerical representation. I will be using the 'bag of words' model, which represents each piece of text as a numerical vector.

In [2]:

```
tokenized_headlines = []
for item in submissions['headline']:
    tokenized_headlines.append(item.split())

#preview the data
print(tokenized_headlines[0:5])
```

```
[['Software:', 'Sadly', 'we', 'did', 'adopt', 'from', 'the', 'construction',
'analogy'], ['Google’s', 'Stock', 'Split', 'Means', 'More', 'Control', 'for',
'Larry', 'and', 'Sergey'], ['SSL', 'DOS', 'attack', 'tool', 'released', 'expl
oiting', 'negotiation', 'overhead'], ['Immutability', 'and', 'Blocks', 'Lambd
as', 'and', 'Closures'], ['Comment', 'optimiser', 'la', 'vitesse', 'de', 'Wor
dpress?']]
```

Now that I have my tokens, I know they will need some processing to help with making predictions later on. I will need to get rid of punctuation, and make all words lowercase for consistency.

In [3]:

```
punctuation = [",", ":", ";", ".", "'", '"', "’", "?", "/",
               "-", "+", "&", "(", ")"]
clean_tokenized = []

for item in tokenized_headlines:
    tokens = []
    for token in item:
        token = token.lower()
        for punc in punctuation:
            token = token.replace(punc, "")
        tokens.append(token)
    clean_tokenized.append(tokens)
```

Now I will retrieve all unique words from each headline, create a matrix, and assign those words as column headers. After, I will populate the matrix with the number of token occurences.

In [4]:
```
unique_tokens = []
single_tokens = []
for tokens in clean_tokenized:
    for token in tokens:
        if token not in single_tokens:
            single_tokens.append(token)
        elif token in single_tokens and token not in unique_tokens:
            unique_tokens.append(token)

counts = pd.DataFrame(0, index=np.arange(len(clean_tokenized)), columns=unique
_tokens)
counts.head()
```

Out[4]:

	and	for	as	you	is	the	split	good	how	what	...	frameworks	animated	walks	auctio
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 2310 columns

In [5]:
```
for i, item in enumerate(clean_tokenized):
    for token in item:
        if token in unique_tokens:
            counts.iloc[i][token] += 1

counts.head()
```

Out[5]:

	and	for	as	you	is	the	split	good	how	what	...	frameworks	animated	walks	auctio
0	0	0	0	0	0	1	0	0	0	0	...	0	0	0	
1	1	1	0	0	0	0	1	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	2	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 2310 columns

Removing Columns to Increase Accuracy

My resulting matrix contains over 2000 columns. This will make it difficult to implement a linear regression model to make accurate predictions. To fix this, I will remove all columns that represent stopwords and words that occur less than 5 times or more than 100 times.

In [6]:
```
word_counts = counts.sum(axis=0)

counts = counts.loc[:,(word_counts >= 5) & (word_counts <= 100)]

counts.head()
```

Out[6]:

	as	you	good	what	de	amazon	cloud	at	google	back	...	uk	preview	compiler	man
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	1	0	0	0	0	0	...	0	0	0	

5 rows × 661 columns

Linear Regression

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

X_train, X_test, y_train, y_test = train_test_split(counts, submissions["upvot
es"], test_size=0.2, random_state=1)

lr = LinearRegression()
lr.fit(X_train, y_train)
predictions = lr.predict(X_test)

mse = mean_squared_error(predictions, y_test)
print(mse)
```

```
2651.1457056689683
```

This is a fairly large error. In this case, the mean number of upvotes is 10, and the standard deviation is 39.5. The square root of the MSE is 51.5. This means the average error is 51.5 upvotes away from the true value. This is higher than the standard deviation, so my predictions are far off-base. I'll implement a random forest model to see if using a more complex model will improve the accuracy.

In [8]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
rf.fit(X_train, y_train)
preds = rf.predict(X_test)

mse = mean_squared_error(preds, y_test)
print(mse)
```

```
c:\users\deand\appdata\local\programs\python\python37-32\lib\site-packages\sk
learn\ensemble\forest.py:245: FutureWarning: The default value of n_estimator
s will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

2392.1303731985718
```

By implementing a random forest, I reduced the error to 48.9 upvotes which is a small improvement but still a large error. I suspect there are some stopwords in my data set that can be removed and incease my model accuracy. I'll explore the columns in the dataset and make a list of stopwords I could remove.

In [9]:
```
counts.columns[600:]
```

Out[9]:
```
Index(['body', 'secure', 'brain', 'sell', 'workers', 'house', 'funny', 'cit
y',
       'jquery', 'claims', 'research', 'name', 'take', 'care', 'fun',
       'contest', '500', 'got', 'long', 'ui', 'powered', 'demo', 'stanford',
       'products', 'looking', 'clone', 'verizon', 'another', 'matter',
       'revenue', 'healthy', 'wins', 'hunt', 'c#', 'making', 'venture', 'wa
r',
       'see', 'led', '14', 'gates', 'warns', 'erlang', 'pirate', 'bay',
       'shoes', 'stream', 'economy', 'london', 'hit', 'obama', 'uk', 'previe
w',
       'compiler', 'manager', 'sharing', 'sale', 'competition', 'diet',
       'reasons', 'nike'],
      dtype='object')
```

In [10]:
```
#list of columns to remove
stops = ['as', 'you', 'what', 'de', 'at', 'back', 'an', 'from', 'via', 'into', 'or', 'it', 'using', 'but', 'part', '1',
        'get', 'after', 'his', 'three', 'us', 'why', 'that', 'can', 'may', 'this', 'my', 'i', 'by', 'them', 'some',
        'its', 'are', 'be', 'so', 'one', 'any', 'being', '4', '5', 'goes', 'me', 'we', '40', 'has', 'only', '|', 'if',
        'have', 'will', 'x', '8', 'did', 'could', 'isnt', 'through', 'ever', 'should', '3', 'even', 'word', 'they', 'come',
        'must', 'two', 'whats', 'who', 'lets', 'san', '20', 'other', 'there', 'tells', 'center', 'y', 'every', 'too',
        'know', 'put', 'ways', 'were', '100', '6', 'things', 'say', 'when', 'youre', 'head', 'before', 'made', 'right',
        'cant', 'makes', 'inside', 'thoughts', '18', 'let', 'take', '500', 'got', 'another', 'making', '14']

counts = counts.drop(stops, axis=1)
counts.shape
```

Out[10]:
```
(2800, 558)
```

Now that I've dropped columns that contain stopwords, I'll rerun the random forest model to see if the model's accuracy has improved.

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(counts, submissions["upvotes"], test_size=0.2, random_state=1)

lr = LinearRegression()
lr.fit(X_train, y_train)
predictions = lr.predict(X_test)

mse = mean_squared_error(predictions, y_test)
print(mse)
```

```
2361.019676848562
```

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(counts, submissions["upvotes"], test_size=0.2, random_state=1)

rf = RandomForestRegressor()
rf.fit(X_train, y_train)
preds = rf.predict(X_test)

mse = mean_squared_error(preds, y_test)
print(mse)
```

```
c:\users\deand\appdata\local\programs\python\python37-32\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

2061.6395419830546
```

Removing the stopwords only slightly improved the model's accuracy from an error of 51.1 upvotes to 46.3. This error is still large considering the mean number of upvotes is 10.

Headlines may not be the best feature to predict upvotes since it is not necessarily indicative of the post activity. Comments or views may be a better feature to use as well as time posted.