

NLP Project

June 10, 2018

1 Natural Language Processing Project

In this NLP project you will be attempting to classify Yelp Reviews into 1 star or 5 star categories based off the text content in the reviews.

We will use the [Yelp Review Data Set from Kaggle](#).

Each observation in this dataset is a review of a particular business by a particular user.

The "stars" column is the number of stars (1 through 5) assigned by the reviewer to the business. (Higher stars is better.) In other words, it is the rating of the business by the person who wrote the review.

The "cool" column is the number of "cool" votes this review received from other Yelp users.

All reviews start with 0 "cool" votes, and there is no limit to how many "cool" votes a review can receive. In other words, it is a rating of the review itself, not a rating of the business.

The "useful" and "funny" columns are similar to the "cool" column.

1.1 Imports

```
In [94]: import numpy as np
import pandas as pd
```

1.2 The Data

Read the yelp.csv file and set it as a dataframe called yelp.

```
In [95]: yelp = pd.read_csv('yelp.csv')
```

**** Check the head, info , and describe methods on yelp.****

```
In [96]: yelp.head()
```

```
Out[96]:
```

	business_id	date	review_id	stars	\
0	9yKzy9PApeiPP0UJEtnvkg	2011-01-26	fWKvX83p0-ka4JS3dc6E5A	5	
1	ZRJwVLyzEJq1VAihDhYiow	2011-07-27	IjZ33sJrzXqU-0X6U8NwyA	5	
2	6oRAC4uyJCsJl1X0WZpVSA	2012-06-14	IESLBzqUCLdSzSqm0eCSxQ	4	
3	_1QQZuf4zZ0yFCvXc0o6Vg	2010-05-27	G-WvGaISbqqqMHlNnByodA	5	
4	6ozycU1RpktNG2-1BroVtw	2012-01-05	1uJFq2r5QfJG_6ExMRCaGw	5	

	text	type	\
0	My wife took me here on my birthday for breakf...	review	

```

1 I have no idea why some people give bad review... review
2 love the gyro plate. Rice is so good and I als... review
3 Rosie, Dakota, and I LOVE Chaparral Dog Park!!... review
4 General Manager Scott Petello is a good egg!!!... review

```

		user_id	cool	useful	funny
0	rLtl8ZkDX5vH5nAx9C3q5Q		2	5	0
1	0a2KyEL0d3Yb1V6aivbIuQ		0	0	0
2	0hT2KtfLiobPvh6cDC8JQg		0	1	0
3	uZetl9T0NcR0G0yFfughhg		1	2	0
4	vYmM4KTsC8ZfQBg-j5MWkw		0	0	0

In [97]: yelp.info()

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
business_id    10000 non-null object
date           10000 non-null object
review_id      10000 non-null object
stars          10000 non-null int64
text           10000 non-null object
type           10000 non-null object
user_id        10000 non-null object
cool           10000 non-null int64
useful         10000 non-null int64
funny          10000 non-null int64
dtypes: int64(4), object(6)
memory usage: 781.3+ KB

```

In [99]: yelp.describe()

```

Out[99]:

```

	stars	cool	useful	funny
count	10000.000000	10000.000000	10000.000000	10000.000000
mean	3.777500	0.876800	1.409300	0.701300
std	1.214636	2.067861	2.336647	1.907942
min	1.000000	0.000000	0.000000	0.000000
25%	3.000000	0.000000	0.000000	0.000000
50%	4.000000	0.000000	1.000000	0.000000
75%	5.000000	1.000000	2.000000	1.000000
max	5.000000	77.000000	76.000000	57.000000

Create a new column called "text length" which is the number of words in the text column.

In [100]: yelp['text length'] = yelp['text'].apply(len)

2 EDA

Let's explore the data

2.1 Imports

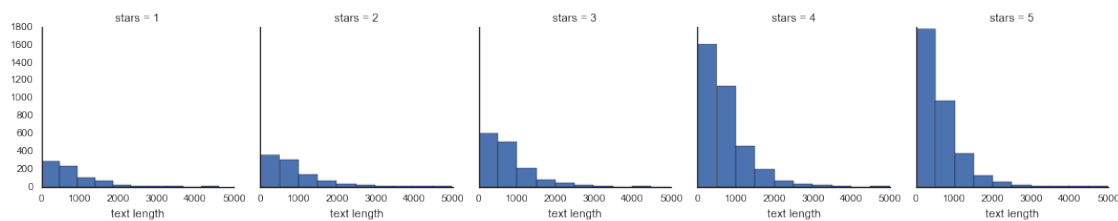
Import the data visualization libraries if you haven't done so already.

```
In [101]: import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('white')
%matplotlib inline
```

Use FacetGrid from the seaborn library to create a grid of 5 histograms of text length based off of the star ratings. Reference the seaborn documentation for hints on this

```
In [102]: g = sns.FacetGrid(yelp,col='stars')
g.map(plt.hist,'text length')
```

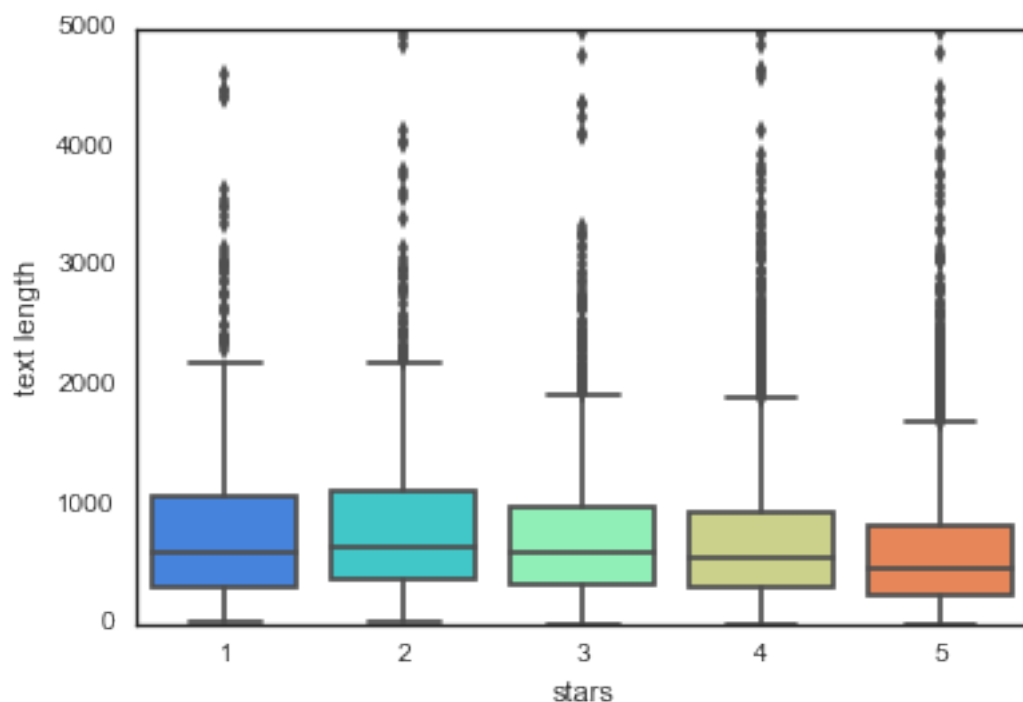
```
Out[102]: <seaborn.axisgrid.FacetGrid at 0x121e705f8>
```



Create a boxplot of text length for each star category.

```
In [103]: sns.boxplot(x='stars',y='text length',data=yelp,palette='rainbow')
```

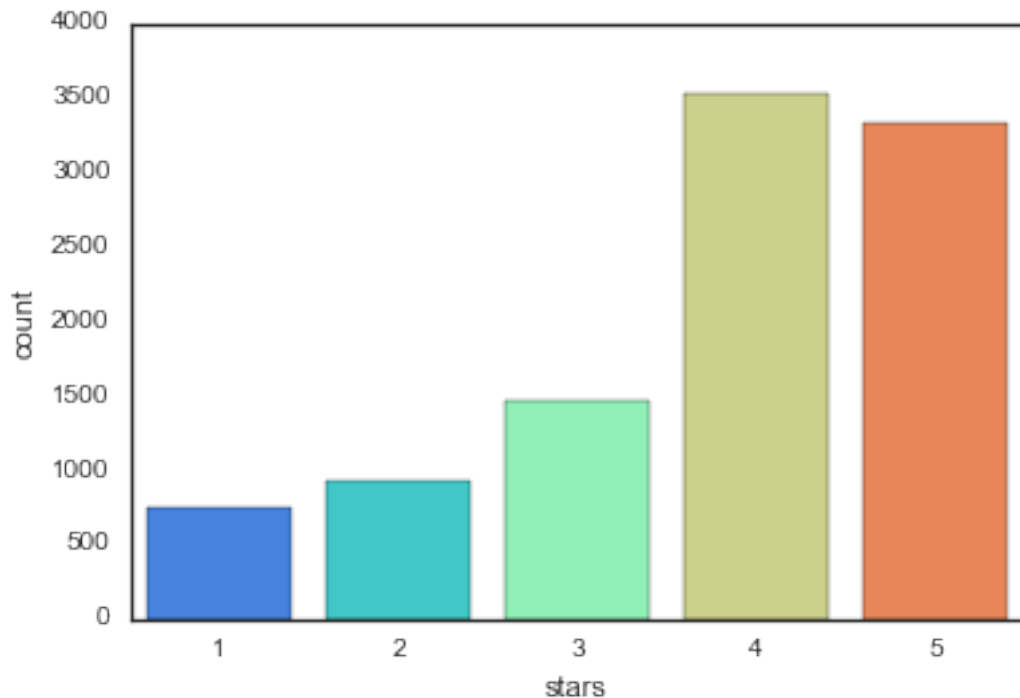
```
Out[103]: <matplotlib.axes._subplots.AxesSubplot at 0x121283470>
```



Create a countplot of the number of occurrences for each type of star rating.

```
In [104]: sns.countplot(x='stars',data=yelp,palette='rainbow')
```

```
Out[104]: <matplotlib.axes._subplots.AxesSubplot at 0x12578fc88>
```



** Use groupby to get the mean values of the numerical columns, you should be able to create this dataframe with the operation:**

```
In [105]: stars = yelp.groupby('stars').mean()  
stars
```

```
Out[105]:
```

	cool	useful	funny	text length
stars				
1	0.576769	1.604806	1.056075	826.515354
2	0.719525	1.563107	0.875944	842.256742
3	0.788501	1.306639	0.694730	758.498289
4	0.954623	1.395916	0.670448	712.923142
5	0.944261	1.381780	0.608631	624.999101

Use the corr() method on that groupby dataframe to produce this dataframe:

```
In [106]: stars.corr()
```

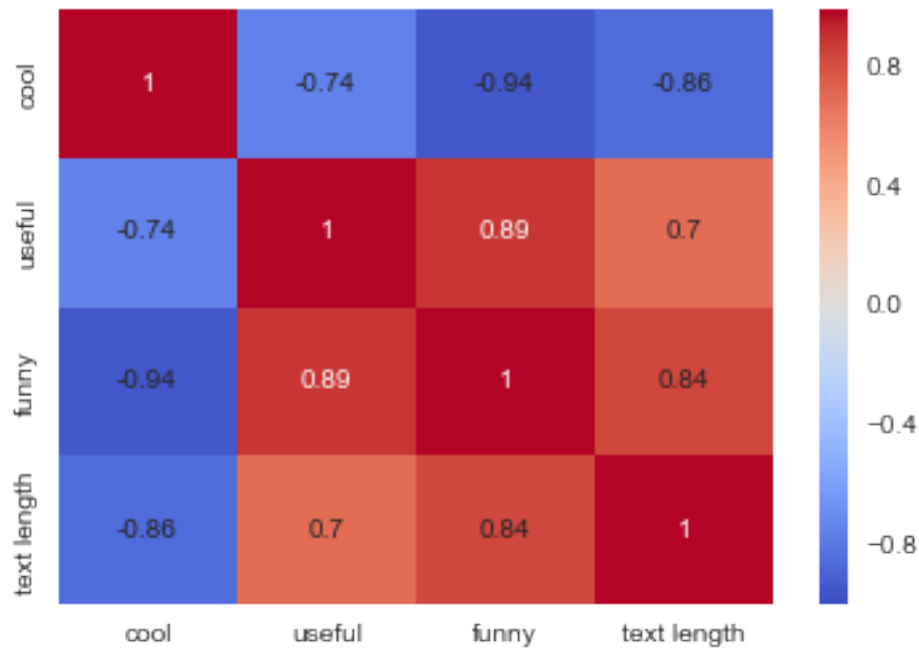
```
Out[106]:
```

	cool	useful	funny	text length
cool	1.000000	-0.743329	-0.944939	-0.857664
useful	-0.743329	1.000000	0.894506	0.699881
funny	-0.944939	0.894506	1.000000	0.843461
text length	-0.857664	0.699881	0.843461	1.000000

Then use seaborn to create a heatmap based off that `.corr()` dataframe:

```
In [38]: sns.heatmap(stars.corr(), cmap='coolwarm', annot=True)
```

```
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x120edb828>
```



2.2 NLP Classification Task

Create a dataframe called `yelp_class` that contains the columns of `yelp` dataframe but for only the 1 or 5 star reviews.

```
In [107]: yelp_class = yelp[(yelp.stars==1) | (yelp.stars==5)]
```

**** Create two objects X and y. X will be the 'text' column of `yelp_class` and y will be the 'stars' column of `yelp_class`. (Your features and target/labels)****

```
In [117]: X = yelp_class['text']
          y = yelp_class['stars']
```

Import `CountVectorizer` and create a `CountVectorizer` object.

```
In [118]: from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
```

**** Use the fit_transform method on the CountVectorizer object and pass in X (the 'text' column). Save this result by overwriting X.****

```
In [119]: X = cv.fit_transform(X)
```

2.3 Train Test Split

**** Use train_test_split to split up the data into X_train, X_test, y_train, y_test. Use test_size=0.3 and random_state=101 ****

```
In [120]: from sklearn.model_selection import train_test_split
```

```
In [121]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

2.4 Training a Model

Time to train a model!

**** Import MultinomialNB and create an instance of the estimator and call it nb ****

```
In [122]: from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
```

Now fit nb using the training data.

```
In [123]: nb.fit(X_train, y_train)
```

```
Out[123]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

2.5 Predictions and Evaluations

Time to see how our model did!

Use the predict method off of nb to predict labels from X_test.

```
In [124]: predictions = nb.predict(X_test)
```

**** Create a confusion matrix and classification report using these predictions and y_test ****

```
In [82]: from sklearn.metrics import confusion_matrix, classification_report
```

```
In [125]: print(confusion_matrix(y_test, predictions))
print('\n')
print(classification_report(y_test, predictions))
```

```
[[159  69]
 [ 22 976]]
```

```
precision    recall  f1-score   support
```

1	0.88	0.70	0.78	228
5	0.93	0.98	0.96	998
avg / total	0.92	0.93	0.92	1226

Great! Let's see what happens if we try to include TF-IDF to this process using a pipeline.

3 Using Text Processing

**** Import TfidfTransformer from sklearn. ****

```
In [155]: from sklearn.feature_extraction.text import TfidfTransformer
```

**** Import Pipeline from sklearn. ****

```
In [156]: from sklearn.pipeline import Pipeline
```

**** Now create a pipeline with the following steps:CountVectorizer(), TfidfTransformer(),MultinomialNB()****

```
In [157]: pipeline = Pipeline([
    ('bow', CountVectorizer()), # strings to token integer counts
    ('tfidf', TfidfTransformer()), # integer counts to weighted TF-IDF scores
    ('classifier', MultinomialNB()), # train on TF-IDF vectors w/ Naive Bayes class
])
```

3.1 Using the Pipeline

Time to use the pipeline! Remember this pipeline has all your pre-process steps in it already, meaning we'll need to re-split the original data (Remember that we overwrote X as the CountVectorized version. What we need is just the text

3.1.1 Train Test Split

Redo the train test split on the yelp_class object.

```
In [158]: X = yelp_class['text']
y = yelp_class['stars']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
```

Now fit the pipeline to the training data. Remember you can't use the same training data as last time because that data has already been vectorized. We need to pass in just the text and labels

```
In [159]: # May take some time
pipeline.fit(X_train,y_train)
```

```
Out[159]: Pipeline(steps=[('bow', CountVectorizer(analyzer='word', binary=False, decode_error=
dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
lowercase=True, max_df=1.0, max_features=None, min_df=1,
ngram_range=(1, 1), preprocessor=None, stop_words=None,
strip_...f=False, use_idf=True)), ('classifier', MultinomialNB(alpha=1.0, cl
```

3.1.2 Predictions and Evaluation

**** Now use the pipeline to predict from the X_test and create a classification report and confusion matrix. You should notice strange results.****

```
In [153]: predictions = pipeline.predict(X_test)
```

```
In [154]: print(confusion_matrix(y_test,predictions))
          print(classification_report(y_test,predictions))
```

```
[[ 0 228]
 [ 0 998]]
```

	precision	recall	f1-score	support
1	0.00	0.00	0.00	228
5	0.81	1.00	0.90	998
avg / total	0.66	0.81	0.73	1226

```
/Users/marci/anaconda/lib/python3.5/site-packages/sklearn/metrics/classification.py:1074: Under
'precision', 'predicted', average, warn_for)
```

Looks like Tf-Idf actually made things worse!