ML1010 - Yelp Reviews

Machine Learning Pokémon: Durai Nachiappan, Iman Lau, Shabeeth Syed, Lijuan Yang, Mohammad Islam

Introduction

Our Jupyter Notebooks and code are available at https://github.com/stellarclass/ML1010-Yelp-Project.

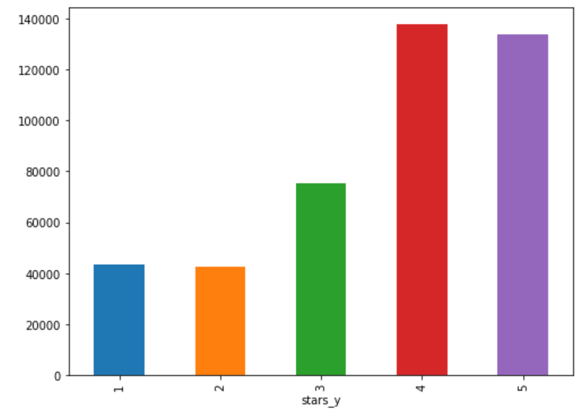
Our project uses data from Yelp, an online review site where users can rate various businesses. Most often, it is used for restaurants, although any business can be rated, from hotels to doctors. Yelp is widely used in North America but is available in numerous countries worldwide. This data is available through Kaggle, provided by Yelp.

Using natural language processing, we can take a large amount of unstructured review data and gather insights from the reviews. If one were to open a business in the city of Toronto, it would be useful to know what reviews tend to talk about. This would let a business owner know what to focus on when creating and running their business.

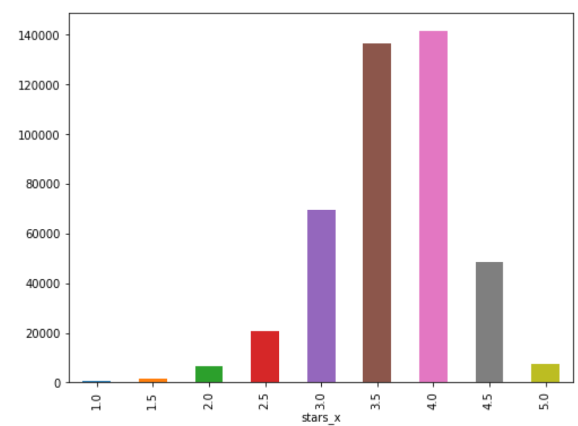
With that in mind, we created some models that would assist a Toronto business owner in this aspect. Our approaches were topic modelling and sentiment analysis. These machine learning approaches would help a business owner quickly find and understand relevant reviews in order to better their business or create a new one.

Exploratory Data Analysis

We briefly explored the data to see what it looked like. We saw that most reviews actually are 4 and 5 star reviews - almost twice as much as 1 - 3 star reviews combined.



We also can see that most businesses are about 3.5 to 4 stars.



This means that we could consider 3.5 to be "good."

Methodology

As mentioned in the introduction, we used topic modelling and sentiment analysis for our project. As this course focuses on Python, that was the language of choice. We used packages such as numpy, pandas, gensim, and scikit-learn.

Preprocessing

A fair amount of preprocessing was done to the dataset. Generally, natural language corpora require a great deal of preprocessing in order to be fed into machine learning models.

The Yelp dataset includes data from 11 metropolitan areas across 4 countries. As we only wanted to know about Toronto businesses, we needed to filter out non-Toronto data.

Due to memory limitations of our personal computers and the large size of the data, it was necessary to create a way to load the data into Python in stages. A script was written to read in the JSON file character by character, which also helped to removed data that was improperly formatted.

Once the data was written in, two pandas dataframes were joined together and then filtered for Toronto businesses.

After this, we could preprocess the Yelp review text to prepare it for modelling. The review text was normalize through removing stop words, forcing all words into lowercase, and removing non-alphanumeric characters. Then we could perform some feature engineering.

Feature Engineering

We used Bag of Words, Bag of n-grams (2-word phrases) and TF-IDF techniques as part of Feature engineering. These techniques helped us to identify the unique words in the reviews provided for Toronto's businesses.

In topic modelling, we created Bigrams and Trigrams. Gensim's Phrases model can build and implement the bigrams, trigrams, quadgrams and more. The two important arguments to Phrases are min_count and threshold. The higher the values of these parameters, the harder it is for words to be combined to bigrams. Once the bigrams model is ready, stopwords are removed and bigrams are created and lemmatized. The two main inputs to the LDA topic model are the dictionary and the corpus, which also were created.

Bag of Words was used in the sentiment analysis for the classifiers and ensemble methods. Neural networks are required to have the corpus converted into cleaned tokens and then encoded. Best performance happens when these tokens are also set to the same length, so shorter tokens are padded with 0's.

Topic Modelling

Latent Dirichlet Allocation (LDA) from Gensim package was used to perform Topic Modeling. Latent Dirichlet Allocation is the most common technique used in Topic Modeling, it is a generalized form of probabilistic latent semantic analysis (PLSA). In a nutshell, LDA considers each document as a collection of topics in a certain proportion, and each topic as a collection of keywords in a certain proportion. Once the algorithm is provided with the number of topics, all it does it to rearrange the topics distribution within the documents and keywords distribution within the topics to obtain a good composition of topic-keywords distribution. A topic is nothing but a collection of dominant keywords that are typical representatives. A Topic is identified by looking at the keywords.

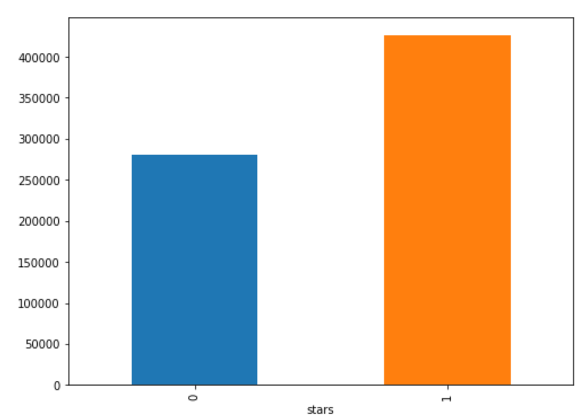
In addition to the corpus and dictionary,we need to provide the number of topics as well. Alpha and eta are hyperparameters that affect sparsity of the topics. According to the Gensim docs, both defaults to 1.0/num_topics prior. Chunksize is the number of documents to be used in each training chunk,and update_every determines how often the model parameters should be updated and passes is the total number of training passes.

Sentiment Analysis

A few different approaches were used to perform sentiment analysis. For standard classifiers and ensemble methods, we used and compared Naive Bayes, Logistic Regression, Stochastic Gradient Descent, Random Forest, and XGBoost.

Neural networks are systems that take input and process them through nodes and layers. The neural networks we tried, we used GLoVE word embeddings. We used the keras package for the neural networks.

In order to classify the reviews into positive or negative, reviews with a star rating of 3 or lower were considered negative with star ratings of 4 or 5 were positive. We can see the new distribution as such:



This is a roughly even distribution but is skewed slightly towards positive reviews.

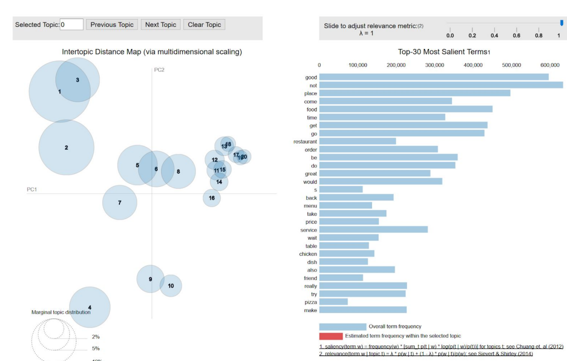
Results

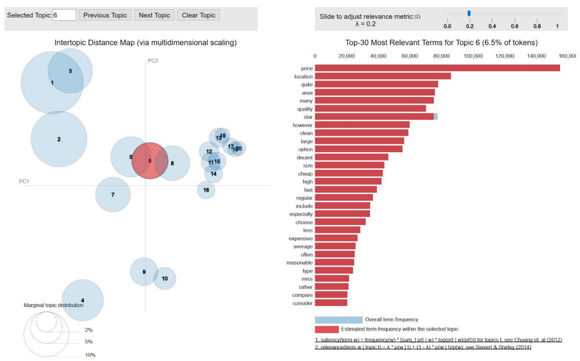
Topic Modelling

Model perplexity and topic coherence provide a convenient measure to judge how good a given topic model is. Topic Coherence is a measure used to evaluate topic models.Each such generated topic consists of words, and the topic coherence is applied to the top N words from the topic. It is defined as the average / median of the pairwise word-similarity scores of the words in the topic (e.g. PMI). A good model will generate coherent topics, and can be described by a short label, therefore this is what the topic coherence measure should capture

```
Perplexity:  -7.610246942214927

Coherence Score:
0.4172512043387064
```

We can also visualize the topics with an interactive chart (available in the Juptyer Notebooks).





Sentiment Analysis

We can see various metrics of our sentiment analysis models in the following table:

Model Name	Accuracy	F1	Precision	Recall
Naive Bayes	0.8496	0.8817	0.8400	0.9278
Logistic Regression	0.8749	0.8980	0.8845	0.9119
Stochastic Gradient Descent				
Random Forest				

Unfortunately the remaining two models did not finish in time for this report.

We can see the accuracy of the neural networks in the following chart:

Neural Network Type	Accuracy
Convolutional Neural Network	0.39591462211969713
Long Short Term Modelr	0.39591462211969713
Recurrent Convolutional Neural Network	0.39591462211969713

As we can see, the neural networks don't have a noticeable improvement over regular classification systems. We can also see that there's a weird issue with the neural networks where the accuracy is the same for all the types that were attempted.

Discussion

We can see that overall, the most relevant topic for all reviews was "good." Since they are reviews, this makes a lot of sense - the most important thing in a review is to know whether or not a business was good. Many reviews on Yelp are restaurant/food based, and these also show up as common topic themes.

When we look closer into the topic clusters, for example topic 6, we can see that there's more concentration in different topics. In cluster 6, people are very concerned about price and location.

In the sentiment analysis, we can see that, without tuning, the standard classifiers perform admirably compared with untuned single-epoch neural networks.

Conclusion and Future Work

We can draw the following conclusions from our work:

- natural language processing is memory intensive
- Python is slow

For future work, we propose:

- better cleaning of data and stemming - some oddities showed up in the data and would benefit from more cleaning

- creating an interactive map to explore the topics and sentiments when attached to their respective businesses
- creating a dashboard/website where people can explore our interactive topic model chart
- fixing neural networks
- more layers and epochs in neural networks
- more hyperparameter tuning
- comparing sentiment analysis ROC curves

References

https://www.kaggle.com/yelp-dataset/yelp-dataset/home

http://blog.conceptnet.io/posts/2017/how-to-make-a-racist-ai-without-really-trying/

https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f

https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/

https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/

https://rare-technologies.com/what-is-topic-coherence/

Appendix

Notes:

- Jupyter Notebooks were saved as PDFs and concatenated into this report as the appendix - they may or may not have titles but are named properly in the GitHub repo
- Some code is included that was ultimately not finished in time for this report (due to model run-time)

```python
import json
import pandas as pd

def fetch_json(file):
    while True:
        json = ""
        while len(json) == 0 or json[-1] != "}":
            ch = file.read(1)
            if not ch:
                yield None
            json += ch

        # Calling strip because my file has newline characters between the
        # json blobs
        yield json.strip()


business = pd.read_json('C:/Users/Peter Dell/Documents/ML1010-Yelp-Project/JSON-data/yelp_academic_
ontario = business.loc[business['state'] == 'ON']

filename = 'C:/Users/Peter Dell/Documents/ML1010-Yelp-Project/JSON-data/yelp_academic_dataset_revie
d = []

f = open(filename, "r", encoding = "utf-8")

for j in fetch_json(f):
    if j == None: break
    try:
        p = json.loads(j)
        d.append(p)
    except ValueError:
        print("Garbage text, ignore this line")
        pass

f.close()

df = pd.DataFrame(d)

subset = pd.merge(ontario, df, on="business_id", how="inner")

subset.to_csv("subset.csv", index = False)
```

Feature Engineering

This section will cover the following types of features for the Yelp reviews:

1. Bag of Words
2. Bag of N-Grams
3. TF-IDF (term frequency over inverse document frequency)

```
import pandas as pd
import numpy as np
import re
import nltk
```

The corpus or the reviews were extracted from the Yelp review dataset using pandas

```
corpus_df = pd.read_csv('C:/Users/Peter Dell/Documents/ML1010-Yelp-Project/data/subset.csv'
corpus = corpus_df['text']
corpus.head()
```

```
0    Hallelujah! I FINALLY FOUND IT! The frozen yog...
1    I drop by BnC on a weekly basis to pick up my ...
2    My personally experience here wasn't the best,...
3    37 °C = 98.6°F\r\nKoreatown establisments disp...
4    My husband & I visited Toronto from the U.S. f...
Name: text, dtype: object
```

Text pre-processing

As part of Text pre-processing we removed the special characters, whitespaces and numbers and, converted all the text to lower case.

```
wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I)
    # doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    doc = ''.join(i for i in doc if not i.isdigit())
    return doc
```

```
normalize_corpus = np.vectorize(normalize_document)
```

```
norm_corpus = normalize_corpus(corpus)
norm_corpus
```

1. Bag of Words Model

```
We created the Bag of Words model to determine the unique words in each document along with
```

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(norm_corpus)
cv_matrix = cv_matrix.toarray()
cv_matrix
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

Thus you can see that our documents have been converted into numeric vectors such that each document is represented by one vector (row) in the above feature matrix. The following code will help represent this in a more easy to understand format.

```
# get all unique words in the corpus
vocab = cv.get_feature_names()
vocab
```

```
['______',
 '_______',
 '_______________',
 '____berto',
 '_accommodating',
 '_c',
 '_finally_',
 '_gyibeahdfylsszc_g',
 '_lozhqednolhvbg',
 '_reasonable',
 '_she',
 '_third_',
 '_us_',
 '_very',
 '_xhxtuykqnyphmylm',
 'aa',
 'aaa',
 'aaaaaalright',
 'aaaamazing',
 'aaammmazzing',
 'aaand',
 'aah',
 'aand',
 'aaron',
 'aarp',
 'ab',
 'aback',
 'abacus',
 'abandon',
 'abandoned',
 'abandoning',
 'abba',
 'abbaye',
 'abbey',
 'abbreviate',
 'abbreviated',
 'abbreviations',
 'abby',
 'abc',
 'abdomen',
 'abe',
 'aberration',
```

```
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab)
```

	______	_______	_______________	____berto	_accommodating	_c	_finally_	_g
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	
12	0	0	0	0	0	0	0	
13	0	0	0	0	0	0	0	
14	0	0	0	0	0	0	0	
15	0	0	0	0	0	0	0	
16	0	0	0	0	0	0	0	
17	0	0	0	0	0	0	0	
18	0	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	
21	0	0	0	0	0	0	0	
22	0	0	0	0	0	0	0	
23	0	0	0	0	0	0	0	
24	0	0	0	0	0	0	0	

2. Bag of N-Grams Model

```
We created the Bag of bi-grams and tri-grams to look at the 2-word and 3-word strings used
```

```
bv = CountVectorizer(ngram_range=(2,2))
```

```
bv_matrix = bv.fit_transform(norm_corpus)

bv_matrix = np.asarray(bv_matrix)
vocab = bv.get_feature_names()
# pd.DataFrame(bv_matrix, columns=vocab)
vocab
```

```
['______ ordered',
 '_______ oakland',
 '_______________ update',
 '____berto matter',
 '_accommodating evening',
 '_finally_ found',
 '_gyibeahdfylsszc_g adventures',
 '_lozhqednolhybg http',
```

```
bv = CountVectorizer(ngram_range=(3,3))
bv_matrix = bv.fit_transform(norm_corpus)

bv_matrix = np.asarray(bv_matrix)
vocab = bv.get_feature_names()
vocab
```

```
['______ ordered chicken',
 '_______ oakland coliseum',
 '_______________ update first',
 '____berto matter basically',
 '_accommodating evening appointments',
 '_finally_ found place',
 '_gyibeahdfylsszc_g adventures phoenix',
 '_lozhqednolhvbg http www',
 '_reasonable amount time',
 '_she listens every',
 '_she pretty busy',
 '_third_ visit since',
 '_us_ going wonder',
 '_very friendly _accommodating',
 '_xhxtuykqnyphmylm mqg dessert',
 'aa accessories fab',
```

3. TF-IDF Model

```
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(norm_corpus)
tv_matrix = tv_matrix.toarray()

vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)
```

	______	_______	_______________	____berto	_accommodating	_c	_finally_	_
0	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	0.0	
2	0.0	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	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

The TF-IDF based feature vectors for each of our text documents show scaled and normalized values as compared to the raw Bag of Words model values.

18	0.0	0.0	0.0	0.0	0.0	0.0	0.0
19	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	0.0	0.0	0.0	0.0	0.0	0.0
21	0.0	0.0	0.0	0.0	0.0	0.0	0.0
22	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26	0.0	0.0	0.0	0.0	0.0	0.0	0.0
27	0.0	0.0	0.0	0.0	0.0	0.0	0.0
28	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29	0.0	0.0	0.0	0.0	0.0	0.0	0.0

...	...	...	...	...	...	...	...

```
import sys
import os


current_work_directory = os.getcwd()    # Return a string representing the current working
print('Current work directory: {}'.format(current_work_directory))
# Make sure it's an absolute path.
abs_work_directory = os.path.abspath(current_work_directory)
print('Current work directory (full path): {}'.format(abs_work_directory))
print()

filename = 'subset.csv'
# Check whether file exists.
if not os.path.isfile(filename):
    # Stop with leaving a note to the user.
    print('It seems file "{}" not exists in directory: "{}"'.format(filename, current_work_
    sys.exit(1)

import csv

with open('subset.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        print(row)
```

```
Current work directory: C:\Users\shabe
Current work directory (full path): C:\Users\shabe

['address', 'attributes', 'business_id', 'categories', 'city', 'hours', 'is_open', '
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['4568 Highway 7 E', "{'GoodForKids': 'True', 'NoiseLevel': 'loud', 'RestaurantsAtti
['4568 Highway 7 E', "{'GoodForKids': 'True', 'NoiseLevel': 'loud', 'RestaurantsAtti
['4568 Highway 7 E', "{'GoodForKids': 'True', 'NoiseLevel': 'loud', 'RestaurantsAtti
['4568 Highway 7 E', "{'GoodForKids': 'True', 'NoiseLevel': 'loud', 'RestaurantsAtti
['4568 Highway 7 E', "{'GoodForKids': 'True', 'NoiseLevel': 'loud', 'RestaurantsAtti
```

```python
import pandas as pd
from pandas import DataFrame

ReadCsv = pd.read_csv (r'C:\Users\shabe\subset.csv')

df = DataFrame(ReadCsv,columns=['address', 'attributes', 'business_id', 'categories', 'city

print (df)


from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

no_features = 1000

# NMF is able to use tf-idf
tfidf_vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, max_features=no_features, stop_wo
tfidf = tfidf_vectorizer.fit_transform(df['text'])
tfidf_feature_names = tfidf_vectorizer.get_feature_names()

# LDA can only use raw term counts for LDA because it is a probabilistic graphical model
tf_vectorizer = CountVectorizer(max_df=0.95, min_df=2, max_features=no_features, stop_words
tf = tf_vectorizer.fit_transform(df['text'])
tf_feature_names = tf_vectorizer.get_feature_names()
```

```
                                    address  \
0                          631 Bloor St W
1                          631 Bloor St W
2                          631 Bloor St W
3                          631 Bloor St W
4                          631 Bloor St W
5                          631 Bloor St W
6                          631 Bloor St W
7                3417 Derry Road E, Unit 103
8                3417 Derry Road E, Unit 103
9                3417 Derry Road E, Unit 103
10               3417 Derry Road E, Unit 103
11               3417 Derry Road E, Unit 103
12               3417 Derry Road E, Unit 103
13               3417 Derry Road E, Unit 103
14                       4568 Highway 7 E
15                       4568 Highway 7 E
16                       4568 Highway 7 E
17                       4568 Highway 7 E
18                       4568 Highway 7 E
19                       4568 Highway 7 E
20                       4568 Highway 7 E
21                       4568 Highway 7 E
22                       4568 Highway 7 E
23                       4568 Highway 7 E
24                       4568 Highway 7 E
25                       4568 Highway 7 E
26                     595 Markham Street
27                     595 Markham Street
28                     595 Markham Street
29                     595 Markham Street
...                                   ...
706701                   3199 Dufferin St
706702                   3199 Dufferin St
706703                   3199 Dufferin St
706704                   3199 Dufferin St
706705                   3199 Dufferin St
706706                   3199 Dufferin St
706707                   3199 Dufferin St
706708                   3199 Dufferin St
706709                   3199 Dufferin St
706710                   3199 Dufferin St
706711                2215 Queen Street E
706712                2215 Queen Street E
706713                2215 Queen Street E
706714       1402 Queen Street E, Suite D
```

```python
from sklearn.decomposition import NMF, LatentDirichletAllocation

no_topics = 20

# Run NMF
nmf = NMF(n_components=no_topics, random_state=1, alpha=.1, l1_ratio=.5, init='nndsvd').fit

# Run LDA
lda = LatentDirichletAllocation(n_topics=no_topics, max_iter=5, learning_method='online', l
```

```
C:\Users\shabe\Anaconda3\lib\site-packages\sklearn\decomposition\online_lda.py:294:
  DeprecationWarning)
```

```
def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print ("Topic %d:" % (topic_idx))
        print (" ".join([feature_names[i]
                        for i in topic.argsort()[:-no_top_words - 1:-1]]))

no_top_words = 10
display_topics(nmf, tfidf_feature_names, no_top_words)
display_topics(lda, tf_feature_names, no_top_words)
```

```
Topic 0:
time order minutes asked got said told service wait didn
Topic 1:
chicken fried rice sauce jerk curry butter spicy wings ordered
Topic 2:
great service atmosphere selection spot awesome beer amazing prices patio
Topic 3:
food service restaurant quality excellent fast price better chinese indian
Topic 4:
sushi sashimi rolls roll salmon fresh ayce fish quality japanese
Topic 5:
good pretty service price overall bit prices selection decent nice
```

INTRODUCTION

One of the primary applications of natural language processing is to automatically extract what topics people are discussing from large volumes of text. Some examples of large text could be feeds from social media, customer reviews of hotels, movies, etc, user feedbacks, news stories, e-mails of customer complaints etc. Knowing what people are talking about and understanding their problems and opinions is highly valuable to businesses, administrators, political campaigns. It is really hard to manually read through such large volumes and compile the topics, and we need an algorithm that can read through the text doucments and output the topics discussed. I used the Latent Dirichlet Allocation (LDA) from Gensim package to perform Topic Modeling. Latent Dirichlet Allocation is the most common technique used in Topic Modeling, it is a generalized form of probabilistic latent semantic analysis (PLSA). In a nutshell, LDA considers each document as a collection of topics in a certain proportion, and each topic as a collection of keywords in a certain proportion. Once the algorithm is provided with the number of topics, all it does it to rearrange the topics distribution within the documents and keywords distribution within the topics to obtain a good composition of topic-keywords distribution. A topic is nothing but a collection of dominant keywords that are typical representatives. A Topic is identified by looking at the keywords.

We will need the stopwords from NLTK and spacy's en model for text pre-processing. We will be using the spacy model for lemmatization. Lemmatization is nconverting a word to its root word.

The core packages used are re, gensim, spacy and pyLDAvis. Besides this we will also using matplotlib, numpy and pandas for data handling and visualization.

```
Topic 0:
```

```
import nltk; nltk.download('stopwords')

import re
import numpy as np
import pandas as pd
from pprint import pprint

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# spacy for lemmatization
import spacy

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
```

```
import matplotlib.pyplot as plt
%matplotlib inline

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERROR

import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\shabe\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
C:\Users\shabe\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detect
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

We will be using Yelp reviews from Toronto, and see what topics are within those reviews. Pandas was used to import the data.

```
# NLTK Stop words
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])
```

```
# Import Dataset

import pandas as pd
from pandas import DataFrame

ReadCsv = pd.read_csv (r'C:\Users\shabe\subset.csv')

df = DataFrame(ReadCsv,columns=['address', 'attributes', 'business_id', 'categories', 'city

#print (df)

#print(df.target_names.unique())
#df.head()
```

Next, we remove unwanted spaces and characters using regular expression. Below is the code to do it. The text still looks messy even after removing unwanted characters and extra spaces. It is not ready for the LDA to consume. We need to break down each sentence into a list of words through tokenization, while clearing up all the messy text in the process. We will use Gensim's simple_preprocess is great for this, and we removed puncuation as well.

```
# Convert to list
data = df.text.values.tolist()

# Remove Emails
data = [re.sub('\S*@\S*\s?', '', sent) for sent in data]

# Remove new line characters
data = [re.sub('\s+', ' ', sent) for sent in data]

# Remove distracting single quotes
data = [re.sub("\'", "", sent) for sent in data]

pprint(data[:1])
```

```
['Hallelujah! I FINALLY FOUND IT! The frozen yogurt that launched the Red '
 'Mango and Pinkberry craze in the States. (Google it.) The Canadian '
 'incarnation goes by the name Yogoberri and I discovered it inside a tiny '
 'Korean bakery along Bloor Streets K-town. For the uninitiated, this frozen '
 'yogurt is more tart and less sweet than the TCBY kind. You can have plain '
 'vanilla yogurt but its all about the toppings; fresh fruit, nuts, '
 'cereal...weird-looking powders I never tried. A small (5 oz.) is $2.97 + 95 '
 'cents per topping. Medium (8 oz.) including three toppings is $4.99. I used '
 'to eat this frozen yogurt all the time when I lived in Korea and I was '
 'practically weeping with joy when I was reunited with it today. Shameless '
 'plea: Go eat lots of it so the chain will multiply and open a branch near my '
 'home. THANKS! (FYI, the 5 stars is for the yogurt. I havent tried anything '
 'else at the bakery.) **ETA: Dear Fro Yo Gods, Thanks for opening up '
 'Blushberry closer to my home. xoxo, susan c.**']
```

```
def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))  # deacc=True remo

data_words = list(sent_to_words(data))

print(data_words[:1])
```

```
[['hallelujah', 'finally', 'found', 'it', 'the', 'frozen', 'yogurt', 'that', 'launch
```

After tokenization and removing puncuation, we created Bigrams and Trigrams. Bigrams are two words frequently occurring together in the document. Trigrams are 3 words frequently occurring.

Gensim's Phrases model can build and implement the bigrams, trigrams, quadgrams and more. The two important arguments to Phrases are min_count and threshold. The higher the values of these param, the harder it is for words to be combined to bigrams. once the bigrams model is ready, we are going to define the functions to remove the stopwords, make bigrams and lemmatization and call them. We would also need to create the inputs for LDA. The two main inputs to the LDA topic model are the dictionary(id2word) and the corpus. These are created later down in the code.

```
# Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100) # higher threshold f
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)

# See trigram example
print(trigram_mod[bigram_mod[data_words[0]]])
```

```
C:\Users\shabe\Anaconda3\lib\site-packages\gensim\models\phrases.py:494: UserWarning
  warnings.warn("For a faster implementation, use the gensim.models.phrases.Phraser
['hallelujah', 'finally', 'found', 'it', 'the', 'frozen_yogurt', 'that', 'launched',
```

```
# Define functions for stopwords, bigrams, trigrams and lemmatization
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for do

def make_bigrams(texts):
```

```
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

```
# Remove Stop Words
data_words_nostops = remove_stopwords(data_words)

# Form Bigrams
data_words_bigrams = make_bigrams(data_words_nostops)

# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
# python3 -m spacy download en
nlp = spacy.load('en', disable=['parser', 'ner'])

# Do lemmatization keeping only noun, adj, vb, adv
data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=['NOUN', 'ADJ', 'VERB',

print(data_lemmatized[:1])
```

```
[['finally', 'find', 'frozen_yogurt', 'launch', 'red', 'mango', 'pinkberry', 'craze'
```

This is where the inputs for LDA is created. Gensim creates a unique id for each word in the document. The produced corpus shown below is a mapping of (word_id, word_frequency). This is used as the input by the LDA model.

If you want to see what word a given id corresponds to, pass the id as a key to the dictionary. or look at the Human readable format.

```
# Create Dictionary
id2word = corpora.Dictionary(data_lemmatized)

# Create Corpus
texts = data_lemmatized

# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]

# View
print(corpus[:1])
```

```
[[(0, 1), (1, 2), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (1
```

```
id2word[0]
```

```
'anything'
```

```
# Human readable format of corpus (term-frequency)
[[(id2word[id], freq) for id, freq in cp] for cp in corpus[:1]]
```

```
[[('anything', 1),
  ('bakery', 2),
  ('bloor', 1),
  ('blushberry', 1),
  ('branch', 1),
  ('canadian', 1),
  ('cent', 1),
  ('cereal', 1),
  ('chain', 1),
  ('close', 1),
  ('craze', 1),
  ('dear_fro', 1),
  ('discover', 1),
  ('eat', 2),
  ('else', 1),
  ('eta', 1),
  ('finally', 1),
  ('find', 1),
  ('fresh', 1),
  ('frozen_yogurt', 3),
  ('fruit', 1),
  ('fyi', 1),
  ('go', 2),
  ('god', 1),
  ('have', 1),
  ('home', 2),
  ('incarnation', 1),
  ('include', 1),
  ('joy', 1),
  ('kind', 1),
  ('korea', 1),
  ('korean', 1),
  ('launch', 1),
  ('less', 1),
  ('live', 1),
  ('look', 1),
  ('lot', 1),
  ('mango', 1),
  ('medium', 1),
  ('name', 1),
  ('never', 1),
  ('not', 1),
  ('nut', 1),
  ('open', 2),
  ('pinkberry', 1),
  ('plain', 1),
  ('plea', 1),
  ('powder', 1),
  ('practically', 1),
  ('red', 1),
  ('reunite', 1),
  ('shameless', 1),
  ('small', 1),
  ('star', 1),
  ('state', 1),
  ('street', 1),
  ('susan', 1),
```

```
        ('sweet', 1),
```

We have everything required to train the LDA model. In addition to the corpus and dictionary,we need to provide the number of topics as well. alpha and eta are hyperparameters that affect sparsity of the topics. According to the Gensim docs, both defaults to 1.0/num_topics prior. Chunksize is the number of documents to be used in each training chunk,and update_every determines how often the model parameters should be updated and passes is the total number of training passes.

```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                           id2word=id2word,
                                           num_topics=20,
                                           random_state=100,
                                           update_every=1,
                                           chunksize=100,
                                           passes=10,
                                           alpha='auto',
                                           per_word_topics=True)
```

The above LDA model is built with 20 different topics where each topic is a combination of keywords and each keyword contributes a certain weightage to the topic. We can see the keywords for each topic and the weightage(importance) of each keyword using lda_model.print_topics() as shown next. Topic 0 is a represented as 0.024*"day" + 0.023*"ask" + 0.020*"work" + 0.020*"tell" + 0.019*"need" + '0.018*"say" + 0.018*"never" + 0.017*"hour" + 0.016*"customer" + '0.016*"leave. It means the top 10 keywords that contribute to this topic are: ‘day’, ‘ask’, ‘work’.. and so on and the weight of ‘day’ on topic 0 is 0.024. The weights reflect how important a keyword is to that topic.

```
# Print the Keyword in the 10 topics
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0,
  '0.024*"day" + 0.023*"ask" + 0.020*"work" + 0.020*"tell" + 0.019*"need" + '
  '0.018*"say" + 0.018*"never" + 0.017*"hour" + 0.016*"customer" + '
  '0.016*"leave"'),
 (1,
  '0.040*"chicken" + 0.032*"taste" + 0.030*"fry" + 0.027*"sauce" + '
  '0.021*"soup" + 0.020*"noodle" + 0.019*"pork" + 0.018*"hot" + 0.018*"meat" + '
  '0.018*"side"'),
 (2,
  '0.087*"raman" + 0.078*"store" + 0.065*"item" + 0.051*"sandwich" + '
  '0.050*"shop" + 0.023*"sell" + 0.022*"product" + 0.022*"bean" + '
  '0.020*"vegetarian" + 0.020*"vegan"'),
 (3,
  '0.033*"fancy" + 0.023*"event" + 0.019*"chat" + 0.017*"member" + '
  '0.017*"chill" + 0.016*"moment" + 0.016*"shopping" + 0.015*"management" + '
  '0.013*"push" + 0.013*"team"'),
 (4,
  '0.204*"s" + 0.076*"there" + 0.063*"that" + 0.017*"patient" + 0.016*"brand" '
  '+ 0.015*"office" + 0.015*"mistake" + 0.013*"park" + 0.012*"what" + '
  '0.012*"depend"'),
 (5,
  '0.080*"roll" + 0.077*"sushi" + 0.063*"fish" + 0.037*"thick" + '
  '0.031*"salmon" + 0.027*"piece" + 0.027*"thin" + 0.018*"easily" + '
  '0.016*"addition" + 0.016*"basic"'),
 (6,
  '0.079*"not" + 0.054*"get" + 0.053*"go" + 0.045*"be" + 0.044*"do" + '
  '0.040*"would" + 0.026*"make" + 0.023*"have" + 0.019*"look" + 0.017*"want"'),
 (7,
  '0.054*"room" + 0.030*"parking" + 0.024*"class" + 0.022*"drive" + '
  '0.019*"fix" + 0.016*"wall" + 0.016*"complaint" + 0.014*"schnitzel" + '
  '0.013*"tire" + 0.012*"crazy"'),
 (8,
  '0.113*"pizza" + 0.108*"cheese" + 0.061*"bread" + 0.046*"bowl" + '
  '0.035*"topping" + 0.033*"slice" + 0.032*"tomato" + 0.022*"avocado" + '
  '0.022*"waffle" + 0.019*"burrito"'),
```

```
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the mode

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized, dictionary=id2
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Perplexity:  -7.610246942214927

Coherence Score:  0.4172512043387064
```

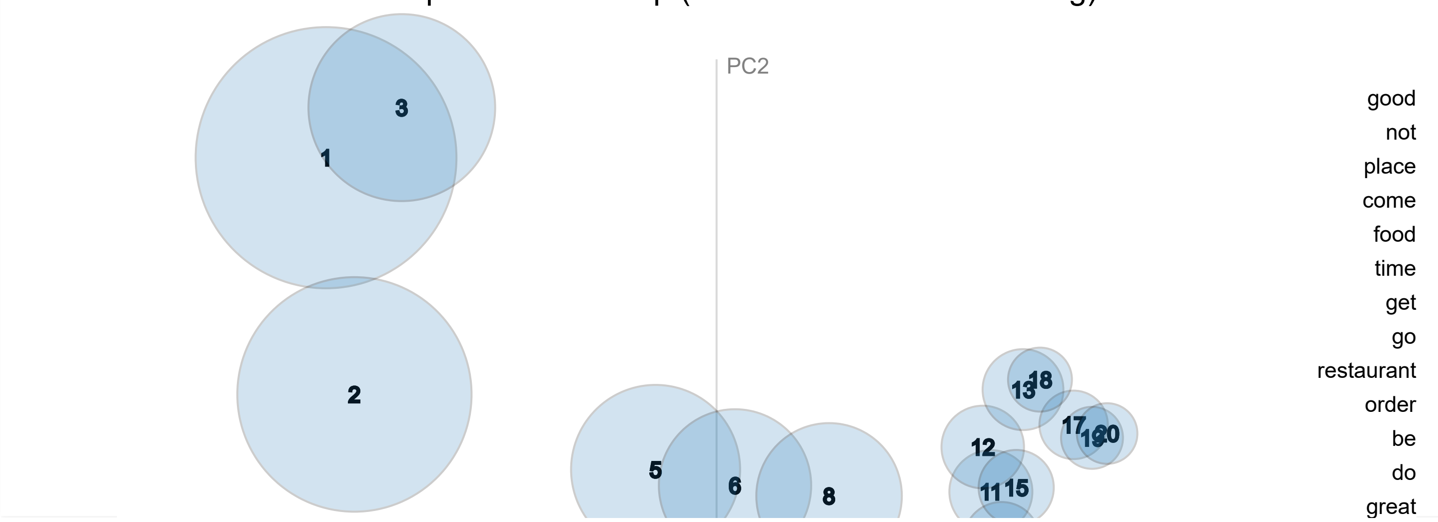
Model perplexity and topic coherence provide a convenient measure to judge how good a given topic model is. Topic Coherence is a measure used to evaluate topic models.Each such generated topic consists of words, and the topic coherence is applied to the top N words from the topic. It is defined as the average / median of the pairwise word-similarity scores of the words in the topic (e.g. PMI). A good model will generate coherent topics, and can be described by a short label, therefore this is what the topic coherence measure should capture

```
# Visualize the topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```

```
C:\Users\shabe\Anaconda3\lib\site-packages\pyLDAvis\_prepare.py:257: FutureWarning:
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  return pd.concat([default_term_info] + list(topic_dfs))
```

Selected Topic: 0 Previous Topic Next Topic Clear Topic

Intertopic Distance Map (via multidimensional scaling)



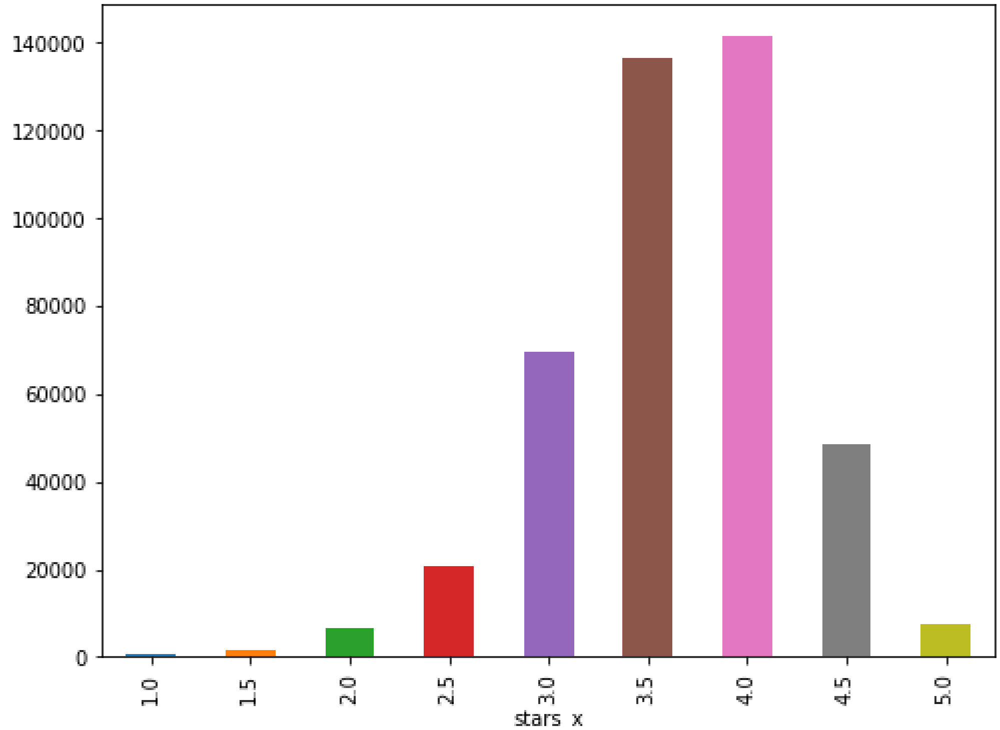
In [42]:
```
#import packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from io import StringIO
from collections import Counter
from keras.preprocessing.sequence import pad_sequences
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

from sklearn import model_selection, preprocessing, linear_model, naive_bayes,
 metrics, svm, ensemble

from sklearn.linear_model import SGDClassifier

from sklearn.datasets import make_classification

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
score, classification_report, confusion_matrix

import re
import nltk

%matplotlib inline
```

```
/home/iman_lau/anaconda3/lib/python3.5/site-packages/sklearn/ensemble/weight_
boosting.py:29: DeprecationWarning: numpy.core.umath_tests is an internal Num
Py module and should not be imported. It will be removed in a future NumPy re
lease.
  from numpy.core.umath_tests import inner1d
```

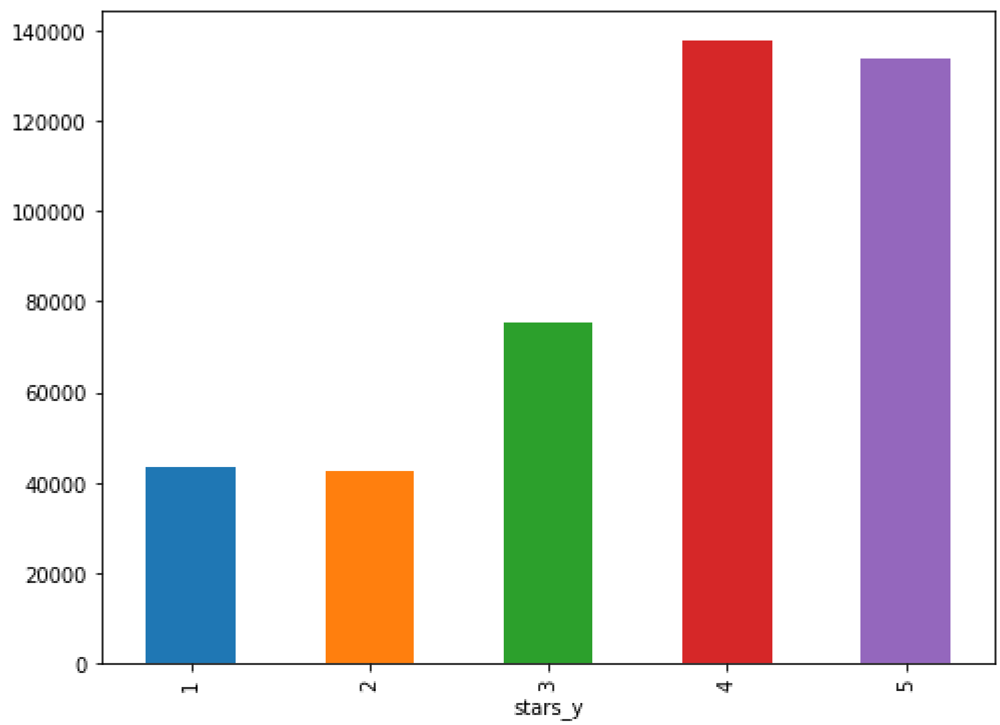
In [2]:
```
#Load in corpus

df = pd.read_csv('data/subset.csv')

# take a peek at the data
print(df.head())
```

```
          address                                         attributes  \
0  631 Bloor St W  {'BusinessParking': "{'garage': False, 'street...
1  631 Bloor St W  {'BusinessParking': "{'garage': False, 'street...
2  631 Bloor St W  {'BusinessParking': "{'garage': False, 'street...
3  631 Bloor St W  {'BusinessParking': "{'garage': False, 'street...
4  631 Bloor St W  {'BusinessParking': "{'garage': False, 'street...

              business_id      categories     city hours  is_open   latitude
\
0  9A2quhZLyWk0akUetBd8hQ  Food, Bakeries  Toronto   NaN        0  43.664378

1  9A2quhZLyWk0akUetBd8hQ  Food, Bakeries  Toronto   NaN        0  43.664378

2  9A2quhZLyWk0akUetBd8hQ  Food, Bakeries  Toronto   NaN        0  43.664378

3  9A2quhZLyWk0akUetBd8hQ  Food, Bakeries  Toronto   NaN        0  43.664378

4  9A2quhZLyWk0akUetBd8hQ  Food, Bakeries  Toronto   NaN        0  43.664378


   longitude            name           ...           stars_x state  cool  \
0 -79.414424  Bnc Cake House           ...               4.0    ON     5
1 -79.414424  Bnc Cake House           ...               4.0    ON     1
2 -79.414424  Bnc Cake House           ...               4.0    ON     0
3 -79.414424  Bnc Cake House           ...               4.0    ON     2
4 -79.414424  Bnc Cake House           ...               4.0    ON     0

         date funny               review_id stars_y  \
0  2009-07-30     5  EeMl58L8N2mWmwjLg09IcQ       5
1  2013-08-02     1  gopANOnehicgh_dAWVoxyA       5
2  2014-06-21     0  PUQYyEXwrpqjtmpG6vIU1g       3
3  2011-07-22     2  LIqVjPT-DiLsPv4U1l6Wcw       3
4  2011-08-13     0  0rU5CA1bDy15_feU7D-WMw       5

                                                text useful  \
0  Hallelujah! I FINALLY FOUND IT! The frozen yog...      5
1  I drop by BnC on a weekly basis to pick up my ...      1
2  My personally experience here wasn't the best,...      0
3  37 °C = 98.6°F\r\nKoreatown establisments disp...      2
4  My husband & I visited Toronto from the U.S. f...      0

                  user_id
0  Tj-6FX0ZnqHEZYO9iFSD4w
1  7OURjtceW40mhpRX9P2dDg
2  qQ4bfJmrfK0iWCZjl8cavQ
3  Wu0yySWcHQ5tZ_59HNiamg
4  UoCtS7YT00XyZtfDi9ZW7A

[5 rows x 23 columns]
```

In [25]:
```
#distribution of restaurant ratings
fig = plt.figure(figsize=(8,6))
df.groupby('stars_x').business_id.count().plot.bar(ylim=0)
plt.show()
```



In [24]:
```
#distribution of reviews
fig = plt.figure(figsize=(8,6))
df.groupby('stars_y').text.count().plot.bar(ylim=0)
plt.show()
```



In [4]:
```
# normalize function

wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    #doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I)
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    doc = ''.join(i for i in doc if not i.isdigit())
    return doc

normalize_corpus = np.vectorize(normalize_document)
```

In [5]:
```
# new dataframe of just reviews and star ratings

col = ['stars_y', 'text']
df = df[col]
df = df[pd.notnull(df['text'])]

df.columns = ['stars_y', 'text']

df.head()
```

Out[5]:

	stars_y	text
0	5	Hallelujah! I FINALLY FOUND IT! The frozen yog...
1	5	I drop by BnC on a weekly basis to pick up my ...
2	3	My personally experience here wasn't the best,...
3	3	37 °C = 98.6°F\r\nKoreatown establisments disp...
4	5	My husband & I visited Toronto from the U.S. f...

In [6]:
```
# normalize corpus

norm_df = normalize_corpus(df['text'])
norm_df
```

Out[6]:
```
array(["hallelujah finally found frozen yogurt launched red mango pinkberry c
raze states . ( google .) canadian incarnation goes name yogoberri discovered
inside tiny korean bakery along bloor street ' k - town . uninitiated , froze
n yogurt tart less sweet tcby kind . plain vanilla yogurt ' toppings ; fresh
fruit , nuts , cereal ... weird - looking powders never tried . small (  oz
.) $  .  +  cents per topping . medium (  oz .) including three toppings $  .
  . used eat frozen yogurt time lived korea practically weeping joy reunited
today . shameless plea : go eat lots chain multiply open branch near home . t
hanks ! ( fyi ,  stars yogurt . ' tried anything else bakery .) ** eta : dear
fro yo gods , thanks opening blushberry closer home . xoxo , susan c .**",
       "drop bnc weekly basis pick favourite buns korean bread go mid afterno
on good popular buns sold . also cakes - best green tea cake . tried bing - s
oo , dessert ice shavings , milk , red bean fruits . ' simply amazing perfect
summer . ' must try !",
       'personally experience wasnt best drink watered , tapioca bubble tea l
ittle harden . people working friendly nice , decently quiet atmosphere . goo
d place come sit chill chatting away friends .',
       ...,
       'good place get fresh quick indian food places serve authentic indian
reasonable price fast service however , would like suggest couple things  . c
hola poori combo - poori less quantity mix veg chana masala good , serve bigg
er poori  pooris  . butter chicken spicy chicken combo okay . give quantity s
auce rice .  . tried new introductory dish chicken biryani flavoured meat ric
e . would suggest increase quantity rice give . $  .  get sufficient amount r
ice fill . tandoori chicken looks yummy , going try next time . overall , goo
d place quick delicious treat . would go back .',
       'really quiet pm say , place new ( name signs previous place still ) g
etting pm lunch , \' really fair give   star review . first , get rid name /
signs anything shows \' previous shawarma place , door stopped going . though
t wrong place . second , chime bell something let know someone came place . w
alked peek kitchen / prep area , guy \' know someone . third , seems serve ch
icken veg . \' seems meat . guess \' alright , people like chicken ... oh fis
h ! seems place . services ... weird , ordered meal combo ($  .  ) picture se
e , mix salad , piece papadum , rice green pea , potato veg , meat sauce , sw
eet . ask choose chicken , spicy one butter chicken main . rice \' anything g
reen . salad given explain ran . papadum . sweet , right beside stove , serve
r ask " want sweets ?" twice reply yes yes , put box top rice . item shown me
nu , included , \' explain find something substitue . samosa added without as
king wanted , weird , hole inthe middle , try check see filling done , though
t samosa made cooked fillings ? may really caught guard . may really good nor
mal lunch hours . review benefit doubt goes toward place . back another lunch
, hopefully inthe regular lunch hours .',
       'little bit pricy based quality food ok dessert tastes really wired'],
      dtype='<U4692')
```

In [26]:
```
header = ["stars_y"]
df.to_csv('output.csv', columns = header, index = False)
```

In [7]:
```
cv = CountVectorizer(binary=False, min_df=0.0, max_df=1.0, ngram_range=(1,2))
features = cv.fit_transform(norm_df)

features.shape
```

Out[7]:
```
(706731, 9682018)
```

In [35]:
```
# binarize reviews
df['stars'] = (df['stars_y'] > 3).astype(int)

labels = df.stars
```

In [36]:
```
# build train and test datasets

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_siz
e=0.33, random_state=42)
```

In [8]:
```
def train_model(classifier, feature_vector_train, label, feature_vector_valid
):
    # fit the training dataset on the classifier
    classifier.fit(feature_vector_train, label)

    # predict the labels on validation dataset
    predictions = classifier.predict(feature_vector_valid)

    return predictions
```

In [38]:
```
# Naive Bayes
predictions = train_model(naive_bayes.MultinomialNB(), X_train, y_train, X_tes
t)

accuracy = accuracy_score(y_test, predictions)
F1 = f1_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)

print ("NB:")
print ("Accuracy: ", accuracy)
print ("F1: ", F1)
print ("Precision: ", precision)
print ("Recall: ", recall)
```

```
NB:
Accuracy:  0.8496968553566988
F1:  0.8817700428344969
Precision:  0.8400683787049176
Recall:  0.9278281731328876
```

In [39]:
```
# Logistic Regression
predictions = train_model(linear_model.LogisticRegression(), X_train, y_train,
 X_test)

accuracy = accuracy_score(y_test, predictions)
F1 = f1_score(y_test, predictions)
precision = precision_score(y_test, predictions)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-6054f38f2311> in <module>()
      5 F1 = f1_score(y_test, predictions)
      6 precision = precision_score(y_test, predictions)
----> 7 recall = recall_score((y_test, predictions))
      8
      9 print ("LR:")

TypeError: recall_score() missing 1 required positional argument: 'y_pred'
```

In [40]:
```
recall = recall_score(y_test, predictions)

print ("LG:")
print ("Accuracy: ", accuracy)
print ("F1: ", F1)
print ("Precision: ", precision)
print ("Recall: ", recall)
```

```
LG:
Accuracy:  0.8749003095762835
F1:  0.8980316501705531
Precision:  0.8845650707095744
Recall:  0.9119145976179323
```

In [ ]:
```
# Random Forest
predictions = train_model(ensemble.RandomForestClassifier(), X_train, y_train,
 X_test)

accuracy = accuracy_score(y_test, predictions)
F1 = f1_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)

print ("RF:")
print ("Accuracy: ", accuracy)
print ("F1: ", F1)
print ("Precision: ", precision)
print ("Recall: ", recall)
```

In [ ]:
```
# Stochastic Gradient Descent
predictions = train_model(SGDClassifier(), X_train, y_train, X_test)

accuracy = accuracy_score(y_test, predictions)
F1 = f1_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)

print ("SGD:")
print ("Accuracy: ", accuracy)
print ("F1: ", F1)
print ("Precision: ", precision)
print ("Recall: ", recall)
```

In [ ]:
```
```

In [8]:
```
#import packages

import pandas as pd
import numpy as np
import model_evaluation_utils as meu
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

import xgboost
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
score, classification_report, confusion_matrix

import re
import nltk

%matplotlib inline
```

In [2]:
```
# normalize function

wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    #doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I)
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    doc = ''.join(i for i in doc if not i.isdigit())
    return doc

normalize_corpus = np.vectorize(normalize_document)

#load in corpus
df = pd.read_csv('data/subset.csv')

col = ['stars_y', 'text']
df = df[col]
df = df[pd.notnull(df['text'])]

df.columns = ['stars_y', 'text']

norm_df = normalize_corpus(df['text'])
```

In [3]:
```
cv = CountVectorizer(binary=False, min_df=0.0, max_df=1.0, ngram_range=(1,2))
features = cv.fit_transform(norm_df)
labels = df.stars_y
features.shape
```

Out[3]: (706731, 9682018)

In [4]:
```
# build train and test datasets

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_siz
e=0.33, random_state=42)
```

In [7]:
```
def train_model(classifier, feature_vector_train, label, feature_vector_valid
):
    # fit the training dataset on the classifier
    classifier.fit(feature_vector_train, label)

    # predict the labels on validation dataset
    predictions = classifier.predict(feature_vector_valid)

    return metrics.accuracy_score(predictions, y_test)

predictions = train_model(xgboost.XGBClassifier(), X_train.tocsc(), y_train, X
_test.tocsc())


accuracy = accuracy_score(y_test, predictions)
F1 = f1_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score((y_test, predictions))

print ("NB:")
print ("Accuracy: ", accuracy)
print ("F1: ", F1)
print ("Precision: ", precision)
print ("Recall: ", recall)
```

```
/home/iman_lau/anaconda3/lib/python3.5/site-packages/sklearn/preprocessing/la
bel.py:151: DeprecationWarning: The truth value of an empty array is ambiguou
s. Returning False, but in future this will result in an error. Use `array.si
ze > 0` to check that an array is not empty.
  if diff:

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-1e3489979976> in <module>()
      8     return metrics.accuracy_score(predictions, y_test)
      9
---> 10 accuracy = train_model(xgboost.XGBClassifier(), X_train.tocsc(), y_tr
ain, X_test.tocsc())
     11 print("Xgb, Count Vectors: ", accuracy)

<ipython-input-7-1e3489979976> in train_model(classifier, feature_vector_trai
n, label, feature_vector_valid)
      6     predictions = classifier.predict(feature_vector_valid)
      7
----> 8     return metrics.accuracy_score(predictions, y_test)
      9
     10 accuracy = train_model(xgboost.XGBClassifier(), X_train.tocsc(), y_tr
ain, X_test.tocsc())

NameError: name 'metrics' is not defined
```

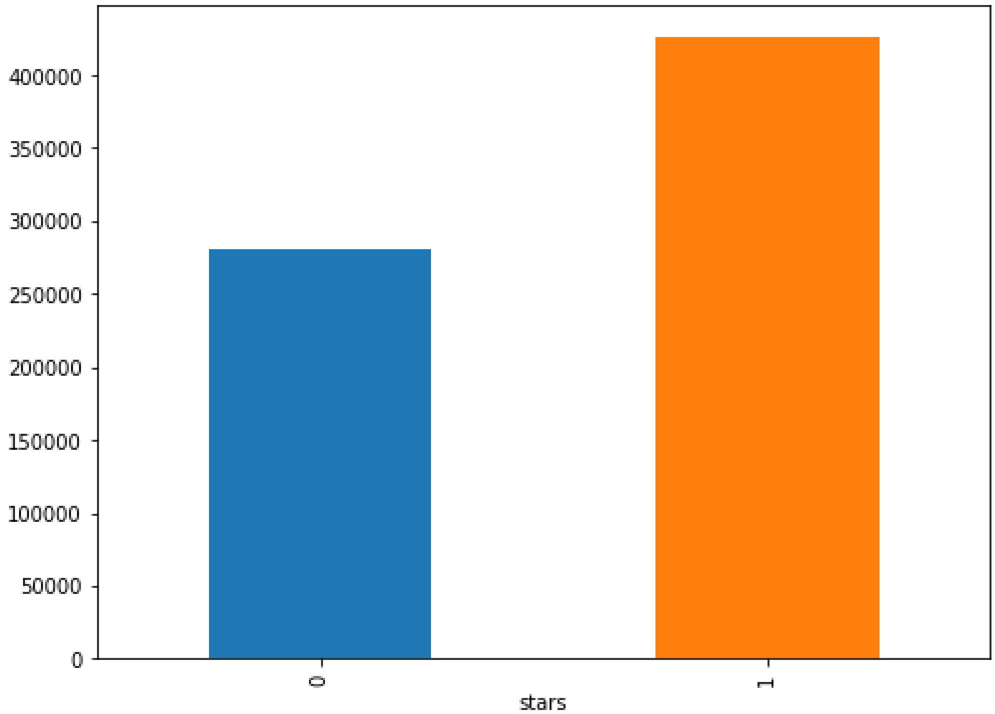
In [31]:
```
#import packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from io import StringIO
from collections import Counter
from keras.preprocessing.sequence import pad_sequences
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

from keras import models
from keras import layers
from keras import regularizers
from keras import optimizers

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
score, classification_report, confusion_matrix

import re
import nltk

%matplotlib inline
```

In [2]:
```
# normalize function

wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    #doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I)
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    doc = ''.join(i for i in doc if not i.isdigit())
    return doc

normalize_corpus = np.vectorize(normalize_document)

#load in corpus
df = pd.read_csv('data/subset.csv')

col = ['stars_y', 'text']
df = df[col]
df = df[pd.notnull(df['text'])]

df.columns = ['stars_y', 'text']

norm_df = normalize_corpus(df['text'])
```

In [16]:
```
features = df.text
```

In [34]:
```
# binarize reviews
df['stars'] = (df['stars_y'] > 3).astype(int)

labels = df.stars
```

In [37]:
```
#distribution of reviews
fig = plt.figure(figsize=(8,6))
df.groupby('stars').text.count().plot.bar(ylim=0)
plt.show()
```



In [27]:
```
from keras.preprocessing.text import Tokenizer

# load the pre-trained word-embedding vectors
embeddings_index = {}
for i, line in enumerate(open('data/wiki-news-300d-1M.vec', encoding="utf8")):
    values = line.split()
    embeddings_index[values[0]] = np.asarray(values[1:], dtype='float32')

# create a tokenizer
token = Tokenizer()
token.fit_on_texts(features)
word_index = token.word_index

# convert text to sequence of tokens and pad them to ensure equal length vecto
rs
features = pad_sequences(token.texts_to_sequences(features), maxlen=70)

# create token-embedding mapping
embedding_matrix = np.zeros((len(word_index) + 1, 300))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

In [35]:
```
# build train and test datasets

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_siz
e=0.33, random_state=42)
```

In [45]:
```
def train_model(classifier, feature_vector_train, label, feature_vector_valid,
 is_neural_net=False):
    # fit the training dataset on the classifier
    classifier.fit(feature_vector_train, label)

    # predict the labels on validation dataset
    predictions = classifier.predict(feature_vector_valid).argmax(-1)

    return predictions
```

In [36]:
```
def create_cnn():
    # Add an Input Layer
    input_layer = layers.Input((70, ))

    # Add the word embedding Layer
    embedding_layer = layers.Embedding(len(word_index) + 1, 300, weights=[embe
dding_matrix], trainable=False)(input_layer)
    embedding_layer = layers.SpatialDropout1D(0.3)(embedding_layer)

    # Add the convolutional Layer
    conv_layer = layers.Convolution1D(100, 3, activation="relu")(embedding_lay
er)

    # Add the pooling Layer
    pooling_layer = layers.GlobalMaxPool1D()(conv_layer)

    # Add the output Layers
    output_layer1 = layers.Dense(50, activation="relu")(pooling_layer)
    output_layer1 = layers.Dropout(0.25)(output_layer1)
    output_layer2 = layers.Dense(1, activation="sigmoid")(output_layer1)

    # Compile the model
    model = models.Model(inputs=input_layer, outputs=output_layer2)
    model.compile(optimizer=optimizers.Adam(), loss='binary_crossentropy')

    return model

classifier = create_cnn()
predictions = train_model(classifier, X_train, y_train, X_test)

accuracy = accuracy_score(y_test, predictions)
F1 = f1_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)

print("CNN, Word Embeddings")
print ("Accuracy: ", accuracy)
print ("F1: ", F1)
print ("Precision: ", precision)
print ("Recall: ", recall)
```

```
Epoch 1/1
473509/473509 [==============================] - 846s 2ms/step - loss: 0.3656
CNN, Word Embeddings
Accuracy:  0.39591462211969713
F1:  0.0
Precision:  0.0
Recall:  0.0

C:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1143: Undefin
edMetricWarning: F-score is ill-defined and being set to 0.0 due to no predic
ted samples.
  'precision', 'predicted', average, warn_for)
C:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1143: Undefin
edMetricWarning: Precision is ill-defined and being set to 0.0 due to no pred
icted samples.
  'precision', 'predicted', average, warn_for)
```

In [38]:
```
def create_rnn_lstm():
    # Add an Input Layer
    input_layer = layers.Input((70, ))

    # Add the word embedding Layer
    embedding_layer = layers.Embedding(len(word_index) + 1, 300, weights=[embe
dding_matrix], trainable=False)(input_layer)
    embedding_layer = layers.SpatialDropout1D(0.3)(embedding_layer)

    # Add the LSTM Layer
    lstm_layer = layers.LSTM(100)(embedding_layer)

    # Add the output Layers
    output_layer1 = layers.Dense(50, activation="relu")(lstm_layer)
    output_layer1 = layers.Dropout(0.25)(output_layer1)
    output_layer2 = layers.Dense(1, activation="sigmoid")(output_layer1)

    # Compile the model
    model = models.Model(inputs=input_layer, outputs=output_layer2)
    model.compile(optimizer=optimizers.Adam(), loss='binary_crossentropy')

    return model

classifier = create_rnn_lstm()
predictions = train_model(classifier, X_train, y_train, X_test)

accuracy = accuracy_score(y_test, predictions)
F1 = f1_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)

print("RNN-LSTM, Word Embeddings")
print ("Accuracy: ", accuracy)
print ("F1: ", F1)
print ("Precision: ", precision)
print ("Recall: ", recall)
```

```
Epoch 1/1
473509/473509 [==============================] - 1919s 4ms/step - loss: 0.379
7
RNN-LSTM, Word Embeddings
Accuracy:  0.39591462211969713
F1:  0.0
Precision:  0.0
Recall:  0.0

C:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1143: Undefin
edMetricWarning: F-score is ill-defined and being set to 0.0 due to no predic
ted samples.
  'precision', 'predicted', average, warn_for)
C:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1143: Undefin
edMetricWarning: Precision is ill-defined and being set to 0.0 due to no pred
icted samples.
  'precision', 'predicted', average, warn_for)
```

In [46]:
```
def create_rcnn():
    # Add an Input Layer
    input_layer = layers.Input((70, ))

    # Add the word embedding Layer
    embedding_layer = layers.Embedding(len(word_index) + 1, 300, weights=[embe
dding_matrix], trainable=False)(input_layer)
    embedding_layer = layers.SpatialDropout1D(0.3)(embedding_layer)

    # Add the recurrent layer
    rnn_layer = layers.Bidirectional(layers.GRU(50, return_sequences=True))(em
bedding_layer)

    # Add the convolutional Layer
    conv_layer = layers.Convolution1D(100, 3, activation="relu")(embedding_lay
er)

    # Add the pooling Layer
    pooling_layer = layers.GlobalMaxPool1D()(conv_layer)

    # Add the output Layers
    output_layer1 = layers.Dense(50, activation="relu")(pooling_layer)
    output_layer1 = layers.Dropout(0.25)(output_layer1)
    output_layer2 = layers.Dense(1, activation="sigmoid")(output_layer1)

    # Compile the model
    model = models.Model(inputs=input_layer, outputs=output_layer2)
    model.compile(optimizer=optimizers.Adam(), loss='binary_crossentropy')

    return model

classifier = create_rcnn()
rcnn_predictions = train_model(classifier, X_train, y_train, X_test, is_neural
_net=True)

accuracy = accuracy_score(y_test, rcnn_predictions)
#F1 = f1_score(y_test, predictions)
#precision = precision_score(y_test, predictions)
#recall = recall_score(y_test, predictions)

print("CNN, Word Embeddings")
print ("Accuracy: ", accuracy)
#print ("F1: ", F1)
#print ("Precision: ", precision)
#print ("Recall: ", recall)
```

```
Epoch 1/1
473509/473509 [==============================] - 823s 2ms/step - loss: 0.3675
CNN, Word Embeddings
Accuracy:  0.39591462211969713
```

In [53]:
```
rcnn_predictions[1:70]
```

Out[53]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0], dtype=int64)
```

In [48]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc

fpr_rcnn, tpr_rcnn, thresholds_rcnn = roc_curve(y_test, rcnn_predictions)
auc_rcnn = auc(fpr_rcnn, tpr_rcnn)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rcnn, tpr_rcnn, label='RCNN (area = {:.3f})'.format(auc_rcnn))
#plt.plot(fpr_rf, tpr_rf, label='RF (area = {:.3f})'.format(auc_rf))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```