Project 5 Report

Team members:

Jefferson Roylance

Justin Fairbourn

Introduction

In this project, we decided to use a dataset of sarcastic and non-sarcastic comments to see if we could classify them correctly using the naive bayes classifier. We combined two ideas - textual analysis of the comment itself, and analysis of the rest of the information about the comment contained in the dataset. In the end, our results were intriguing but we don't feel that it's quite good enough for widespread use. Ideas for further exploration include using k-nearest-neighbors or a neural network in addition to bayes to improve classification ability.

Dataset

The dataset we used was a set of over a million comments, balanced between sarcastic and non-sarcastic. The data was loaded from the .csv file, packaged into an array, and then processed into separate lists containing the data and the labels. The columns of the original dataset are as follows:

- label
- comment
- author
- subreddit
- score (# of upvotes - # of downvotes)
- ups (# of upvotes)
- downs (# of downvotes)
- date
- created_utc
- parent_comment

Analysis technique

For our analysis, we used the multinomial naive bayes classifier, since most of our data was not normally distributed and tests confirmed that using the multinomial classifier was more effective than the gaussian classifier.

For the textual analysis of the comment, each comment was converted into a long array containing boolean and number values representing the following characteristics:

- Amounts of each letter
- Length of the comment (in characters)
- Presence of punctuation (boolean)
- Average word length
- Words used - this was found out by taking the top 500 words used in all comments and then finding the counts of each of those words in the comment
- Checking for predefined patterns (we only got around to checking for the presence of '...')
- Number of uppercase letters

This information was fed into a classifier, which was cross-validated 4 times and scored using the f-score.

Justin analysis

Combined analysis

Results

Using the classifier to judge comments solely based on our analysis of the comment itself, we got the following result: Average f-score with only comment textual analysis (using all comments): 0.6209258413787513

The most fascinating part of this system is how it could be potentially used. If accuracy was extremely high, for example, reddit could go through and tag all sarcastic comments to stop people from misunderstanding. This could also be used in messaging apps if the classifier thinks that a message is sarcastic.

%%latex \newpage

Project X Code

Comments relating to code snippet 1

In [2]:
```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
import random
import re
```

In [3]:
```
comments = pd.read_csv("train-balanced-sarcasm.csv")
```

In [4]:
```
display(comments.head())
```

	label	comment	author	subreddit	score	ups	downs	date	created_utc	pa
0	0	NC and NH.	Trumpbart	politics	2	-1	-1	2016-10	2016-10-16 23:55:23	ar
1	0	You do know west teams play against west teams...	Shbshb906	nba	-4	-1	-1	2016-11	2016-11-01 00:24:10	T we
2	0	They were underdogs earlier today, but since G...	Creepeth	nfl	3	3	0	2016-09	2016-09-22 21:45:37	Th
3	0	This meme isn't funny none of the "new york ni...	icebrotha	BlackPeopleTwitter	-8	-1	-1	2016-10	2016-10-18 21:03:47	de
4	0	I could use one of those tools.	cush2push	MaddenUltimateTeam	6	-1	-1	2016-12	2016-12-30 17:00:13	Ye sa

In [5]:
```
comment_list = [(
    row['label'],
    str(row['comment']),
) for index, row in comments.iterrows()]
random.shuffle(comment_list)
```

In [159]:
```
comment_list_truncated = comment_list[:5000]
```

In [160]:
```
bigString = ' '.join([comment[1] for comment in comment_list])
wordList = re.sub("[^\w]", " ",  bigString.lower()).split()

bigDict = {}
for word in wordList:
    if word in bigDict:
        bigDict[word] += 1
    else:
        bigDict[word] = 1
display(len(bigDict))
```

```
167472
```

In [161]:
```
topWords = sorted(bigDict, key=bigDict.__getitem__, reverse=True)[:500]
display(topWords[:10])
```

```
['the', 'a', 'to', 'i', 'you', 'it', 'and', 'that', 'is', 'of']
```

In [162]:
```
from sklearn import preprocessing

alphabet = 'abcdefghijklmnopqrstuvwxyz1234567890'
uppercaseAlphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
punctuation = '*\"\(\)~:/&\'.-#[];_^$\{\}=!+?@%`,|\x08'
patterns = ['...']
alphabet += punctuation
le = preprocessing.LabelEncoder()
le.fit([l for l in alphabet])

def comment_features_word(comment):
    features = []

    # Letter counts
    for letter in alphabet:
        features.append(comment.lower().count(letter))

    # Length
    features.append(len(comment))

    # Presence of punctuation
    punctPresence = False
    for p in punctuation:
        punctPresence = punctPresence or p in comment
    features.append(punctPresence)

    # Average word length
    commentWords = re.sub("[^\w]", " ",  comment).split()
    a = sum([len(word) for word in commentWords]) / len(commentWords) if
len(commentWords) > 0 else 0
    features.append(a)

    # Words used
    a = []
    for word in topWords:
        a.append(comment.lower().count(word))
    features.extend(a)

    # Checking for predefined patterns
    a = []
    for pattern in patterns:
        a.append(comment.count(pattern))
    features.extend(a)

    # Checking for number of uppercase letters
    a = 0
    for letter in uppercaseAlphabet:
        a += comment.count(letter)
    features.append(a)

    return features
```

In [163]:
```
X = [comment_features_word(comment) for (_, comment) in comment_list_tru
ncated]
y = [label for (label, _) in comment_list_truncated]
```

In [171]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.model_selection import cross_validate
```

In [172]:
```
clf = MultinomialNB()

cv_results = cross_validate(clf, X, y, cv=4, scoring='f1')

print("Average f-score: ", sum(cv_results['test_score']) / len(cv_result
s['test_score']))
```

```
Average f-score:  0.6425650674933281
```

In [ ]:

In [ ]: