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

In addition to the comments themselves, we wanted to see if we could build a classifier that can detect sarcasm using the subreddit and score data. For memory reasons, we only considered the top 15 subreddits by summing up the frequencies of each subreddit. This gave us a 1,000,000 x 22 DataFrame with "dummy" variables that allowed us to represent the categorical subreddit data numerically.

Then, we created a Naive-Bayes classifier and fed it the following attributes:

- The total amount of "upvotes" the comment received.
- The total amount of "downvotes" the comment received.
- 15 dummy boolean variables representing the comment's subreddit.

Not satisfied with the f-scores, we also ran another analysis and instead fed the classifier *only* the subreddit data. Surprisingly, this gave slightly better performance.

In both of these analyses, cross-validation was performed 20 times.

Combined analysis

Wanting to get the best results possible, we decided to combine the feature sets used in the previous two analyses to create one giant featureset with both the word features and the subreddit features.

Unfortunately, this led to some intense memory issues, so we needed to reduce the amount of comments we were analyzing to 10000 in order to successfully fit our 500 features into the Naive-Bayes classifier. Once this was done succesfully, we ran cross-validation 5 times.

Results

Average f-score with only comment textual analysis (using all comments): 0.6209258413787513

Average f-score with subreddit and score data (using comments in the top 15 subreddits): 0.4480299005407648

Average f-score with just subreddit data (no scores): 0.5123293945443769

Average f-score with subreddit data and comment textual analysis: pending due to MemoryError.

Results Explained

The f-scores above indicate that the filter was most effective when only considering the comment's text. By finding the most common words used in the top 10,000 comments we were able to build a sarcasm filter that could successfully detect sarcasm more than 50% of the time.

By comparison, a filter that just relied on the subreddit and score data detected sarcasm less than 50% of the time, while a filter on subreddit data was barely above 50%.

Overall: The results hint that *where* a sarcastic Reddit comment is posted is not as important as *what* the comment actually contains.

Improvements

To improve the performance of this filter, the first thing we would need to do is optimize the way it handles the memory of the analysis. This will allow us to analyze more comments and subreddits more quickly, and allow us to handle more meaningful feature sets to provide better f-scores.

%%latex \newpage

Project X Code

Comments relating to code snippet 1

In [1]:
```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
import random
import re
```

In [9]:
```
comments = pd.read_csv("data/train-balanced-sarcasm.csv")
```

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

In [10]:
```
comment_list = [(
    row['label'],
    str(row['comment']),
) for index, row in comments[:10000].iterrows()]
# random.shuffle(comment_list)
```

In [11]:
```
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))
```

```
13679
```

In [12]:
```
topWords = sorted(bigDict, key=bigDict.__getitem__, reverse=True)[:500]
display(topWords[:10])
```

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

In [13]:
```
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(c
ommentWords) > 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 [ ]:
```
X = [comment_features_word(comment) for (_, comment) in comment_list]
y = [label for (label, _) in comment_list]
```

In [3]:
```
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 [ ]:
```
clf = MultinomialNB()

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

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

Using the Subreddit Data

In [4]:
```
subreddit_counts = comments['subreddit'].value_counts()
comments_counts = comments.join(subreddit_counts, on='subreddit', rsuffix='_co
unt')
comments_counts = comments_counts[comments_counts.subreddit_count > 8000]
subreddit_data = pd.get_dummies(comments_counts['subreddit'], prefix='r',spars
e=True)
display(comments_counts['label'].value_counts())
```

```
1    183069
0    169832
Name: label, dtype: int64
```

In [5]:
```
# Drop everything except for the subreddit dummy values and the up/down scores
and the labels
comments_test = comments_counts.drop(['comment', 'subreddit', 'author', 'date'
, 'created_utc', 'parent_comment','score','subreddit_count'], axis=1)
dummy_comments = pd.concat([comments_test, subreddit_data], axis=1)

# Take absolute value of each score since NBC's don't like negative numbers
dummy_comments['ups'] = abs(dummy_comments['ups'])
dummy_comments['downs'] = abs(dummy_comments['downs'])

X = dummy_comments.drop('label', axis=1).values
y = dummy_comments['label'].values

f_scores_0 = []
f_scores_1 = []

for _ in range(20):
    X_train, X_test, y_train, y_test = train_test_split(X, y)

    clf = MultinomialNB()
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    p,r,f,s = precision_recall_fscore_support(y_test, y_pred)

    f_scores_0.append(f[0])
    f_scores_1.append(f[1])

display(sum(f_scores_0) / float(len(f_scores_0)))
display(sum(f_scores_1) / float(len(f_scores_1)))
```

```
0.4480299005407648
```

```
0.6316247059037982
```

In [6]:
```
# Drop everything except for the subreddit dummy values and the labels
comments_test_no_scores = comments_counts.drop(['comment', 'subreddit', 'autho
r', 'date', 'created_utc', 'parent_comment','score','subreddit_count','ups','d
owns'], axis=1)
dummy_comments_no_scores = pd.concat([comments_test_no_scores, subreddit_data
], axis=1)

X = dummy_comments_no_scores.drop('label', axis=1).values
y = dummy_comments_no_scores['label'].values

f_scores_0_no_scores = []
f_scores_1_no_scores = []

for _ in range(20):
    X_train, X_test, y_train, y_test = train_test_split(X, y)

    clf = MultinomialNB()
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    p,r,f,s = precision_recall_fscore_support(y_test, y_pred)

    f_scores_0_no_scores.append(f[0])
    f_scores_1_no_scores.append(f[1])

display(sum(f_scores_0_no_scores) / float(len(f_scores_0_no_scores)))
display(sum(f_scores_1_no_scores) / float(len(f_scores_1_no_scores)))
```

```
0.5123293945443769

0.6119059818643958
```

The f-scores for the data *without* the scores was slightly better, we'll combine those features with the comment word features to do our final analysis.

In [16]:

```
import sys
# Drop everything except for the subreddit dummy values and the labels
comments_test_full = comments_counts.head(1000)
comments_test_full = comments_test_full.drop(['subreddit', 'author', 'date',
'created_utc', 'parent_comment','score','subreddit_count','ups','downs'], axis
=1)
comments_test_full = pd.concat([comments_test_full, subreddit_data], axis=1)

X = comments_test_full.drop(['label','comment'], axis=1).values.tolist()
y = comments_test_full['label'].values.tolist()
comments = comments_test_full['comment'].values.tolist()

for i in range(len(X)):
    next_comment = str(comments[i])
    X[i].extend(comment_features_word(next_comment))


comments_test_full = []

print('finished extending')
f_scores_0_no_scores = []
f_scores_1_no_scores = []

for i in range(5):
    print(i)
    X_train, X_test, y_train, y_test = train_test_split(X, y)

    clf = MultinomialNB()
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    p,r,f,s = precision_recall_fscore_support(y_test, y_pred)

    f_scores_0_no_scores.append(f[0])
    f_scores_1_no_scores.append(f[1])

display(sum(f_scores_0_no_scores) / float(len(f_scores_0_no_scores)))
display(sum(f_scores_1_no_scores) / float(len(f_scores_1_no_scores)))
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-16-029a295abfeb> in <module>
      3 comments_test_full = comments_counts.head(1000)
      4 comments_test_full = comments_test_full.drop(['subreddit', 'author',
'date', 'created_utc', 'parent_comment','score','subreddit_count','ups','down
s'], axis=1)
----> 5 comments_test_full = pd.concat([comments_test_full, subreddit_data],
 axis=1)
      6
      7 X = comments_test_full.drop(['label','comment'], axis=1).values.tolis
t()

c:\users\justi\appdata\local\programs\python\python36-32\lib\site-packages\pa
ndas\core\reshape\concat.py in concat(objs, axis, join, join_axes, ignore_ind
ex, keys, levels, names, verify_integrity, sort, copy)
    226                       keys=keys, levels=levels, names=names,
    227                       verify_integrity=verify_integrity,
--> 228                       copy=copy, sort=sort)
    229     return op.get_result()
    230

c:\users\justi\appdata\local\programs\python\python36-32\lib\site-packages\pa
ndas\core\reshape\concat.py in __init__(self, objs, axis, join, join_axes, ke
ys, levels, names, ignore_index, verify_integrity, copy, sort)
    379         self.copy = copy
    380
--> 381         self.new_axes = self._get_new_axes()
    382
    383     def get_result(self):

c:\users\justi\appdata\local\programs\python\python36-32\lib\site-packages\pa
ndas\core\reshape\concat.py in _get_new_axes(self)
    446                 if i == self.axis:
    447                     continue
--> 448                 new_axes[i] = self._get_comb_axis(i)
    449         else:
    450             if len(self.join_axes) != ndim - 1:

c:\users\justi\appdata\local\programs\python\python36-32\lib\site-packages\pa
ndas\core\reshape\concat.py in _get_comb_axis(self, i)
    467             return _get_objs_combined_axis(self.objs, axis=data_axis,
    468                                            intersect=self.intersect,
--> 469                                            sort=self.sort)
    470         except IndexError:
    471             types = [type(x).__name__ for x in self.objs]

c:\users\justi\appdata\local\programs\python\python36-32\lib\site-packages\pa
ndas\core\indexes\api.py in _get_objs_combined_axis(objs, intersect, axis, so
rt)
     68                  if hasattr(obj, '_get_axis')]
     69     if obs_idxes:
---> 70         return _get_combined_index(obs_idxes, intersect=intersect, so
rt=sort)
     71
     72
```

```
c:\users\justi\appdata\local\programs\python\python36-32\lib\site-packages\pa
ndas\core\indexes\api.py in _get_combined_index(indexes, intersect, sort)
    115             index = index.intersection(other)
    116     else:
--> 117         index = _union_indexes(indexes, sort=sort)
    118         index = ensure_index(index)
    119

c:\users\justi\appdata\local\programs\python\python36-32\lib\site-packages\pa
ndas\core\indexes\api.py in _union_indexes(indexes, sort)
    181         else:
    182             for other in indexes[1:]:
--> 183                 result = result.union(other)
    184             return result
    185     elif kind == 'array':

c:\users\justi\appdata\local\programs\python\python36-32\lib\site-packages\pa
ndas\core\indexes\base.py in union(self, other, sort)
   2320         if self.is_monotonic and other.is_monotonic:
   2321             try:
-> 2322                 result = self._outer_indexer(lvals, rvals)[0]
   2323             except TypeError:
   2324                 # incomparable objects

c:\users\justi\appdata\local\programs\python\python36-32\lib\site-packages\pa
ndas\core\indexes\base.py in _outer_indexer(self, left, right)
    223
    224     def _outer_indexer(self, left, right):
--> 225         return libjoin.outer_join_indexer(left, right)
    226
    227     _typ = 'index'

pandas\_libs\join.pyx in pandas._libs.join.outer_join_indexer()

MemoryError:
```

In [ ]:

```
X = None
y = None
comments_test_full = None
X_train = None
y_train = None
X_test = None
y_test = None
clf = None
```