Tweet-Party Classifier

Neel Mehta, Ajay Nathan, Saranya Vijayakumar, Samuel Lam.

About

Uses the Naive Bayes Classifier to predict the political party of the author of a Tweet. Uses data from Quorum Analytics.

Data Procurement

Quorum has over 1 million labeled Tweets in their database. We will use these Tweets to train our classifier. We will be writing a small API to easily interact with the Tweet data. That API is currently under construction.

Interface

Enums

- Party
  - 0: Democratic
  - 1: Republican

We story party using an Enum instead of a string; it is a useful abstraction that guards against accidentally mistyping identifier names and makes type annotations more descriptive.

Objects

- Tweet
  - `party` : Party
  - `name` : str
  - `text` : str
  - `freqlist` : bool list
  - `__init__` (party : Party, name : str, text : str)
    - Generates the frequency list by running extract(text)
- Classifier (implementation of Naive Bayes Classifier)
  - `train` (Tweet list) : None
  - `classify` (str) : Party
    - Guesses the party of the author of the given text (could be Tweet or other writing)
  - `accuracy` (Tweet list) : float

        - Runs classify on the text of every Tweet and returns the percent of correct party classifications.

We could simply have represented a Tweet as a tuple, but to allow for an initializer that auto-generates the frequency list out of the Tweet text (and to allow for better abstraction), we use a Tweet object. The Classifier object lets us encapsulate the state of the Bayes Classifier.

Functions

- `get_tweets` () : Tweet list
    - Reads in raw Tweet data and constructs a Tweet object out of each. Data acquisition under construction by Ajay.
- `partition_tweets` (Tweet list) : (Tweet list, Tweet list)
    - Randomly splits the given Tweet list into a training and a testing list.
- `extract` (str) : bool list
    - Uses NLTK to remove English stopwords, remove inflections, and return a frequency list for the top remaining words (if word *i* is in the Tweet, position *i* in the list will be true.)

These are largely utility functions.

Process

We simple get the Tweets, train the classifier using a subset of them, and test the classifier using the remainder.

```
tweets = get_tweets()
(train, test) = partition(tweets)
classifier = new Classifier()
classifier.train(train)
print(classifier.accuracy(test))
```

Technical Specifications

This project uses Python 2 and Django. Check the Readme for instructions on running on your own machine.

We use the Natural Language Toolkit to simplify processing the raw Tweet text.

Naive Bayes

The classifier is based on the Naive Bayes algorithm: the probabilty of a tweet is determined first by applying Bayes Rule to express P(label/feature) in terms of P(label) and P(features|label):

```
|                       P(label) * P(features|label)
|  P(label|features) = ------------------------------
|                              P(features)
```

The 'naive' assumption is then made that all features are independent, given the label:

```
|                       P(label) * P(f1|label) * ... * P(fn|label)
|  P(label|features) = ------------------------------------------
|                                       P(features)
```

Rather than computing P(features) explicitly, the algorithm calculates the denominator for each label and then normalizes them so that they sum to one:

```
|                       P(label) * P(f1|label) * ... * P(fn|label)
|  P(label|features) = ------------------------------------------
|                        SUM[l]( P(l) * P(f1|l) * ... * P(fn|l) )
```

In order to find the most common words in the population of training data that are not stop words, this project utilizes our custom classifier with the class nltk.probability.FreqDist to identify top words.

```
from nltk.probability import FreqDist, DictionaryProbDist
```

- Collect_all_words method to return an array of all words from the training tweets
- The array is then passed to Identify_top_words method to identify the most frequent words
- nltk.FreqDist class : hash, sort the keys by their corresponding values, or counts
- We will obtain the top ### words with [:###]

```
def collect_all_words(self, items):
    all_words = []
    for item in items:
    for w in item.all_words:
        words.append(w)
        return all_words

def identify_top_words(self, all_words):
    freq_dist = nltk.FreqDist(w.lower() for w in all_words)
    return freq_dist.keys()[:###]
```

- Get the features for each tweet
- Run array of all_words to reduce to a smaller set object to eliminate duplicate words
- Iterate through top_words and compare to this set for presence or absence, a hash of ### Booleans is returned

```
def features(self, top_words):
    word_set = set(self.all_words)
    features = {}
    for w in top_words:
    features["w_%s" % w] = (w in word_set)
    return features
```

- Collect the training set of tweets and their individual features and pass them to algorithm
- Once the classifier is trained, we will iterate through the set of tweets that remain to be classified. The classifier will guess the category for each item.

```
for item in tweets_to_classify:
    features = item.features(top_words)
    category = classifier.classify(features)
```

For accuracy evaluation we can use the custom `classifier.accuracy` function.

```
print(classifier.accuracy(test_set))
```

Version Control

We are using GitHub for our project; it lives at [hathix/tweet-party-classifier](). So far we have scaffolded out a Django app that can be run by following instructions in the [README]().

Advanced stuff

Based on rate of successful categorization, we can further assess whether we will need additional classification methods or variations of current feature evalutations.

Timeline

Writeup due Friday, April 17

Starting Friday, April 17:

- Create front end - Django app
- Write extract function
- Figure out how to get tweets from Quorum (data procurement)
- Write classifier - train and test

Starting Friday, April 24:

- Finish writing Naive Bayes classifier
- Interface
- Helper functions (like accuracy calculation)
- Finish front end
- "Reach" goals
- testing

Due: May 1