Sentiment Analysis, a hands-on tutorial

Peter Conway

About me

Physics graduate from Swansea University.

Back-end Python developer.

Software developer for Hertzian Ltd.

Contact: peter@hertzian.co.uk

About Hertzian

Customer feedback analysis company based in Truro.

Born from Falmouth University's incubation program.

Clients in health, film & tv, e-commerce, and games.

Majority of our business done with clients in the USA.

About today

An introduction to Sentiment Analysis and Sentiment Classification.

Python workshop.

Code improvements & alternative methods to increase accuracy.

Classification in ML

What is classification?

How we determine which object(s) belong to which group(s).

Image classification

Probably the most broadly known form of classification.

- Face detection.
- Self-driving cars.

Tank myth/story & moral.

Text classification

Part of the NLP (Natural Language Processing) sphere discussed by Tariq in first talk.

Features of text you may want to classify:

- Spam filters (spam/not spam).
- 'Topics' of conversation, e.g. in automated phone systems.

The topic of one of our tasks today: writing a sentiment classifier.

Sentiment classifiers

Goal: To categorize reviews as either positive or negative.

Manifestation of tank story in my work.

Python's NLTK library offers a Naive Bayes classifier we can use to learn about NLP.

Sentiment calculation

Non-ML approach to sentiment classification

Task 1

VADER Sentiment Lexicon

A lexicon is a dictionary or vocabulary of words.

The VADER lexicon offers us sentiment scores associated to 7502 words!

Open source, I've downloaded + .pickle'd it for the talk's repository.

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()
lexicon = sia.lexicon
```

Scores provided by NLTK are in [-4,4] scale, I've provided mine in [-1,1].

(Aside) .pickle in Python

Used to 'serialize' data and preserve object type.

Extremely common in ML for storing models.

```
import pickle
```

```
#Saving to .pickle:
with open('fileName.pickle', 'wb') as f:
    pickle.dump(obj, f)
```

```
#Loading from .pickle:
with open('fileName.pickle', 'rb') as f:
    obj = pickle.load(f)
```

Algorithmic Logic

Objective: calculate a single score to represent each review's sentiment.

We need to convert our review from a string to word tokens, then check whether our lexicon has any scores associated to any of our words, and perform some calculation to combine individual word scores.

The maths is up to you! I'm interested to see what approaches are used to combine scores.

Tokenizing reviews

Word tokenization:

```
from nltk.tokenize import word_tokenize
tokens = word_tokenize('this is an example')
```



```
['this', 'is', 'an', 'example']
```

Sentence tokenization:

```
import nltk
review = "Wow! This game is great. I love the characters."
print(sent_tokenize(review))
```

```
['Wow!', 'This game is great.', 'I love the characters.']
```

Assorted useful Python

Iteration (loops):

```
for review in reviews:
    print(review)
```

Using dictionaries:

```
words = {}
words['review'] = 2
words['game'] = 3
words['Python'] = 1
```

→ `{'review': 2, 'game': 3, 'Python': 1}`

Dictionary .keys():

`words.keys()` → `['review', 'game', 'Python']`

Reading dictionaries:

`print(words['Python'])` → `1`

Naive Bayes Classifier

Task 2

Naive Bayes classifier

Built on the logic of Bayes theorem. [See link on last slide]

Uses Bag of Words (BOW) approach, as opposed to superior Continuous Bag of Words (CBOW).

Supervised machine learning method (requires labelled data).

Loading and organizing data

We can access data with:

```
with open('filename.txt', 'r', encoding='utf-8') as f:
    reviews = f.readlines()
```

Shuffling our data for run-to-run variance (optional):

```
import random
random.shuffle(reviews)
```

Creating `training` and `evaluation` subsets:

```
some_reviews = reviews[:100]
```

Training our model

NLTK expects us to structure our data in a particular way...

```
({'not': True, 'long': True, 'enough': True}, 'pos')
```

RED: A dictionary whose keys are word tokens, with values `True`.

GREEN: Our classification, known by opening pos/neg file.

Finally, we can train our model:

```
#Train our classifier:
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

Evaluating our model

Two suggested methods of evaluation: Informative features + accuracy tests.

Informative features:

```
classifier.show_most_informative_features(15)
```

```
    awesome = True              pos : neg    =     25.8 : 1.0
     acting = True              pos : neg    =     12.3 : 1.0
 spaceland = True               pos : neg    =     11.7 : 1.0
       fans = True              pos : neg    =     11.7 : 1.0
       8/10 = True              pos : neg    =     11.0 : 1.0
      waste = True              neg : pos    =     11.0 : 1.0
    deserve = True              pos : neg    =     10.3 : 1.0
       yeah = True              pos : neg    =     10.2 : 1.0
      ethan = True              pos : neg    =      9.7 : 1.0
     unique = True              pos : neg    =      9.7 : 1.0
      laggy = True              neg : pos    =      9.0 : 1.0
     wasted = True              neg : pos    =      9.0 : 1.0
    visuals = True              pos : neg    =      9.0 : 1.0
      terms = True              pos : neg    =      9.0 : 1.0
   missions = True              pos : neg    =      8.7 : 1.0
```

Accuracy tests:

```
nltk.classify.accuracy(classifier, eval_set)
```

```
Naive Bayes Algo accuracy: 0.75
```

Calculating Accuracy

The *correct* way to calculate accuracy

When collecting evaluation data, it's important to compare the results one at a time. If you're expecting 50 positive and 50 negative reviews, your classifier can be exactly wrong (classify all 50 positive reviews as negative, and all 50 negative reviews as positive) and still claim to be 100% accurate.

Precision and recall

Helps evaluate accuracy of imbalanced classification tasks, e.g. detection of cancer.

Recall: Ability to identify relevant instances.

Precision: Proportion of identified instances that actually were relevant.

F1 score combines results:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Datasets

Movie reviews: Open-source, provided by NLTK. Contains 1000 positive and 1000 negative reviews.

Steam (game) reviews: Using Steam recommendations given by reviewers. I crawled Infinite Warfare due to its useful split between pos/neg reviews.

Workshop tasks

Task 1: Sentiment score calculation

- (Load reviews)
- Load lexicon.
- Tokenize text.
- Look up words in lexicon.
- Combine scores with maths.
- Identify issues/improvements.
    - There's at least one glaring problem.

Task 2: Naive Bayes classifier

- (Load reviews)
- Create training and evaluation sets.
- Reformat reviews to ({'word': True}, 'pos')
- Train model.
    - .pickle model to avoid recalculation?
- Evaluate model.
    - Informative features.
    - .accuracy() function.
- Try out the model!
- Identify issues/improvements.

Extra links/resources

Explanation of Bayes Theorem.