COMP348 — Document Processing and the Semantic Web

Week 04 Lecture 1: Developing Text Classification Systems

Diego Mollá

COMP348 2018H1

Abstract

This lecture will focus on several text processing tasks than can be performed as text classification. This is a very practical lecture that shows how these tasks can be done in NLTK and sklearn, and how we can evaluate them.

Update March 16, 2018

Contents

Reading

- NLTK Book Chapter 6 "Learning to Classify Text"

1 Some Text Classification Tasks

1.1 Gender Classification (again)

Gender Classification - Take 2

- In a past lecture we have used this code to encode the last two characters of a name:

```
def gender_features(word):
    "Return the ASCII value of the last two characters"
    return [ord(word[-2]), ord(word[-1])]
```

- This code is not entirely correct since it is representing characters as numbers.
- In general, non-numerical information is best represented using one-hot encoding.
- sklearn provided the following functions to produce one-hot-encoding vectors:

- preprocessing.OneHotEncoding: from integers to one-hot vectors.
- preprocessing.LabelBinarizer: from labels to one-hot vectors.

One-hot Encoding

- Suppose you want to encode five labels: 'a', 'b' ,'c', 'd', 'e'.
- Each label represents one element in the one-hot vector.
- Thus:
  - 'a' is represented as (1, 0, 0, 0, 0).
  - 'b' is represented as (0, 1, 0 ,0 ,0).
  - and so on.
- This is also called binarization.

One-hot Encoding for Gender Classification

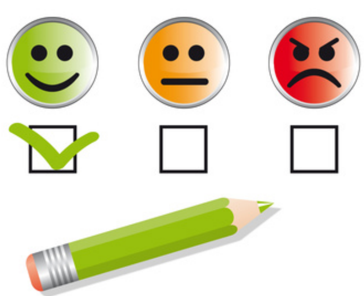
```
def one_hot_character(c):
  alphabet = 'abcdefghijklmnopqrstuvwxyz'
  result = [0]*len(alphabet)
  result[alphabet.index(c.lower())] = 1
  return result

def gender_features(word):
    last = one_hot_character(word[-1])
    secondlast = one_hot_character(word[-2])
    return secondlast + last
```

1.2 Movie Reviews

Movie Reviews

- Movie reviews can be positive or negative.
- This can be easily modelled as a text classification task.
- This is also an example of sentiment analysis.



One-Hot Encoding of Words

- A common approach to represent bag of words is the one-hot encoding.
- Basically, each element in the word vector is either 0 or 1.
- The result is a sparse matrix.
- Tf.idf can be seen as an extension of one-hot encoding.

Remember this slide on Vector Space Model?

Each document vector is like an OR operation of the one-hot encoding vectors of each word in the document.

Template:

{*computer,software,information,document,retrieval,language,library,filtering*}

Initial documents
D1:{computer,software,information,language}
D2:{computer,document,retrieval,library}
D3:{computer,information,filtering,retrieval}

Document vectors
D1: (1,1,1,0,0,1,0,0)
D2: (1,0,0,1,1,0,1,0)
D3: (1,0,1,0,1,0,0,1)

Document matrix

$$D = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Finding the 2000 most frequent non-stop words

```python
import nltk
import collections
from nltk.corpus import stopwords
stop = stopwords.words('english')
c = collections.Counter([w.lower()
                         for (words, category) in train
                         for w in words
                           if w.lower() not in stop])
top2000words = [w for (w, count) in c.most_common(2000)]
```

Remember
When you compute statistics for machine learning, only use the training set.

Implementing the document features in NLTK

```
def document_features(words):
    "Return the document features for an NLTK classifier"
    words_lower = [w.lower() for w in words]
    result = dict()
    for w in top2000words:
        result['has(%s)' % w] = (w in words_lower)
    return result
```

Training and Evaluating NLTK's Naive Bayes

```
train_features = [(document_features(x),y)
                        for (x,y) in train]
devtest_features = [(document_features(x),y)
                        for (x,y) in devtest]
classifier = nltk.NaiveBayesClassifier.train(train_features)
nltk.classify.accuracy(classifier,devtest_features)
```

Implementing the document features in Scikit-Learn

```
def vector_features(words):
   "Return a vector of features for sklearn"
   words_lower = [w.lower() for w in words]
   result = []
   for w in top2000words:
     if w in words_lower:
       result.append(1)
     else:
       result.append(0)
   return result
```

Training and Evaluating Scikit-Learn's Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

train_vectors = [vector_features(x) for (x, y) in train]
train_labels = [y for (x, y) in train]
devtest_vectors = [vector_features(x) for (x, y) in devtest]
devtest_labels = [y for (x, y) in devtest]

sklearn_classifier = MultinomialNB()
sklearn_classifier.fit(train_vectors, train_labels)
MultinomialNB(alpha=1.0, class prior=None, fit prior=True)

predictions = sklearn_classifier.predict(devtest_vectors)
accuracy_score(devtest_labels, predictions)
```

Using tf-idf as document features

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
train_features = tfidf.fit_transform([x for x, y in train_raw])

sklearn_classifier2 = MultinomialNB()
sklearn_classifier2.fit(train_features,
                        [y for x, y in train_raw])

devtest_features = tfidf.transform([x for x, y in devtest_raw])
predictions = sklearn_classifier2.predict(devtest_features)
accuracy_score([y for x, y in devtest_raw], predictions)
```

1.3 Sentence Segmentation

Sentence Segmentation

- Sentence segmentation can be reduced to the process of disambiguating potential punctuation characters.
- So we can apply a classifier to the characters.
- For sentence segmentation we need to look at the context.

Possible Context-based Features

The following features could be useful to determine if a punctuation character is an end of sentence:

1. Is the next word capitalised?
2. What is the previous word?
3. What is the current punctuation character?
4. Is the previous word a character?

Feature Extraction with Context

```
def segmenter_features(tokens, i):
    """Return the features of token[i]"""
    return {'next-word-capitalized':
                tokens[i+1][0].isupper(),
            'prev-word': tokens[i-1].lower(),
            'punct': tokens[i],
            'prev-word-is-one-char':
                len(tokens[i-1]) == 1}
```

ML for Sentence Segmentation

Training and Testing Data

- We only need to look at candidate characters.
- For training and testing we need to keep track of the positions of all sentence endings.

```
candidates = '.?!'
train_features = [(segmenter_features(train_tokens, i),
                    (i in train_boundaries))
                   for i in range(1, len(train_tokens)-1)
                   if train_tokens[i] in candidates]
test_features = [(segmenter_features(test_tokens, i),
                   (i in test_boundaries))
                   for i in range(1, len(test_tokens)-1)
                   if test_tokens[i] in candidates]
```

Running the Classifier

```
segmenter=nltk.NaiveBayesClassifier.train(train_features)

test = ["This", "is", "a", "sentence", ".", "This",
        "is", "another", "one"]
for i in range(0,len(test)):
    if test[i] in candidates and \
       classifier.classify(features(test,i)):
       print "Sentence boundary at position", i
```

Evaluation of the Classifier

```
>>> nltk.classify.accuracy(segmenter,
                           test_features)
1.0
```

- Accuracy on the test set is impressive.
- However, sentence segmentation is almost always easy.
- Let's compare against a very simple baseline.

A Majority Baseline

- A majority baseline is a classifier that always returns the majority class.
- In our case, in most cases the punctuation character is an end of sentence.

```
>>> from collections import Counter
>>> train_counter = Counter([f[1] for f in train_features])
>>> train_counter
Counter({False: 62, True: 3687})
```

- If our classifier always classify the end of sentence as True, the accuracy is very high:

```
>>> test_counter = Counter([f[1] for f in test_features])
>>> test_counter
Counter({False: 2, True: 405})
>>> 405/407
0.995085995085995
```

- High accuracies are common when there is unbalanced data.

See the separate notebook for the complete code and comments.

2 Multi-label Classification

Multilabel Text Categorisation

The Reuters-21578 Corpus

- http://about.reuters.com/researchandstandards/corpus/
- A collection of Reuters news stories.
- Each news stories has one or more topics.
- Available in NLTK.

Demo on classifying Reuters documents using NLTK:

- See related notebook.

Assigning Multiple Labels to a Document

- You can use independent classifiers, one per label.
- Each classifier is trained independently from the others.
- When you want to label a document, run each independent classifier.
- If classifier X assigns the positive class to the document, then label the document with label X.

Evaluation of Multilabel Text Categorisation

- Recall, Precision, F are designed for binary classification.
- In multilabel classification, we will need to evaluate each separate label and average the results.

Macro-averaged Recall (Precision, F)

- Average the evaluation across all labels.
- All labels have the same weight.
- But what if some labels may have very few samples?

Micro-averaged Recall (Precision, F)

- The evaluation is based on the total number of true positives, false positives, and false negatives.

3 Evaluation

Recap on Training and Testing

- Use the training set to train the classifier.
- Use a separate test set to evaluate the classifier.
- Make sure that there is no bias in the training or test sets.
- We often need a dev-test set if we are going to fine-tune our system.

Remember

- Never look at the test set to fine-tune your system.
- Never train the system using the test set.
- Do not evaluate using the training set unless you want to check for over-fitting.

Cross-Validation
If we do not have enough data we can use cross-validation.

N-fold Cross-Validation

1. Divide the data into N partitions called folds.
2. Keep fold 1 for testing and train with the other folds.
3. Repeat with folds 2, 3, ..., N.
4. Report the average of the results.
5. The standard deviation gives you an indication of the robustness of the system.

Cross-validation in Scikit-learn
http://scikit-learn.org/stable/modules/cross_validation.html

The Easiest Way
Scikit-learn provides several evaluation metrics that can be used for cross-validation, such as accuracy, F1, and many more.

```
>>> cross_val_score(crossval_classifier, train_features, train_labels,
                    cv=10, scoring="accuracy")
array([ 0.79338843,  0.80833333,  0.76666667,  0.79166667,  0.80833333,
        0.79166667,  0.825     ,  0.83333333,  0.79166667,  0.80672269])
```

Iterating over KFolds
If we want to have more flexibility on what to do on each iteration of cross-validation we can use the KFold iterator.
(see notebook for an example of use)

Take-home Messages

1. Create one-hot encoding of words for NLTK and Scikit-learn.
2. Design useful feature extractors in NLTK and Scikit-learn.
3. Design feature extractors that use context.
4. Train and evaluate statistical classifiers.
5. Compare with a baseline.
6. Perform cross-validation.

What's Next

Week 5

- Sequence Labelling.
- Friday: GOOD FRIDAY (Easter)
  No lecture, no workshop session

Reading

- NLTK Chapter 6.