Lab for week 7: Text Classification

Author: Ida Szubert
Author: Sharon Goldwater
Author: Henry S. Thompson
Date: 2018-10-24

This lab is available as a web page[2] or pdf document[3].

Goals and motivation of this lab

The idea of text classification is to assign labels to pieces of text, based on their content. Examples include:

- Classifying news articles by topic: sport, politics, fashion etc.
- Classifying student essays according to grade categories.
- Classifying tweets by language.
- Classifying books as potentially interesting or not for a particular customer.

In this lab we will demonstrate simple generative (Naive Bayes) and discriminative (logistic regression) classifiers, first on a toy data set and then on a more realistic text collection. Along the way you will be introduced to some powerful and widely used Python libraries for numerical computation (numpy) and machine learning (scikit-learn).

We want you to see some of the practical differences between the two methods and the types of features used in text classification. You'll also learn to experiment with feature design and practice evaluating results of your experiments.

Isn't this just like IAML?

We have tried to focus this lab on some issues that were not covered in the IAML labs. In particular, rather than giving you an existing set of preprocessed features, we'll lead you through some of the steps you might go through when choosing features for yourself.

Preliminaries

As usual, create a directory for this lab inside your `labs` directory and activate the environment:

[1] http://creativecommons.org/licenses/by-nc/4.0/.
[2] http://www.inf.ed.ac.uk/teaching/courses/anlp/labs/lab7.html
[3] http://www.inf.ed.ac.uk/teaching/courses/anlp/labs/lab7.pdf
[4] http://www.inf.ed.ac.uk/teaching/courses/anlp/labs/lab7.zip

```
cd ~/anlp/labs
mkdir lab7
cd lab7
conda activate anlp
```

Download the file lab7.zip[4] and unzip it in your `lab7` directory. You should find `lab7_toy.py`, `lab7_toy-sol.py`, `lab7_big.py`, and `lab7_helper.py`.

Two paths through this lab

This lab uses the `numpy` library. If you are already familiar with `numpy`, you have the option to implement parts of the lab yourself (extracting feature counts and implementing the core of Naive Bayes). If you and your partner agree to do this, you should start from **lab7_toy.py** and **do the steps marked (Opt)**.

Otherwise, we recommend that you just start from **lab7_toy-sol.py** and **skip the steps marked (Opt)**. However, you may want to come back later to look at the code in more detail. Looking at other people's code is a great way to learn more about programming.

Toy dataset

To start, we will explore a simple binary classification problem with an artificial dataset.

Start up Spyder and open either `lab7_toy-sol.py` or `lab7_toy.py` (see "two paths" above).

You should see the data consisting of 5 training and 4 test sentences. Each sentence describes a company, and the descriptions fall into two classes: *restaurants* (1), or *furniture stores* (0).

Look at the labelled training sentences. Which words do you think might be indicative of each class?

In addition to the words in the sentences, we will also consider the POS tags and lemmas.

Feature extraction

For the purposes of classification we need to represent a document as a collection of numerical features. Transforming a document into a vector of numerical features is called *vectorization*.

The simplest features to use for text are just the frequency counts of words, i.e., the "bag-of-words" representation.

(Opt) Complete the function `encode_token_freq()`, which produces a bag-of-words representation for a collection of documents, given a vocabulary.

Now, run the file and then use the following line to extract feature vectors for each sentence, along with the vocabulary for the word features:

```
(train_word_features, word_vocab) = vectorize(train_x_data)
```

The first return value `train_word_features` is a matrix, where each row holds the vector of features for the corresponding sentence in the data set.

What should be the count for word *fresh* for the first sentence? You can check that the feature vector agrees with your answer by inspecting the entry corresponding to the first sentence and the appropriate feature:

```
i = word_vocab.index("fresh")
train_word_features[0][i]
```

In addition to word counts, we can also extract counts of POS tags and lemmas:

```
(train_pos_features, pos_vocab) = vectorize(train_x_pos)
(train_lemma_features, lemma_vocab) = vectorize(train_x_lemma)
```

From `train_pos_features`, how can you get the feature count of the POS tag 'NNS' in the third training sentence?

We'll focus on the word-based representation for the remainder of the lab, but in the Going Further section you could consider tags and lemmas again if you want.

To encode the test data as feature vectors, run:

```
test_word_features = encode_token_freq(test_x_data, word_vocab)
```

Generative modelling: Naive Bayes

First, to remind yourself about Naive Bayes classifiers, answer the following questions:

- How is $P(class|features)$ computed in a Naive Bayes classifier? (Look back at the lecture slides if you can't remember).
- In this toy data set, what is the prior probability of each class if we use Maximum Likelihood Estimation?
- What is the equation to compute the feature probabilities using add-alpha smoothing?

(Opt) Complete the function `feature_probabilities()`, which estimates the class-conditional log probabilities of the features using add-alpha smoothing, and the function `nb_posterior_log_probability()`, which computes $logP(class|observation)$.

Run your NB classifier by calling:

```
nb_model = train_naive_bayes(train_word_features, train_y_data, 0.1)
nb_classify(test_word_features, nb_model)
```

Verify that you get the same results as the Naive Bayes classifier implemented in the `sklearn` library, using the same value of alpha:

```
nb_sk = MultinomialNB(alpha=0.1).fit(train_word_features, train_y_data)
nb_sk.predict(test_word_features)
```

Find the 3 highest probability words for each class using the `most_probable_features()` function, which is defined in `lab7_helper.py`:

```
most_probable_features(nb_model, word_vocab, categories, 3)
```

Do the results seem reasonable? Do you think those features are salient and discriminative? (You might already see a problem with looking at the most probable features. If not, it may become more clear once you get to the larger dataset.)

Discriminative model: Logistic regression

Discriminative classifiers, such as logistic regression, model $P(class|observation)$ directly, without the decomposition into conditional and prior probability. Instead of modelling all of the classes, logistic regression estimates the boundaries between the classes in the feature space.

Train and test an LR classifier:

```
lr_sk = LogisticRegression().fit(train_word_features, train_y_data)
lr_sk.predict(test_word_features)
```

The function which describes the boundary between the classes is a linear combination of weighted features. We can look at the *weights* of the trained LR model to find the features that are the most influential:

```
most_influential_features(lr_sk, word_vocab, categories, 6)
```

Are they similar or different to the most probable features of each class in the Naive Bayes model? Would you expect them to be similar? Why or why not?

Bigger dataset

For something much more realistic in both size and variability, let's look at a database of 12000 consumer complaints. The are four topics of complaints represented in this set: *credit card*, *bank account or service*, *student loan*, and *consumer loan*.

Your task will be to run a Naive Bayes and logistic regression classifiers using a variety of features. For this part of the lab we will be using sklearn implementations of classifiers and vectorization functions.

Reset the interpreter by running:

```
%reset
```

Open `lab7_big.py` in the editor and run it. This will load the data and extract a simple bag-of-words representation for the documents in the training and test sets, and print out how many features were extracted. (See the code under "Feature extraction" near the top of the file.)

Now train and test Naive Bayes and Logistic Regression classifiers:

```
nb_model, nb_predict = nb_fit_and_predict(train_x_features, train_y,
                         test_x_features, test_y, average="weighted")
lr_model, lr_predict = lr_fit_and_predict(train_x_features, train_y,
                         test_x_features, test_y, average="weighted")
```

Notice the difference in how long each model takes to train.

Let's check the 10 most probable and most influential features for the trained models (the data is anonymised and 'xxxx' stands for a redacted word):

```
most_probable_features(nb_model, feature_names, categories, 10)
most_influential_features(lr_model, feature_names, categories, 10)
```

You should now clearly see the difference between generative and discriminative approaches by looking at those features. Are the most probable features in a generative model likely to be discriminative?

We will now explore one of the problems with simple bag-of-words features. Check the vocabulary size:

```
len(feature_names)
```

As the datasets get large, feature vectors created using our simple approach get very high-dimensional. But it's unlikely that all the features are equally important, so we may want to exclude unimportant features. Let's see if there's a better way than trying to choose features by hand.

First, let's try a simple idea: use a pre-existing stop word set from `nltk`:

```
stopset = set(stopwords.words("english"))
```

Before going further, consider the following questions:

- Will removing stopwords reduce the size of the vocabulary by much?
- Will removing stopwords change the performance of the classifiers by much?

Now, check your predictions. Uncomment and run the relevant lines (78-92) in `lab_big.py` which (a) vectorize the data without using the stopwords, (b) train and test models on the new feature representation, and (c) print out the most probable and most influential features of those models.

Did the reduction in the number of features harm performance? Are the probable/influential features more sensible? Do you see any other improvements? Have we succeeded in reducing the size of the feature vectors much?

As an alternative to removing stopwords, we can decide automatically which features to get rid of. Here, we will try removing features that have low *variance* across classes: that is, where the counts of the feature are very similar across all classes.

Re-comment the lines from the previous section and un-comment lines 97-110, which train and test the models after filtering out low variance features. How many features are there now? Did this big reduction affect performance much? Can you detect any change in how fast the models run?

Going Further

1. Try to find a variance threshold such that the performance does not suffer. You can also combine stopword filtering with variance-based reduction.
2. Take a look at the length of the documents in this dataset. Plot a histogram of the lengths:

   ```
   plot_document_lengths(train_x_words)
   ```

   Are the lengths distributed fairly uniformly? Are the documents rather short, or rather long? What sort of transformations would you apply to count based features knowing that the lengths of the documents are varied, and assuming that the length itself is not correlated with class? Try using the code we wrote for normalising each feature vector to unit length and see what happens. Does normalization affect Naive Bayes and logistic regression performance equally?
3. There are many kinds of features other than word counts. At the end of `lab7_big.py` we presented options from `sklearn`: ngrams, binarized features, and tf-idf weighted count features. Try to get a better classification performance by playing with those options, and don't forget that you can also reduce and normalize. You could also try using lemmas or POS tags if you work with nltk to get those features.

   When you are experimenting, you can inspect the most probabile / most influential features to get a feeling of how sensible your classifiers are.