Second Year Project: Natural Language Processing and Deep Learning Week 2

Barbara Plank, Rob van der Goot

February 4, 2020

This is your assignment for the second week of the course *Second Year Project*. You are expected to work on sections 1 and 2 in the Tuesday exercise class, and section 3 in the Friday class.

After completing the whole assignment, you should be:

- able to implement and evaluate various *n*-gram language models, and use them to generate novel sentences
- be familiar with language modeling with *n*-grams, including treatment of unseen words and *n*-grams
- be familiar with the Naive Bayes classifier (pen and paper)
- able to implement a perceptron and logistic regression classifier for language identification and reflect on differences in performance evaluation

Requested reading: Chapters 3, 4, and 5 from *Speech and Language Processing* by Jurafsky and Martin [1] (for details per lecture, see our course page on LearnIT).

1 Language Modeling with *n*-grams

Complete the LM exercises provided in the notebook: `language_models.ipynb`. They contain implementation exercises as well as a pen-and-paper exercise.

2 Naive Bayes Classifier (pen and paper)

Solve the following exercises from Chapter 4 of [1]:

- Exercise 4.1 from J&M: What class will Naive Bayes assign to the sentence:
  I always like foreign films.?
- Exercise 4.2 from J&M (copied here for your convenience):
  Given the following short movie reviews, each labeled with a genre, either comedy or action:
  1. fun, couple, love, love comedy
  2. fast, furious, shoot action
  3. couple, fly, fast, fun, fun comedy
  4. furious, shoot, shoot, fun action
  5. fly, fast, shoot, love action

  and a new document D:

  `fast, couple, shoot, fly`

  compute the most likely class for D. Assume a naive Bayes classifier and use add-1 smoothing for the likelihoods.
- Exercise 4.3 from J&M (on whether a binarized or a multinomial Naive Bayes classifier agree on the provided sample data).

3 Language Classification

In this section, you will learn how to build a language classifier. The task is simple: given a language utterance, automatically predict in which language it is written.

For these exercises, we have prepared text data extracted from two wiki websites (Wikipedia and Wookieepedia). Three languages are present in the data:

1. Danish
2. Norwegian (Bokmål)
3. English

We provide one training dataset containing utterances about *Star Wars* in the three languages, as well as three development sets containing respectively:

- data extracted from a page also present in the training data (`star`)
- data from other pages about *Star Wars* (`starOther`)
- data from pages on other topics (`other`)

The files are named as follows: `<SPLIT>.<TOPIC>.txt`. For example, the file `dev.starOther.txt` contains the development data from *Star Wars* pages not present in the training data. All files contain two columns separated by a tab: the first column contains the language index and the second the utterance.

This task consists of four parts: (1) getting the features, (2) training the classifiers, (3) predicting on the development set, and (4) evaluation.

3.1 Get bag-of-words features

First, convert the data to word unigram features. Please, do *not* use a vectorizer from the *sklearn* package (`DictVectorizer`, `CountVectorizer`, or `TfidfVectorizer`). You can decide for yourself whether you want to tokenize and/or filter the features (using all words might not be the best approach).

Tip: Start with one sentence, as it is much easier to debug.

3.2 Training the classifiers

Train a perceptron classifier, you can make use of the *scikit-learn* implementation. For more information, see: Perceptron (you need the `fit` function). Note that the input is a list of lists of features x and a list of corresponding gold labels y. Therefore, following should hold `len(x) == len(y)` and their indices should match. Additionally, every instance in x should have the same length (the number of features).

Next, train a logistic regression classifier in a similar fashion. For more information, see: LogisticRegression.

3.3 Predict on the development data

Run both of the classifiers on all 3 development datasets (total of 6 runs). It is crucial that you ensure that the feature values have exactly the same order as during training. Also note that you cannot introduce new features here (!): you have to use the exact same features as the ones used during training.

Save the accuracies. Are the classifiers robust to transfer to pages not in training data? How about pages from another domain?

Hint: If the accuracy is lower than 50%, you are probably mixing up the feature order, either during training or during development or both.

3.4 Evaluation

There are two obvious ways to inspect the classifiers in more detail: by calculating a confusion matrix and by examining the feature weights.

Confusion matrix

Plot a confusion matrix for both classifiers when used on the pages on other topics, and inspect the differences (it is not important how you visualize the results: a table, a figure, or even an ASCII table will suffice). Are there any interesting differences?

Feature weights

In *scikit-learn*, you can inspect the internal weights given to each feature in the `.coef_` variable. Inspect the most important features for both classifiers. Are there any interesting differences?

Hint: The weights are given per class, so you can either inspect three lists, or compute the average importance (make sure to use the absolute feature values for the average).

References

[1] Jurfasky and Martin, In Preparation. *Speech and Language Processing (3rd ed. draft)*. Available at https://web.stanford.edu/~jurafsky/slp3/