NLP Coursework 1: Deception on Amazon

January 26, 2018

Consumers tend to rely heavily on reviews when making decisions about what to buy online. For a company like Amazon, which depends on this process, it is therefore particularly important that these reviews can be trusted. Your task will therefore be to develop an method for automatically classifying Amazon reviews as real or fake, to explore how plausible it is to automate this task. You will be working with a recently released corpus of Amazon reviews which have been manually analysed and annotated by the company itself (see `https://s3.amazonaws.com/amazon-reviews-pds/readme.html`)[1]. Along with the review texts, which are labelled as either *fake* (`__label1__`) or *real* (`__label2__`), the data set contains a series of other features for each review (*rating, verified purchase, product category, product ID, product title, review title*). The corpus is made up of 21,000 reviews, equally distributed across product categories, which have been identified as 'non-compliant' with respect to Amazon policies.

In this coursework, you will implement a Support Vector Machine classifier (or SVM) that classifies the reviews as real or fake. You will use both the review text and the additional features contained in the data set to build and train the classifier on part of the data set. You will then test the accuracy of your classifier on an unseen portion of the corpus.

You will be provided with the data set (`amazon_reviews.txt`) and a template file (`ex1_part1_template.py`) that you will use as a starting point for this coursework. When completed, you must submit two things:

- Your completed Python code;
- A PDF document describing what you did (instructions below).

The deadline for this coursework (Part A and B) is **Monday 12th February 2018 23:59:59**. Please submit via the module QMPlus page.

[1] See https://s3.amazonaws.com/amazon-reviews-pds/LICENSE.txt for the licensing information and terms and conditions for the use of the dataset.

Part A: Deception detection (60 points)

The template file contains some functions to load in the dataset, but there are some missing parts that you are going to fill in.

1. **(5 points)** Start by implementing the `parseReview` and the `preProcess` functions. Given a line of a tab-separated text file, `parseReview` should return a triple containing the identifier of the review (as an integer), the review text itself, and the label (either 'fake' or 'real'). The `preProcess` function should turn a review text (a string) into a list of tokens.

   Hint: you can start by tokenising on white space; but you might want to think about some simple normalisation too.

2. **(10 points)** The next step is to implement the `toFeatureVector` function. Given a preprocessed review (that is, a list of tokens), it will return a Python dictionary that has as its keys the tokens, and as values the weight of those tokens in the preprocessed reviews. The weight could be simply the number of occurrences of a token in the preprocessed review, or it could give more weight to specific words. While building up this *feature vector*, you may want to incrementally build up a global `featureDict`, which should be a list or dictionary that keeps track of all the tokens in the whole review dataset. While a global feature dictionary is not strictly required for this coursework, it will help you understand which features (and how many!) you are using to train your classifier and can help understand possible performance issues you encounter on the way.

   Hint: start by using binary feature values; 1 if the feature is present, 0 if it's not.

3. **(15 points)** Using the `loadData` function already present in the template file, you are now ready to process the review data from `amazon_reviews.txt`. In order to train a good classifier, finish the implementation of the `crossValidate` function to do a 10-fold cross validation on the training data. Make use of the given functions `trainClassifier` and `predictLabels` to do the cross-validation. Make sure that your program stores the (average) precision, recall, f1 score, and accuracy of your classifier in a variable `cv_results`.

   Hint: the package `sklearn.metrics` contains many utilities for evaluation metrics - you could try `precision_recall_fscore_support` to *start with.*

4. **(15 points)** Now that you have the numbers for accuracy of your classifier, think of ways to improve this score. Things to consider:

   - Improve the preprocessing. Which tokens might you want to throw out or preserve?
   - What about punctuation? Do not forget normalisation and lemmatising - what aspects of this might be useful?
   - Think about the features: what could you use other than unigram tokens from the review texts? It may be useful to look beyond single words to combinations of words or characters. Also the feature weighting scheme: what could you do other than using binary values?
   - You could consider playing with the parameters of the SVM (cost parameter? per-class weighting?)

   Report what methods you tried and what the effect was on the classifier performance.

5. **(15 points)** Now look beyond textual features of the review. The data set contains a number of other features for each review (*rating, verified purchase, product category, product ID, product title, review title*). How can the inclusion of these features improve your classifier's performance? Pick three of these metadata types to use as additional features and report how they improve the classifier performance.

Continue with Part B below.

Part B: Data Exploration (40 points)

1. **(5 points)** When you are convinced your classifier works well (think about what are acceptable accuracy scores!), spend some time exploring the data. Are there any interesting correlations between the review class (fake or real) and product category? What about review rating? Is verified purchase a useful indicator for whether the review is genuine?

2. **(15 points)** It is also interesting to consider whether there is anything intrinsically different about the ways in which the fake reviews are written when compared with the genuine ones. To examine this, you will now explore some of the linguistic and stylistic traits of the reviews and compare the two classes. Think about the following areas (and use the original raw data set to preserve the stylistic features):

   - On average, how long are the reviews for each class? Does one class use more complex vocabulary than the other (consider word length, but also more complex measures of reading level e.g. the Flesch-Kincaid readability test)?
   - Does one class contain more stopwords than the others? What about use of capslock and punctuation?
   - Does the product name appear in the review? If so, does this happen more or less often in genuine and fake reviews?

     Hint: You could play around with regular expressions to search for (variations of) the product name in a review.
   - Are there any other surface features you can think of that may be interesting to compare?

   Write up your findings in your report.

3. **(20 points)** You will now look at the sentiment of the reviews in your data set. Build a sentiment classifier using the review rating as the sentiment gold standard. Treating this as a binary classification task, you can consider a 1-2 star review as negative, and a 4-5 star review as positive. In order to achieve a roughly balanced data set, you may want to remove some of the positive reviews.

   Hint: Try retraining your deception classifier for this task - how does it perform? Think of which features may be better suited for detecting sentiment instead of deception.

Some tips and tricks

Virtual Environments

To be able to work with everything Python has to offer, a wise thing to do is to install a *virtual environment* on your Lab machine. This creates a local separate Python installation in which you have the power to install any package you may need. On the ITL machines:

- `pip install virtualenv --user`
- `python3 -m venv NAME` where `NAME` is the name you want to give this new virtual environment;[2] Don't worry if you get a message `Unable to symlink...` , your virtual environment should be set up.
- `source NAME/bin/activate` to activate it. You are now in the new virtual environment (and it should show in your terminal prompt).
- when you want to leave it, type `deactivate`.

If you're using your own machine, see the instructions at:

`http://docs.python-guide.org/en/latest/dev/virtualenvs/`

for details on how to set up a virtual environment.

Once you are in your virtual environment, install the numpy, scipy, scikit-learn, nltk (and unicodecsv if you're using Python 2.7) packages:

- `pip install --upgrade pip`
- `pip install numpy scipy scikit-learn nltk`

List of Python packages you may want to look into

- `nltk.classify` contains several classifiers that act as wrappers around known implementations. Pay particular attention to the `SklearnClassifier` which supports (linear) SVC.
- `re` is a package that has support for regular expressions and also substitutions based on word grouping. You may want to use this for preprocessing.
- `sklearn` contains several tools for classifying, cross validation, and reporting on accuracy scores of a classifier. Pay attention to `sklearn.metrics`.

[2] For Python 2.7, the following "should" work:

```
python ~/.local/lib/python2.7/site-packages/virtualenv.py NAME
```