Movie reviews

Machine Learning

Claudio Cusano

A.A. 2021/2022

Text classification finds application in countless domains, from spam filters in email clients, to document classification in search engines. An interesting case of text classification is that of *sentiment analysis*. In sentiment analysis we want to predict the emotional state of the writers of text messages. It is a classification problem, where the classes are the authors' emotions (angry, delighted, sad, happy, disgusted...). Sentiment analysis have a large range of applications of significant economical and social value. It can be used as a cost-effective way of measuring the opinion of people by processing their comments about products, books, tv shows, political parties, etc.

1 Lab activity

In this lab exercise we will analyze movie reviews written by the users of the IMDB website (`www.imdb.com`). We will try to predict the polarity of the reviews distinguishing the positive comments from the negative ones.

Before starting the exercise be sure to read the "Preliminaries" section here below, download the data set and review the relevant classification models.

1.0 Preliminaries

We will use the data set that has been introduced in the paper: "Learning Word Vectors for Sentiment Analysis," by Andrew L. Maas et al. It includes 50 000 reviews equally divided among the two classes. You can download it from the Kiro page of the course.

The data set is already divided in a training set of 25 000 reviews, a validation set of 12 500 and a test set of 12 500. A subset of the training set is also available to allow faster experiments.

We will build a multinomial Naïve Bayes classifier. If you are not familiar with this model, please review it before starting the lab activity.

1.1 Build a vocabulary

Write a Python script that reads all the training data and use it to build a list of the n most frequent words (hint: use the `Counter` class from the `collections` module in the

standard library[1]). Write the words in a file named `vocabulary.txt`. Let's start with a relatively small vocabulary (say, $n = 1000$). Later on you will repeat the experiments with more words.

1.2 Extract the features

One of the most common representations for text data is *Bag of Words* (BoW). It consists in building the feature vectors as the vector of counters of the occurrences of the words in the vocabulary.

Write a Python script that compute the BoW representations of a set of documents, resulting in a matrix where each row represents a different document. At the same time, build a vector with the binary labels associated to the documents. Finally, save features and labels in a file for later use.

WARNING: this may result in a very large file! It can be reduced to a manageable size by either (i) use `numpy.savetxt` specifying a file name with extension '`.gz`' or (ii) use the function `numpy.save_compressed` (the first produces a compressed text file, while the second a compressed binary file).

Apply the process to the train, validation and test data.

1.3 Train a classifier

Multinomial Naïve Bayes is a simple classification model that is especially designed to work with histogram representations such as BoW. Given a BoW feature vector $\mathbf{x}$ the multinomial NB model predicts the class $\hat{y}$ defined as follows:

$$\hat{y} = \underset{y=0,\ldots,k-1}{\arg\max} \sum_{j=0}^{n-1} x_j \log \pi_{y,j} + \log P(y),$$

where $\pi_{y,j}$ represents the probability that a random word taken from a document of class y is the j-th word in the vocabulary, and where $P(y)$ is the prior probability for class y.

In the particular binary classification case ($k = 2$) this means that class 1 will be predicted if and only we have:

$$\hat{y} = 1 \iff \sum_{j=0}^{n-1} x_j \log \pi_{1,j} + \log P(1) > \sum_{j=0}^{n-1} x_j \log \pi_{0,j} + \log P(0),$$

and this condition can be rewritten as:

$$\hat{y} = 1 \iff \sum_{j=0}^{n-1} x_j (\log \pi_{1,j} - \log \pi_{0,j}) + (\log P(1) - \log P(0)) > 0,$$

or even:

$$\hat{y} = 1 \iff \mathbf{w}^T \mathbf{x} + b > 0, \;\; w_j = \log \pi_{1,j} - \log \pi_{0,j}. b = \log P(1) - \log P(0).$$

[1] `https://docs.python.org/3.8/library/collections.html#collections.Counter`

Write a Python script that estimates the probabilities $\pi_{y,j}$ to build a binary classifier. Estimate its training and test accuracy.

1.4 Variants

Try the following variants of the method:

1. make it so the model ignores very common words (you can find a list in the `stopwords.txt` file);
2. make it so the model does not distinguish among words with the same root (such as "run", "running", "runners"); you can use the `stem` method in the `porter.py` module. This technique is called *stemming*.

Observe how these modifications affect the performance of the model.

1.5 Analysis

Identify the set of most impactful words on the predictions made by the model.

Also identify the worst errors on the test set, that is, those that the model classifies with the highest confidence.

2 Assignment

As homework, review and refine the scripts programmed in the lab activity. In addition, perform the following exercises.

2.1 Vocabulary size

Repeat the main steps by changing the size of the vocabulary. Does it influence the results? How much?

2.2 Classifier (optional)

Try to build and train a different linear model (logistic regression or SVM). How does it compare to multinomial Naïve Bayes?

2.3 Report

Prepare a report of one or two pages with the answers to document all the experiments and their results. The report must be in the PDF format. Include your name in the report and conclude the document with the following statement: "I affirm that this report is the result of my own work and that I did not share any part of it with anyone else except the teacher."

Make a ZIP archive with the report and the python scripts you used and and upload it on the course website.