ML Commando Course 2018

Session 2a - Text Classification with Naïve Bayes

Russell Moore

ALTA / Computer Laboratory

July 2018

Update your repo

Activate your virtualenv and run:

```
pip install pydotplus
```

Get the latest notebook files:

```
cd c:\git\ml_commando_2018\
git fetch origin master
```

Newsgroups

Back before the World Wide Web (when dinosaurs roamed the Earth, around 1980) there was *Usenet*.

Newsgroups allowed early internet dwellers to be rude to each other online.

Slightly like reddit, Newsgroups have hierarchical topics such as `comp.os.ms-windows.misc` and `talk.politics.guns` to help you find the right people to insult.

There are still around 20,000 active Newsgroups. Here we will use archived discussions to classify data by content.

Convert text to numbers...

Our machine learning algorithms can work only on numeric data. We must convert our text-based dataset to a numeric dataset.

The text content of the message is like a single ‘entangled feature’ - but we want to convert this into a set of meaningful numeric features.

Intuitively one could try to look at which words (or more precisely, tokens, incl. numbers or punctuation signs) are used in each of the text categories, and try to characterise each category with the frequency distribution of each of those words.

The `sklearn.feature_extraction.text` module has some useful utilities to build numeric feature vectors from text documents.

Word counts

Simple idea - just count the words in a document to get some idea of the content of that document:

An article about pets might have [Cat=9, Dog=8, Fleas=5, $\cdots$]

An article about pubs might have [Beer=9, Billiards=4, Fleas=2, $\cdots$]

Not all words are equally informative - words like "a" and "the" occur often. We usually ignore them as they provide little information (stopwords).

TF-IDF

Even words that do provide information are not all equal.
TF-IDF is a weight to measure how important a word is to a particular document in a corpus. Invented in Cambridge in 1972!

Term frequency, $TF_{w,d} = \#$times word w in document d

Inverse document frequency for N docs, $IDF_w = log(\frac{N}{DF_w})$

Weighted wordcount in a document becomes:
$TDFIF_{w,d} = TF_{w,d}.IDF_w$

IDF is loosely grounded in information theory, but is also just a good heuristic. Something like 80% of library search software uses TF-IDF as the weighting for documents.

Naive Bayes

A ‘Naive Bayes’ classifier uses the probabilistic relation between observed features (e.g. words) and classes (e.g. topics).

Naive Bayes is fast and often very effective.

Very widely used in search, information retrieval, spam filtering etc.

Bayes' Theorem

First cited by Thomas Bayes (1701-1761):

$$Pr(c|e) = \frac{Pr(e|c)Pr(c)}{Pr(e)}$$

Extremely important, comes up everywhere. Bayesian stats treats probabilities as degrees of belief, not just counts.

Bayesian Inference

Sometimes people use the terms:

$$\textit{posterior} = \frac{\textit{likelihood} \times \textit{prior}}{\textit{evidence}}$$

The reason is that you can iterate across Bayes' rule. Start with a 'prior' belief about distribution of classes, gather information on the evidence and likelihood, to update this belief. Then repeat...

Bayes in practice

For evidence, take a vector of features (such as word counts in a document):

$$\vec{x} = [x_0, x_1, \cdots, x_n]$$

Apply Bayes' rule to get:

$$Pr(c|\vec{x}) = \frac{Pr(\vec{x}|c)Pr(c)}{Pr(\vec{x})}$$

'Multinomial' Bayes simply means we are dealing with some number of classes > 2

A Note on ‘Naivety’

The name ‘naive’ refers to an assumption that the elements of $\vec{x}$ are probabilistic independent from one another.

In practice this isn’t true (‘government’,‘spending’ more likely to occur together than ‘government’,‘milkshake’).

The result of this is that the posterior probabilities are not always good. **However, the classes they predict often are.** Frequently that is fine for classification.

Speedy and easy to interpret, often a good first choice of classifier. Not so naive when used non-naively.

Smoothing

If during classification, we see a piece of evidence (e.g. word) we never saw in training, how would we classify it?

We can't - zero probabilities start appearing and causing us problems.

One way to avoid this is to use 'LaPlacian' or additive smoothing. This just adds a small value (alpha) on to every count, so that it cannot be zero. Intuition is that it treats 'never-seen' evidence as 'rarely-seen'.

Lots of other types of smoothing are available. Remain vigilent for cases where 'zero counts' may occur!