Business Analytics

Lecture 2: Sentiment Analysis

Ulrich Wohak[1]

[1]Department of Economics
Vienna University of Economics and Business

Introduction

- We will encounter our first business application of NLP: Sentiment Analysis
- Core idea: We want to know (guess) whether a text has a positive or negative sentiment (sometimes we are also interested in neutral)
- Two basic approaches: Rule-based vs. Statistical
- Before we dive into technicals/code, let's have a look at some examples

Text examples for sentiment analysis

One classic application of sentiment analysis is to classify product reviews. An example product review from Amazon:

> *This monitor is definitely a good value. Does it have superb color and contrast? No. Does it boast the best refresh rate on the market? No. But if you're tight on money, this thing looks and preforms great for the money. It has a Matte screen which does a great job at eliminating glare. The chassis it's enclosed within is absolutely stunning.*

Rule-based approach (i)

- Not always trivial to classify the sentiment of a text
- How can we go about this in a systematic way?
- We'll look at a rules-based approach first using **VADER:** **V**alence **A**ware **D**ictionary for s**E**ntiment **R**easoning (Hutto and Gilbert, 2014)

Rule-based approach (ii)

- VADER contains a lexicon that associated tokens with a 'sentiment' score
- It counts the number of tokens associated with each category: {pos, neutral, neg}
- Based on this association, we can calculate a sentiment score of the text.
- Let's look at some code!

Statistical approach (i)

- We will have a look at our first **supervised** machine learning algorithm that will help us (hopefully) to classify the sentiment of a text
- A supervised machine learning algorithm builds/estimates a model using (training) input *and* desired output data
- That is, we need a data set of the form $[\mathbf{y}, \mathbf{X}]$
- We will talk about unsupervised machine learning algos later in the course

Naive Bayes Classifier

- First introduced by Maron (1961)
- It applied Bayes' Theorem with strong (naive) independence assumptions between the features (aka tokens)
- In effect, when we apply the classic naive Bayers' Classifier, we assume: (i) independence between tokens, (ii) each word is equally 'important'

Naive Bayes (i): Brief intro

- Let us consider a vector **x** that represents (the features of) a document in our corpus:

$$x_i = (x_1, x_2, ..., x_n)$$

where n could be the size of our vocabulary V, $|V|$.
The naive Bayes' Model assigns to each x_i (representing d_i) the probabiity

$$p(C_k|\mathbf{x}) = p(C_k|x_1, x_2, ..., x_n) \tag{1}$$

for each of the k possible outcomes. In our case, we can think of C_k as sentiment categories, i.e. *positive, neutral, negative*.

Naive Bayes (ii): Brief intro

- Bayes' Theorem tells us that we can decompose this conditiona probability in the following way:

$$p(C_k|\mathbf{x}) = \frac{p(C_k)p(\mathbf{x}|C_k)}{p(\mathbf{x})} \tag{2}$$

Since $p(\mathbf{x})$ is constant (i.e. independent of C_k), we can simplify this expression to

$$p(C_k|\mathbf{x}) \propto p(C_k)p(\mathbf{x}|C_k) \tag{3}$$

- Let's get our intution right:
  a. Naive Bayes' tries to find keywords in a set of documents that are predictive of the target (output) variable
  b. The internal coefficients will try to map tokens to scores (not that in stark contrast to econometric methods, we **do not care** about the individual coefficients)

Naive Bayes (iii): Let's code!

We can derive an estimating equation for and then estimate this model via maximum likelihood, but we will omit the technical details.

Let's jump right into the code!