

L90 Practical: Part I

Helen Yannakoudakis¹

November 8, 2017

¹Adapted from Simone Teufel's slides



The Task: Sentiment Classification

- Write Code that automatically classifies a given review of a movie into Negative or Positive
- Obviously without looking at the Star Rating :)
- Data: 1,000 POS + 1,000 NEG example texts
- Use Virtual Machines (VM) on Mphil machines; submit code running on these



- Today:
 - How to develop the baseline system(s)
 - How to write the (baseline) report
- Nov 15:
 - Check on your baseline system
 - Work on your 1,000-word baseline report (ticked)
- Nov 17: Submit baseline report
- Nov 24: I will give you feedback by email
- Nov 29: How to develop the extension system (doc2vec)
- Jan 17: Submit 4,000-word report on the extension system



Symbolic System – First Baseline

- Use existing sentiment lexicon
- Count and sum number of matches between text and lexicon (pos–neg)
- Variation of system – Use Magnitude to sum matches
- But: Does using magnitude produce better results?
- Better in science means **statistically better**



Statistical Significance Testing

- Null Hypothesis: two result sets come from the same distribution
 - System 1 is (really) equally good as System 2
- First, choose a **significance level** (p), e.g., $p = 0.01$
- We then try to reject the null hypothesis with at least probability $1 - p$ (99% in this case)
- Rejecting the null hypothesis means showing that the observed result is very unlikely to have occurred by chance
- If we manage to do so, we can report a statistically significant difference at $p = 0.01$
- Now – a particularly nice and simple test (**non-parametric and paired**): the sign test



Sign Test (non-parametric and paired)

- The sign test uses a **binary event model**
- Here, events correspond to documents (2,000 events in our case)
- Events have binary outcomes:
 - **Positive:** System 1 beats System 2 on this document
 - **Negative:** System 2 beats System 1 on this document
 - (**Tie:** System 1 and System 2 could also have the identical result on this document – more on this later)
- Call the probability of a positive outcome q (here $q = 0.5$)
- Binary distribution allows us to calculate the probability that, say, at least 1,247 out of 2,000 such binary events are positive
- Or otherwise the probability that at most 753 out of 2,000 are negative



Binomial Distribution $B(N, q)$

- Probability of $X = k$ negative events out of N :

$$P_q(X = k|N) = \binom{N}{k} q^k (1 - q)^{N-k}$$

- At most k negative events:

$$P_q(X \leq k|N) = \sum_{i=0}^k \binom{N}{i} q^i (1 - q)^{N-i}$$



Binary Event Model and Statistical Tests

- If the probability of observing our events under Null Hypothesis is very small (smaller than our pre-selected significance level p , e.g., 1%), we can safely reject the Null hypothesis
- The calculated $P(X \leq k)$ directly gives us the significance level p we are after
- This means there is only a 1% chance that System 1 does not beat System 2



Two-Tailed vs. One-Tailed Tests

- So far we've been testing difference in a specific direction:
 - The probability that, say, at least 1,247 out of 2,000 such binary events are positive [One-tailed test]
- A more conservative, rigorous test would be a non-directional one (though some debate on this)
 - Testing for statistically significant difference regardless of direction [Two-tailed test]
 - This is given by $2P(X \leq k)$ (because $B(N, 0.5)$ is symmetric)
 - We'll be using the two-tailed test for the practical



Don't Ignore Ties

- When comparing two systems in classification tasks, it is common for a large number of ties to occur
- Disregarding ties will tend to affect a study's statistical power
- Here we will treat ties by adding 0.5 events to the positive and 0.5 events to the negative side (and round up at the end)
- Sign test implementation details in instructions handout



- Naive Bayes Classifier:

$$\hat{c} = \operatorname{argmax}_{c \in C} P(c|\vec{f}) = \operatorname{argmax}_{c \in C} P(c) \prod_{i=1}^n P(f_i|c)$$

Feature vector \vec{f} ; most probable class \hat{c} ; C set of classes.

- Write code that calculates $P(f_i|c)$ for each f_i of \vec{f} , using only the Training Set
- (Why not the Test Set too?)
- Then apply the classifier to the Test Set.
- We don't need to worry about calculating $P(c)$ – why not?
- When you design your data structures, please think about later parts of this practical where you will dynamically split data into Training and Test on the fly, and train on different sets each time (Cross-Validation)



- When using a NB classifier, you need to consider what to do with unseen words – words which occur in the development/test data but which do not occur in the training data at all
- Modify your calculation of log probabilities to use Laplace smoothing (add a small positive constant κ)
- Smoothing in general: Instead of

$$\frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)}$$

Use

$$\frac{\text{count}(w_i, c) + \kappa}{\sum_{w \in V} \text{count}(w, c) + \sum_{w \in V} \kappa}$$

- You will see a big improvement in NB Classification with Smoothing



N-Fold Cross-Validation: Splitting

- Split data into N splits / folds / chunks
- Different strategies for splitting:

- Consecutive splitting:

cv000–cv099 = Split 1

cv100–cv199 = Split 2

...

- Round-robin splitting (mod 10):

cv000, cv010, cv020, ... = Split 1

cv001, cv011, cv021, ... = Split 2

...

- Random sampling/splitting: Not used here (but you may choose to split this way in a non-educational situation)



N-Fold Cross-Validation

- For each split X – use all others for training, test on split X only
- The final performance is the average of the performances for each fold/split
- Apply sign test across splitting methods – why does it make sense to do so? What does it tell us if we pass this test? (optional)
- BTW, the way we have chosen to distribute data among splits is called **stratified** cross-validation (each split mirrors the distribution of classes observed in the overall data)



Relationship to Pang et al. (2002)

- Your rough target for the baseline is a simplified implementation of Pang et al (2002)
- But: you are doing some extra things Pang et al didn't do (e.g., lexicon lookup)
- And Pang is doing much more than you do for the baseline:
 - Stemming
 - POS-tagging
 - Ngrams as features
 - Support Vector Machines (SVM) Classifier
- You **can** do these things for the baseline, but you are not expected to



Thank you!

