Sentiment analysis part 2

Module 2

THE UNIVERSITY of ADELAIDE

Plan for today

- Text representation
  - Sub-words
  - Byte-Pair Encoding (BPE)
- Categorisation methods:
  - Review of Bayesian classification
  - Multinomial Naïve Bayes
  - Smoothing
    - Laplace for classification
    - Kneser-Ney for n-grams
  - Other categorization methods: multilabel and stream categorization
- Sentiment analysis

Text representation

Questions

- Difference between One Hot and BOW encoding
- What additional information is captured by n-gram, but not BOW?
- Which is faster: stemming or lemmatisation? Which is more accurate?
- Should stop words be removed?
  - Which are stop words in this sentence?
    “The well is very deep”

Dealing with rare words in BOW

- Consider the following scenario:

  Word ***w1*** (called out-of-vocabulary word OOV) occurs in test instance, but not in training set. How can we use similar words to possibly improve the classification?

1. Find word vector for ***w1*.**
2. Find the most similar word ***w1'*** to ***w1*** in training set vocabulary.
3. Add ***w1'*** to the test instance.
4. Classify the instance

- May not always work, e.g. WV king, queen.
- Use Byte Pair Encoding for subwords

Heap, B., Bain, M., Wobcke, W., Krzywicki, A. and Schmeidl, S., 2017. Word vector enrichment of low frequency words in the bag-of-words model for short text multi-class classification problems. *arXiv preprint arXiv:1709.05778*.

Sub-Words

Word representation cannot handle unseen word or rare word well. Character-level representation is one of the solutions to overcome out-of-vocabulary (OOV). However, it may be too fine-grained and miss some important information.

Subword is in between word and character. It is not too fine-grained while able to handle unseen words and rare words.

- Byte Pair Encoding (BPE: Jurafsky et al. ch. 2.4.3)
- WordPiece: similar to BPE (Jurafsky, ch. 13.2.1)
- We will discuss BPE

THE UNIVERSITY of ADELAIDE

Tokeniser: Byte Pair Encoding: BPE

- The dataset is divided into training and test

Token learner

- The algorithm begins with the set of symbols equal to the set of characters. Each word is represented as a sequence of characters plus a special end-of-word symbol.
- At each step of the algorithm, we count the number of symbol pairs, find the most frequent pair (‘A’, ‘B’), and replace it with the new merged symbol (‘AB’).
- Add the new merged token to the vocabulary
- We continue to count and merge, creating new longer and longer character strings, until we’ve done k merges; k is a parameter of the algorithm.

Token segmenter

- We apply character strings found in the training set to the test set

Byte Pair Encoding example

Corpus and Vacabulary:

Corpus	Vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w
2 l o w e s t _	
6 n e w e r _	
3 w i d e r _	
2 n e w _	

THE UNIVERSITY of ADELAIDE

Byte Pair Encoding example cont.

Step 1

Corpus

```
5 l o w _
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _
```

Vocabulary

```
_, d, e, i, l, n, o, r, s, t, w, er
```

Step 2

Corpus

```
5 l o w _
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _
```

Vocabulary

```
_, d, e, i, l, n, o, r, s, t, w, er, er_
```

Byte Pair Encoding example cont.

Step 3

Corpus

```
5 l o w _
2 l o w e s t _
6 ne w er_
3 w i d er_
2 ne w _
```

Vocabulary

```
_, d, e, i, l, n, o, r, s, t, w, er, er_, ne
```

8 steps, k #merges k=8

Next 5 steps merges	**Current Vocabulary**
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	**_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_**

THE UNIVERSITY of ADELAIDE

BPE algorithm

This is token learner part

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Figure 2.13 The token learner part of the BPE algorithm for taking a corpus broken up into individual characters or bytes, and learning a vocabulary by iteratively merging tokens. Figure adapted from Bostrom and Durrett (2020).

This is token segmenter part

- Apply learned tokens to a new text n same order as learned tokens
- Frequencies of words don't matter in segmenter
- Replace shorter tokens with longer from the learned tokens
- Rare words will get segmented and common subwords identified.

Naïve Bayes vs Multinomial Naïve Bayes

Exercise on Bayesian rule

$$P(h|X) = \frac{P(X|h)\cdot P(h)}{P(X)}, \qquad likelihood = P(X|h)\cdot P(h)$$

$$h_{MAP} = \arg\max_{h\in H} P(h|X) = \arg\max_{h\in H} \frac{P(X|h)\cdot P(h)}{P(X)} = \arg\max_{h\in H} P(X|h)\cdot P(h)$$

A patient takes a lab test and the result comes back positive. The test returns a correct positive result (+) in only 98% of the cases in which the disease is actually present, and a correct negative result (-) in only 97% of the cases in which the disease is not present. Furthermore, 0.008 of the entire population have this cancer.
P(cancer) = 0.008, P(not cancer) = 0.992, these are called prior probabilities
P(+ | cancer) = 0.98 P(- | cancer) = 0.02
P(+ | not cancer) = 0.03 P(- | not cancer) = 0.97
Does the patient have cancer or not? To answer this, calculate P1 = P(not cancer | +) and P2 = P(cancer | +), and then calculate likelihood ratio P1/P2. If ratio > 1, the patient does not have cancer (according to this calculation)

How to Classify Using Naïve Bayes

1. **Using training dataset:**
   a) For each class, calculate *P(h)* as $\frac{n_h}{n}$, where *n* is number of instances, n_h is number of instances in class *y* for hypothesis *h*
   b) For each feature in training set, calculate $P(x_j|h)$ as $\frac{n_j}{n_h}$, where n_h is number of instances in class *h*, n_j is number of instances in class *h* with feature x_j
   c) Store the above calculations for later
2. **Given test instance with features : $\boldsymbol{X} = \{x_1, \dots, x_j, \dots\}$:**
   a) Retrieve all *P(h)* and all $P(x_j|h)$ for these features
   b) Calculate $P(X|h) \cdot P(h) = P(h) * P(x_1|h) * \cdots * P(x_j|h) * \cdots$ for all features *x* in the test instance, and all hypotheses *h*, corresponding to all *m* classes. So we will have a vector of *m* values, one value for each class
   c) Take arg max of that vector, as the predicted class

NB calculations: text example

Data (words are just letters):

Spam	Ham
b b b	a b a b a b a
b c b b c c	a c a c a c a
a a a	a a a
b a b a b	-

We want to classify a new document into HAM or SPAM

Credit: Machine Learning by Dr Michael Bain, UNSW Sydney

BNB calculations: example

	a	b	c	class
e_1	0	1	0	+
e_2	0	1	1	+
e_3	1	0	0	+
e_4	1	1	0	+
e_5	1	1	0	−
e_6	1	0	1	−
e_7	1	0	0	−
e_8	0	0	0	−

- BNB training:
  - For each class: num docs with word/num all docs
  - Spam: P(a|+), P(b|+), P(c|+) = (2/4, 3/4, 1/4) = (0.5; 0.75; 0.25)
  - Ham: P(a|-), P(b|-), P(c|-) = (3/4, 1/4, 1/4) = (0.75; 0.25; 0.25)

Testing for instance x=(a b -) = (1,1,0):

Classify x=(a b -) using MAP:

- P(x|+) = 0.5 *0.75 *(1-0.25) = 0.28;
- P(x|-) = 0.75 *0.25*(1-0.25) = 0.14
- Result: SPAM

Classify using ML: same, since P(-) = P(+) = 0.5

BNB calculations: example with 0

	a	b	c	class
e_1	0	1	0	+
e_2	0	1	1	+
e_3	0	0	0	+
e_4	0	1	0	+
e_5	1	0	0	−
e_6	1	0	1	−
e_7	1	0	0	−
e_8	0	0	0	−

- BNB training:
  - For each class: num docs with word/num all docs
  - Spam: P(a|+), P(b|+), P(c|+) = (0/4, 3/4, 1/4) = (0.0; 0.75; 0.25)
  - Ham: P(a|-), P(b|-), P(c|-) = (3/4, 0/4, 1/4) = (0.75; 0.0; 0.25)

Testing for instance x=(a b -) = (1,1,0):

Classify x=(a b -) using MAP:

- P(x|+) = 0.0 *0.75 *(1-0.25) = 0;
- P(x|-) = 0.75 *0.0*(1-0.25) = 0
- Result: ?

Smoothing

Why we need Smoothing?

- Some words do not appear in some training classes,
  e.g. if c column has all 0's in (+)
  - Solution: probabilistic smoothing (Laplace smoothing we will talk about soon)
- Some words appear in test but not in training set (out of vocab words)
  - Use broader vocab + smoothing
  - Make these words as <UNK> and treat them as any other word.
  - Remove these words from test
- **Problem with** n-gram: zero probability
  - Apply discounting

Laplace Smoothing for classification

If a test case has an attribute that does not occur in every class? Then the probability value for that attribute is 0, so all is 0!

In this case use Laplace smoothing:

$$P(x|y) = \frac{n_{yx}}{n_y} \Longrightarrow \hat{P}(x|y) = \frac{\gamma + n_{yx}}{\gamma \cdot d + n_y}, \; x \in (x_1, \ldots, x_d)$$ d-dimensional multinomial distribution

n_y no. of instances from training in class y

n_{yx} no. of instances from class y with value x for attribute X

d no. of distinct features for that attribute (number of dimensions),

γ is usually taken as 1.

Every feature must be represented in each class with non-zero probability.

BNB example with Laplace smoothing

	a	b	c	class
e_1	0	1	0	+
e_2	0	1	1	+
e_3	1	0	0	+
e_4	1	1	0	+
v_1	1	1	1	+
v_2	0	0	0	+
e_5	1	1	0	−
e_6	1	0	1	−
e_7	1	0	0	−
e_8	0	0	0	−
v_1	1	1	1	−
v_2	0	0	0	−

Smoothing added 2 rows in each class, need to add row with 0's, because each feature must have prob>0 in each class.

And this is binomial, so 2 dimensions

THE UNIVERSITY of ADELAIDE

BNB example with Laplace smoothing

	a	b	c	class
e_1	0	1	0	+
e_2	0	1	1	+
e_3	1	0	0	+
e_4	1	1	0	+
v_1	1	1	1	+
v_2	0	0	0	+
e_5	1	1	0	−
e_6	1	0	1	−
e_7	1	0	0	−
e_8	0	0	0	−
v_1	1	1	1	−
v_2	0	0	0	−

- BNB training:
  - For each class: num docs with word/num all docs
  - Spam: P(a|+), P(b|+), P(c|+) = (3/6, 4/6, 2/6) = (0.5; 0.67; 0.33)
  - Ham: P(a|-), P(b|-), P(c|-) = (4/6, 2/6, 2/6) = (0.67; 0.33; 0.33)

  Testing for instance (a b -) = (1,1,0):

  Classify (a b -) using MAP:

  - P(c|+) = ?
  - P(c|-) = ?
  - Result:

THE UNIVERSITY of ADELAIDE

BNB example with Laplace smoothing

	a	b	c	class
e_1	0	1	0	+
e_2	0	1	1	+
e_3	1	0	0	+
e_4	1	1	0	+
v_1	1	1	1	+
v_2	0	0	0	+
e_5	1	1	0	−
e_6	1	0	1	−
e_7	1	0	0	−
e_8	0	0	0	−
v_1	1	1	1	−
v_2	0	0	0	−

- BNB training:
  - For each class: num docs with word/num all docs
  - Spam: P(a|+), P(b|+), P(c|+) = (3/6, 4/6, 2/6) = (0.5; 0.67; 0.33)
  - Ham: P(a|-), P(b|-), P(c|-) = (4/6, 2/6, 2/6) = (0.67; 0.33; 0.33)

Testing for instance x= (a b -) = (1,1,0):

Classify x=(a b -) using MAP:

- P(x|+) = 0.5 *0.67 *(1-0.33) = 0.224;
- P(x|-) = 0.67 *0.33*(1-0.33) = 0.148
- Result: SPAM (same as before)

Classify using ML: same, since P(-) = P(+) = 0.5

Laplace Smoothing for n-grams

Probabilities of some n-grams are 0.

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \Longrightarrow \hat{P}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+|V|}$$

C function is count of words, *V* is the vocabulary

- Another version of Laplace: k-smoothing, k is optimised on devset

$$\hat{P}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + k \times |V|}$$

- But Laplace smoothing does not work well for n-grams

More on Smoothing

- **Problem with** n-gram: zero probability
- **Back-off smoothing**: if n-gram has zero probability, back-off to n-1 gram, until reaching non-zero probability. But this method has a problem: mixing probabilities for lower and higher order n-grams may sum to greater than 1. We need to discount higher order probabilities to save for lower order.
- Undiscounted back-off smoothing can be used for very large language models, called **stupid back-off (☺)**
- **Interpolation smoothing**: instead of simple back-off, combine n-1, n-2, …, smoothing by weighted averaging probabilities.
- **Kneser-Ney** method addresses this problem and combines both.

Kneser-Ney Smoothing elements

- Kneser-Ney is based on P continuation P_{cont}
  - Discount is based on the number of different contexts word ***w*** has appeared after other words
  - E.g.. AA, **AB**, AC, AD, AC, AB, **CB**, **DB**, BA, CB:

    B appears in 3 different contexts out of 7 contexts, therefore, $P_{cont} = \frac{3}{7}$
  - Intuition words that have appeared in more contexts in the past are more likely to appear in some new context
  - General formula $P_{cont}(w_i) = \frac{|\{w' : 0 < c(w', w_i)\}|}{|\{(w', w'') : 0 < c(w', w'')\}|}$
- Discount $d, 0 \leq d \leq 1$
- Constant λ that distributes probability between 1-gram and 2-gram (for 2-grams)

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}\, v)} |\{w: C(w_{i-1}w) > 0\}|$$

THE UNIVERSITY of ADELAIDE

Advanced: Kneser-Ney Smoothing

- The final probability function using KN smoothing formula is recursive, it calculates probability recursively for sequences until it gets to unigram.
- Formula for bigram is

$$P_{KN}(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{KN}(w_i)$$

Where $\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}\, v)} |\{w: C(w_{i-1}w) > 0\}|$

$P_{KN}(w_i) = P_{cont}(w_i)$ for bi-gram

- General formula for n-gram in Jurafsky ch 3.7.2

Advanced: KN Smoothing full formula

- General formula for n-gram (Kneser–Ney smoothing – Wikipedia)

$$p_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(c(w_{i-n+1}^{i-1}, w_i) - \delta, 0)}{\sum_{w'} c(w_{i-n+1}^{i-1}, w')} + \delta \frac{|\{w' : 0 < c(w_{i-n+1}^{i-1}, w')\}|}{\sum_{w'} c(w_{i-n+1}^{i-1}, w')} p_{KN}(w_i|w_{i-n+2}^{i-1})$$

δ in this formula is our d, which is a discount constant, usually 0.75.

w_{i-n+1}^{i-1} is a sequence of words in n-gram from the one before last token in n-gram w_i back to the first token in n-gram

For example, for $n = 4$, we have $w_{i-3}, w_{i-2}, w_{i-1}$ before token w_i

- For bi-gram P_{KN} is reduced to just $P_{KN}(w_i) = P_{cont}(w_i) = \frac{|\{w' : 0 < c(w', w_i)\}|}{|\{(w', w'') : 0 < c(w', w'')\}|}$
- The full formula is recursive and goes from n-gram to one-gram.

THE UNIVERSITY of ADELAIDE

Multinomial Naïve Bayes

Multinomial Naïve Bayes (MNB)

- What information was not used in NB method so far?
- So far we treated occurrence of a word as binary: it is there or it is not. This is called Bernoulli Naïve Bayes method (BNB), because it follows Bernoulli distribution

$$p(x) = P[X = x] = \begin{cases} q = 1 - p & x = 0 \\ p & x = 1 \end{cases}$$

$$f(k; p) = p^k (1 - p)^{1-k} \quad \text{for } k \in \{0, 1\}$$

k is number of outcomes

- But if we count how many times a word (token) appears in an instance, this is called Multinomial Naïve Bayes method (MNB)
- More complex calculations, but generally performs better

MNB example with Laplace smooting

	a	b	c	class
e_1	0	3	0	+
e_2	0	3	3	+
e_3	3	0	0	+
e_4	2	3	0	+
v_1	1	0	0	+
v_2	0	1	0	+
v_3	0	0	1	+
e_5	4	3	0	−
e_6	4	0	3	−
e_7	3	0	0	−
e_8	0	0	0	−
v_1	1	0	0	−
v_2	0	1	0	−
v_3	0	0	1	−

Smoothing added 3 features, 1 for each class, because this is multinomial 3-dim multinomial distribution

MNB example with Laplace smooting

	a	b	c	class
e_1	0	3	0	+
e_2	0	3	3	+
e_3	3	0	0	+
e_4	2	3	0	+
v_1	1	0	0	+
v_2	0	1	0	+
v_3	0	0	1	+
e_5	4	3	0	−
e_6	4	0	3	−
e_7	3	0	0	−
e_8	0	0	0	−
v_1	1	0	0	−
v_2	0	1	0	−
v_3	0	0	1	−

- MNB training:
  - For each class: word count/all words count
  - Spam: P(a|+), P(b|+), P(c|+) = (6/20, 10/20, 4/20) = (0.3, 0.5, 0.2)
  - Ham: P(a|-), P(b|-), P(c|-) = (12/20, 4/20, 4/20) = (0.6, 0.2, 0.2)

MNB calculations: example

Testing instance: (*a a a b*) = (3a 1b 0c) = (3,1,0):

- Multiple occurrences of a word in doc
- We model this as:

There are k different unique words in a bag of words.

Probability of drawing k^{th} word is p_k.

We draw n of those words from the bag with replacement.

What is the probability that we get x_1 of k_1 word AND x_2 of k_2 AND …, etc.? (sum(x_i) = n). It follows multinomial probability mass function (general formula):

$$f(x_1, \ldots, x_k; n, p_1, \ldots, p_k) = \Pr(X_1 = x_1 \text{ and } \ldots \text{ and } X_k = x_k)$$
$$= \begin{cases} \dfrac{n!}{x_1! \cdots x_k!} p_1^{x_1} \times \cdots \times p_k^{x_k}, & \text{when } \sum_{i=1}^{k} x_i = n \\ 0 & \text{otherwise,} \end{cases}$$

https://en.wikipedia.org/wiki/Multinomial_distribution

MNB calculations: example

Our instance: (#a=3,#b=1,#c=0):

n=4,

k = 3 |(a,b,c)

x1=3, x2=1, x3=0,

$$likelihood = \boldsymbol{argmax}(P(X|h))$$

Therefore:

P(x|+) = ?

P(x|-) =?

MNB calculations: example

Our instance: (#a=3,#b=1,#c=0):

n=4, x1=3, x2=1, x3=0,

Therefore:

$P(x|+) = 4!/(3!*1!*0!)*0.3^3*0.5^1*0.2^0 = 0.054$ for SPAM

$P(x|-) = 4!/(3!*1!*0!)*0.6^3*0.2^1*0.2^0 = 0.1728$ = for HAM

So by arg max likelihood this is ham

BNB: same instance (a a a b -)=(a b -) = (1,1,0): spam,

different than MNB, because of 3 a's.

Sentiment analysis and classfication

Sentiment, aspect, stance

“@united currently on board so not now. Check in was terrible. Staff rude. Expensive luggage thrown around etc.”

- Identify aspects in this tweet

How to use sentiment lexicons

- Lexicons are kind of dictionaries of words labelled with emotion or sentiment
- E.g. NRC: words labelled with emotions, e.g.
  abandon fear 1
  abandon joy 0
  - Score words by summing positive and negative emotion labels
- VADER (Valence Aware Dictionary and sEntiment Reasoner)
  - Lexicon and Tool Set
  - Validated by multiple independent human judges,
  - Suitable for tweets and longer texts.
  - Hard to use on its own, better to use NLTK API or the tools.
  - Additional information and the full VADER dataset and tools are in MyUni Online Learning.

How to classify sentiment

- Use lexicons, e.g. VADER on its own (unsupervised) or combined with a classifier
- Once you have polarity of the word (=sentiment), add a feature to the text, e.g. “__NEG__” and use it in training.
- (Jurafsky&Martin, ch. 4.4) prepend the prefix NOT to every word after a token of logical negation (n’t, not, no, never) until the next punctuation mark. Thus the phrase
  “didn't like this movie , but I”
  becomes
  “didn't NOT_like NOT_this NOT_movie , but I”
- Can parse sentence into phrases (will discuss in future modules)
- (better to test this method on devset before applying)

THE UNIVERSITY
of ADELAIDE

Exercise on Bayesian rule: solution

P(cancer) = 0.008, P(not cancer) = 0.992, these are called prior probabilities

P(+ | cancer) = 0.98 P(- | cancer) = 0.02

P(+ | not cancer) = 0.03 P(- | not cancer) = 0.97

Does the patient have cancer or not? To answer this, calculate P1 = P(not cancer | +) and P2 = P(cancer | +), and then calculate likelihood ratio P1/P2.

P1 = P(nc | +) = P(+|nc)*P(nc) = 0.03*0.992

P2 = P(c|+) = P(+|c)*P(c) = 0.98*0.008

Answer: P1/P2 = 3.8