Foundations of Natural Language Processing Lecture 7 Text Classification

Shay Cohen

(Based on slides from Henry Thompson, Alex Lascarides and Sharon Goldwater)

4 February 2020

Text classification: example

Dear Prof. [ZZ]:

My name is [XX]. I am an ambitious applicant for the Ph.D program of Electrical Engineering and Computer Science at your university. Especially being greatly attracted by your research projects and admiring for your achievements via the school website, I cannot wait to write a letter to express my aspiration to undertake the Ph.D program under your supervision.

I have completed the M.S. program in Information and Communication Engineering with a high GPA of 3.95/4.0 at [YY] University. In addition to throwing myself into the specialized courses in [...] I took part in the research projects, such as [...]. I really enjoyed taking the challenges in the process of the researches and tests, and I spent two years on the research project [...]. We proved the effectiveness of the new method for [...] and published the result in [...].

Having read your biography, I found my academic background and research experiences indicated some possibility of my qualification to join your team. It is my conviction that the enlightening instruction, cutting-edge research projects and state of-the-art facilities offered by your team will direct me to make breakthroughs in my career development in the arena of electrical engineering and computer science. Thus, I shall be deeply grateful if you could give me the opportunity to become your student. Please do not hesitate to contact me, should you need any further information about my scholastic and research experiences.

Yours sincerely, [XX].

Text classification

We might want to categorize the *content* of the text:

- Spam detection (binary classification: spam/not spam)
- Sentiment analysis (binary or multiway)
  - movie, restaurant, product reviews (pos/neg, or 1-5 stars)
  - political argument (pro/con, or pro/con/neutral)
- Topic classification (multiway: sport/finance/travel/etc)

Text classification

Or we might want to categorize the *author* of the text (**authorship attribution**):

- Native language identification (e.g., to tailor language tutoring)
- Diagnosis of disease (psychiatric or cognitive impairments)
- Identification of gender, dialect, educational background, political orientation (e.g., in forensics [legal matters], advertising/marketing, campaigning, disinformation).

N-gram models for classification?

N-gram models can sometimes be used for classification (see Lab 2: identifying specific authors). But

- For many tasks, sequential relationships between words are largely irrelevant: we can just consider the document as a **bag of words**.
- On the other hand, we may want to include other kinds of features (e.g., part of speech tags) that N-gram models don't include.

Here we consider two alternative models for classification:

- **Naive Bayes** (this should be review)
- **Maximum Entropy** (aka **multinomial logistic regression**).

Bag of words

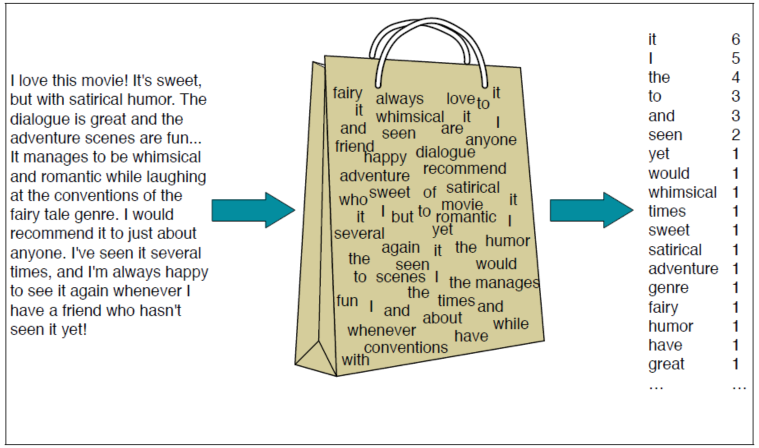

Figure 7.1 Intuition of the multinomial naive Bayes classifier applied to a movie review. The position of the words is ignored (the *bag of words* assumption) and we make use of the frequency of each word.

Figure from J&M 3rd ed. draft, sec 7.1

Naive Bayes: high-level formulation

- Given document d and set of categories C (say, spam/not-spam), we want to assign d to the most probable category $\hat{c}$.

$$\hat{c} = \operatorname*{argmax}_{c \in C} P(c|d)$$

Naive Bayes: high-level formulation

- Given document d and set of categories C (say, spam/not-spam), we want to assign d to the most probable category $\hat{c}$.

$$\begin{aligned}\hat{c} &= \operatorname*{argmax}_{c \in C} P(c|d) \\ &= \operatorname*{argmax}_{c \in C} \frac{P(d|c)P(c)}{P(d)} \\ &= \operatorname*{argmax}_{c \in C} P(d|c)P(c)\end{aligned}$$

- Just as in spelling correction, we need to define $P(d|c)$ and $P(c)$.

How to model $P(d|c)$?

- First, define a set of **features** that might help classify docs.
  - Here we'll assume these are all the words in the vocabulary.
  - But, we could just use **some** words (more on this later...).
  - Or, use other info, like parts of speech, if available.
- We then represent each document d as the set of features (words) it contains: $f_1, f_2, \dots f_n$. So

$$P(d|c) = P(f_1, f_2, \dots f_n|c)$$

Naive Bayes assumption

- As in LMs, we can't accurately estimate $P(f_1, f_2, \dots f_n|c)$ due to sparse data.
- So, make a **naive Bayes assumption**: features are conditionally independent given the class.

$$P(f_1, f_2, \dots f_n|c) \approx P(f_1|c)P(f_2|c) \dots P(f_n|c)$$

- That is, the prob. of a word occurring depends **only** on the class.
  - Not on which words occurred before or after (as in N-grams)
  - Or even which other words occurred at all

Naive Bayes assumption

- Effectively, we only care about the **count** of each feature in each document.
- For example, in spam detection:

	the	your	model	cash	Viagra	class	account	orderz
doc 1	12	3	1	0	0	2	0	0
doc 2	10	4	0	4	0	0	2	0
doc 3	25	4	0	0	0	1	1	0
doc 4	14	2	0	1	3	0	1	1
doc 5	17	5	0	2	0	0	1	1

Naive Bayes classifier

Putting together the pieces, our complete classifier definition:

- Given a document with features $f_1, f_2, \dots f_n$ and set of categories C, choose the class $\hat{c}$ where

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

  - $P(c)$ is the **prior probability** of class c before observing any data.
  - $P(f_i|c)$ is the probability of seeing feature f_i in class c.

Estimating the class priors

- $P(c)$ normally estimated with MLE:

$$\hat{P}(c) = \frac{N_c}{N}$$

  - N_c = the number of training documents in class c
  - N = the total number of training documents

- So, $\hat{P}(c)$ is simply the proportion of training documents belonging to class c.

Learning the class priors: example

- Given training documents with correct labels:

	the	your	model	cash	Viagra	class	account	orderz	spam?
doc 1	12	3	1	0	0	2	0	0	-
doc 2	10	4	0	4	0	0	2	0	+
doc 3	25	4	0	0	0	1	1	0	-
doc 4	14	2	0	1	3	0	1	1	+
doc 5	17	5	0	2	0	0	1	1	+

- $\hat{P}(\text{spam}) = 3/5$

Learning the feature probabilities

- $P(f_i|c)$ normally estimated with simple smoothing:

$$\hat{P}(f_i|c) = \frac{\text{count}(f_i, c) + \alpha}{\sum_{f \in F}(\text{count}(f, c) + \alpha)}$$

  - $\text{count}(f_i, c)$ = the number of times f_i occurs in class c
  - F = the set of possible features
  - α: the smoothing parameter, optimized on held-out data

Learning the feature probabilities: example

	the	your	model	cash	Viagra	class	account	orderz	spam?
doc 1	12	3	1	0	0	2	0	0	-
doc 2	10	4	0	4	0	0	2	0	+
doc 3	25	4	0	0	0	1	1	0	-
doc 4	14	2	0	1	3	0	1	1	+
doc 5	17	5	0	2	0	0	1	1	+

Learning the feature probabilities: example

	the	your	model	cash	Viagra	class	account	orderz	spam?
doc 1	12	3	1	0	0	2	0	0	-
doc 2	10	4	0	4	0	0	2	0	+
doc 3	25	4	0	0	0	1	1	0	-
doc 4	14	2	0	1	3	0	1	1	+
doc 5	17	5	0	2	0	0	1	1	+

$\hat{P}(\text{your}|+) = \frac{(4+2+5+\alpha)}{(\text{all words in + class})+\alpha|F|} = (11+\alpha)/(68+\alpha|F|)$

Learning the feature probabilities: example

	the	your	model	cash	Viagra	class	account	orderz	spam?
doc 1	12	3	1	0	0	2	0	0	-
doc 2	10	4	0	4	0	0	2	0	+
doc 3	25	4	0	0	0	1	1	0	-
doc 4	14	2	0	1	3	0	1	1	+
doc 5	17	5	0	2	0	0	1	1	+

$\hat{P}(\text{your}|+) = \frac{(4+2+5+\alpha)}{(\text{all words in + class})+\alpha|F|} = (11+\alpha)/(68+\alpha|F|)$

$\hat{P}(\text{your}|-) = \frac{(3+4+\alpha)}{(\text{all words in} - \text{class})+\alpha|F|} = (7+\alpha)/(49+\alpha|F|)$

$\hat{P}(\text{orderz}|+) = \frac{(2+\alpha)}{(\text{all words in + class})+\alpha|F|} = (2+\alpha)/(68+\alpha|F|)$

Classifying a test document: example

- Test document d:

  get your cash and your orderz

- Suppose all features not shown earlier have $\hat{P}(f_i|+) = \frac{\alpha}{(68+\alpha|F|)}$

$$
\begin{aligned}
P(+|d) &\propto P(+)\prod_{i=1}^{n} P(f_i|+) \\
&= P(+)\cdot\frac{\alpha}{(68+\alpha|F|)}\cdot\frac{11+\alpha}{(68+\alpha|F|)}\cdot\frac{7+\alpha}{(68+\alpha|F|)} \\
&\quad \cdot\frac{\alpha}{(68+\alpha|F|)}\cdot\frac{11+\alpha}{(68+\alpha|F|)}\cdot\frac{2+\alpha}{(68+\alpha|F|)}
\end{aligned}
$$

Classifying a test document: example

- Test document d:

  get your cash and your orderz

- Do the same for $P(-|d)$
- Choose the one with the larger value

Alternative feature values and feature sets

- Use only **binary** values for f_i: did this word occur in d or not?
- Use only a subset of the vocabulary for F
  - Ignore **stopwords** (function words and others with little content)
  - Choose a small task-relevant set (e.g., using a sentiment lexicon)
- Use more complex features (bigrams, syntactic features, morphological features, ...)

Very small numbers...

Multiplying large numbers of small probabilities together is problematic in practice

- Even in our toy example $P(-|class)P(-|account)P(-|Viagra)$ with $\alpha = 0.01$ is 5×10^{-5}
- So it would only take two dozen similar words to get down to 10^{-44}, which cannot be represented with as single-precision floating point number
- Even double precision fails once we get to around 175 words with total probability around 10^{-310}

So most actual implementations of Naive Bayes use **costs**

- Costs are negative log probabilities
- So we can sum them, thereby avoiding underflow
- And look for the *lowest* cost overall

Costs and linearity

Using costs, our Naive Bayes equation looks like this

$$\hat{c} \;=\; \operatorname*{argmin}_{c \in C} + (-logP(c) + \sum_{i=1}^{n} -logP(f_i|c))$$

We're finding the *lowest* cost classification.

This amounts to classification using a linear function (in log space) of the input features.

- So Naive Bayes is called a **linear classifier**
- As is Logistic Regression (to come)

Task-specific features

Example words from a **sentiment lexicon**:

Positive:

absolutely	beaming	calm
adorable	beautiful	celebrated
accepted	believe	certain
acclaimed	beneficial	champ
accomplish	bliss	champion
achieve	bountiful	charming
action	bounty	cheery
active	brave	choice
admire	bravo	classic
adventure	brilliant	classical
affirm	bubbly	clean
...		...

Negative:

abysmal	bad	callous
adverse	banal	can't
alarming	barbed	clumsy
angry	belligerent	coarse
annoy	bemoan	cold
anxious	beneath	collapse
apathy	boring	confused
appalling	broken	contradictory
atrocious		contrary
awful		corrosive
		corrupt
		...

From `http://www.enchantedlearning.com/wordlist/`

Choosing features can be tricky

- For example, sentiment analysis might need domain-specific non-sentiment words
  - Such as quiet, memory for computer product reviews.
- And for other tasks, stopwords might be very useful features
  - E.g., People with schizophrenia use more 2nd-person pronouns (Watson et al, 2012), those with depression use more 1st-person (Rude, 2004).
- Probably better to use too many irrelevant features than not enough relevant ones.

Advantages of Naive Bayes

- Very easy to implement
- Very fast to train, and to classify new documents (good for huge datasets).
- Doesn't require as much training data as some other methods (good for small datasets).
- Usually works reasonably well
- This should be your baseline method for any classification task

Problems with Naive Bayes

- Naive Bayes assumption is naive!
- Consider categories TRAVEL, FINANCE, SPORT.
- Are the following features independent given the category?

beach, sun, ski, snow, pitch, palm, football, relax, ocean

Problems with Naive Bayes

- Naive Bayes assumption is naive!
- Consider categories TRAVEL, FINANCE, SPORT.
- Are the following features independent given the category?

beach, sun, ski, snow, pitch, palm, football, relax, ocean

  - No! Ex: Given TRAVEL, seeing beach makes sun more likely, but ski less likely.
  - Defining finer-grained categories might help (beach travel vs ski travel), but we don't usually want to.

Non-independent features

- Features are not usually independent given the class
- Adding multiple feature types (e.g., words and morphemes) often leads to even stronger correlations between features
- Accuracy of classifier can sometimes still be ok, but it will be highly **overconfident** in its decisions.
  - Ex: NB sees 5 features that all point to class 1, treats them as five independent sources of evidence.
  - Like asking 5 friends for an opinion when some got theirs from each other.

A less naive approach

- Although Naive Bayes is a good starting point, often we have enough training data for a better model (and not so much that slower performance is a problem).
- We may be able to get better performance using loads of features and a model that doesn't assume features are conditionally independent.
- Namely, a **Maximum Entropy model**.

MaxEnt classifiers

- Used widely in many different fields, under many different names
- Most commonly, **multinomial logistic regression**
  - *multinomial* if more than two possible classes
  - otherwise (or if lazy) just *logistic regression*
- Also called: log-linear model, one-layer neural network, single neuron classifier, etc ...
- The mathematical formulation here (and in the text) looks slightly different from standard presentations of multinomial logistic regresssion, but is ultimately equivalent.

Naive Bayes vs MaxEnt

- Like Naive Bayes, MaxEnt assigns a document d to class $\hat{c}$, where
$$\hat{c} = \operatorname*{argmax}_{c \in C} P(c|d)$$
- Unlike Naive Bayes, we do not apply Bayes' Rule. Instead, we model $P(c|d)$ directly.

Example: classify by topic

- Given a web page document, which topic does it belong to?
  - $\vec{x}$ are the words in the document, plus info about headers and links.
  - c is the latent class. Assume three possibilities:

$c =$	class
1	TRAVEL
2	SPORT
3	FINANCE

Feature functions

- Like Naive Bayes, MaxEnt models use **features** we think will be useful for classification.

- However, features are treated differently in the two models:
  - NB: features are **directly observed** (e.g., words in doc): no difference between features and data.
  - MaxEnt: we will use $\vec{x}$ to represent the observed data. Features are **functions** that depend on both observations $\vec{x}$ and class c.

This way of treating features in MaxEnt is standard in NLP; in ML it's often explained differently.

MaxEnt feature example

- If we have three classes, our features will always come in groups of three. For example, we could have three binary features:

$$\begin{array}{ll} f_1: & \texttt{contains('ski')}\ \&\ c = 1 \\ f_2: & \texttt{contains('ski')}\ \&\ c = 2 \\ f_3: & \texttt{contains('ski')}\ \&\ c = 3 \end{array}$$

  - training docs from class 1 that contain ski will have f_1 active;
  - training docs from class 2 that contain ski will have f_2 active;
  - etc.

- Each feature f_i has a real-valued **weight** w_i (learned in training).

Classification with MaxEnt

Choose the class that has highest probability according to

$$P(c|\vec{x}) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(\vec{x}, c)\right)$$

where the normalization constant $Z = \sum_{c'} \exp(\sum_i w_i f_i(\vec{x}, c'))$

- Inside brackets is just a dot product: $\vec{w} \cdot \vec{f}$.

- And $P(c|\vec{x})$ is a **monotonic function** of this dot product.

- So, we will end up choosing the class for which $\vec{w} \cdot \vec{f}$ is highest.

Classification example

f_1 :	`contains('ski')` $\&\ c = 1$	$w_1 = 1.2$
f_2 :	`contains('ski')` $\&\ c = 2$	$w_2 = 2.3$
f_3 :	`contains('ski')` $\&\ c = 3$	$w_3 = -0.5$
f_4 :	`link_to('expedia.com')` $\&\ c = 1$	$w_4 = 4.6$
f_5 :	`link_to('expedia.com')` $\&\ c = 2$	$w_5 = -0.2$
f_6 :	`link_to('expedia.com')` $\&\ c = 3$	$w_6 = 0.5$
f_7 :	`num_links` $\&\ c = 1$	$w_7 = 0.0$
f_8 :	`num_links` $\&\ c = 2$	$w_8 = 0.2$
f_9 :	`num_links` $\&\ c = 3$	$w_9 = -0.1$

- f_7, f_8, f_9 are **numeric** features that count outgoing links.

Classification example

- Suppose our test document contains ski and 6 outgoing links.
- We don't know c for this doc, so we try out each possible value.
  - Travel: $\sum_i w_i f_i(\vec{x}, c = 1) = 1.2 + (0.0)(6) = 1.2$.
  - Sport: $\sum_i w_i f_i(\vec{x}, c = 2) = 2.3 + (0.2)(6) = 3.5$.
  - Finance: $\sum_i w_i f_i(\vec{x}, c = 3) = -0.5 + (-0.1)(6) = -1.1$.
- We'd need to do further work to compute the probability of each class, but we know already that SPORT will be the most probable.

Feature templates

- In practice, features are usually defined using **templates**

  `contains`(w) $\&$ c
  `header_contains`(w) $\&$ c
  `header_contains`(w) `& link_in_header` $\&$ c

  - instantiate with all possible words w and classes c
  - usually filter out features occurring very few times
- NLP tasks often have a few templates, but 1000s or 10000s of features

Training the model

- Given annotated data, choose weights that make the labels most probable under the model.
- That is, given items $x^{(1)} \dots x^{(N)}$ with labels $c^{(1)} \dots c^{(N)}$, choose

$$\hat{w} = \underset{\vec{w}}{\operatorname{argmax}} \sum_j \log P(c^{(j)} | x^{(j)})$$

Training the model

- Given annotated data, choose weights that make the class labels most probable under the model.
- That is, given examples $x^{(1)} \dots x^{(N)}$ with labels $c^{(1)} \dots c^{(N)}$, choose

$$\hat{w} = \underset{\vec{w}}{\operatorname{argmax}} \sum_j \log P(c^{(j)}|x^{(j)})$$

- called **conditional maximum likelihood estimation** (CMLE)
- Like MLE, CMLE will overfit, so we use tricks (**regularization**) to avoid that.

The downside to MaxEnt models

- Supervised MLE in generative models is easy: compute counts and normalize.
- Supervised CMLE in MaxEnt model not so easy
  - requires multiple iterations over the data to gradually improve weights (using gradient ascent).
  - each iteration computes $P(c^{(j)}|x^{(j)})$ for all j, and each possible $c^{(j)}$.
  - this can be time-consuming, especially if there are a large number of classes and/or thousands of features to extract from each training example.

Summary

- Two methods for text classification: Naive Bayes, MaxEnt
- Make different independence assumptions, have different training requirements.
- Both are easily available in standard ML toolkits.
- Both require some work to figure out what features are good to use.