NLP

- Natural Language Processing (**NLP**) is an active and attractive field
- Most of our activities online are text-based
- Most of the data available today is text: e-mails, blogs, news, search results, reviews, social media, medical reports, course content, etc.
- Leverage the large and valuable amounts of text available (estimated in hundreds of thousands of perabytes)
- Why NLP? Communicating with computers using natural language has always been a dream...

1. **Ambiguity:**
   "At last, a computer that understands you like your mother."
   1985 McDonnell-Douglas ad.
2. **Anaphora:** He bought a brand new car and drove it home.
3. **Metonymy:** She learned how to play Mozart at a very young age.
4. **Metaphor:** He is a walking dictionary! His room is a zoo.
5. **Vagueness, discourse structure, auto correction, etc.**

Text Classification

Learning to classify text. Why?

- Learn which news articles are of interest
- Learn to classify web pages by topic
- Classify Spam from non Spam emails
- Naive Bayes is among most effective algorithms
- What attributes shall we use to represent text documents?

Setting

- A training data (x_i, y_i), x_i is a feature vector and y_i is a discrete label. d features, and n examples.
- Example: consider document classification.
- A new example with feature values $x_{new} = (a_1, a_2, ..., a_d)$.
- We want to predict the label y_{new} of the new example.

$$y_{new} = \arg\max_{y \in \mathbb{Y}} \; p(y|a_1, a_2, \cdots, a_d)$$

Use simplifying assumption:

$$p(a_1, a_2, \cdots, a_d|y) = \prod_j p(a_j|y)$$

Naive Bayes Classifier:

$$y_{new} = \arg\max_{y \in \mathbb{Y}} \; p(y) \prod_j p(a_j|y)$$

Algorithm

Learning: Based on the frequency counts in the dataset:

1. Estimate all $p(y)$, $\forall y \in Y$.
2. Estimate all $p(a_j|y)$, $\forall y \in Y$, $\forall a_j$

Classification: For a new example, use:

$$y_{new} = \text{argmax}_{y \in \mathbb{Y}} p(y) \prod_j p(a_j|y)$$

Note: No model per se or hyperplane, just count the frequencies of various data combinations within the training examples.

Estimating probabilities

m-estimate of the probability: $p(a_j|y) = \frac{n_c + m * p}{n_y + m}$

where:

n_y: total number of examples for which the class is y.

n_c: total number of examples for which the class is y and feature $x_j = a_j$

m: called equivalent sample size

Intuition:

Augment the sample size by m virtual examples, distributed according to prior p (prior estimate of each value).

If prior is unknown, assume uniform prior: if a feature has k values, we can set p=1/k.

Text Classification

- Given a document (corpus), define an attribute for each word position in the document.
- The value of the attribute is the English word in that position.
- To reduce the number of probabilities that needs to be estimated, besides NB independence assumption, we assume that: The probability of a given word w_k occurrence is independent of the word position within the text. That is:

$$p(x_1 = w_k|c_j), p(x_2 = w_k|c_j), \cdots$$

estimated by:

$$p(w_k|c_j)$$

- m-estimate of the probabilities:

$$p(w_k|c_j) = \frac{n_k + 1}{n_j + |Vocabulary|}$$

where:

n_j: total #word positions in all training examples of class c_j.

n_k: number of times the word w_k is found in among these n_j word positions.

- The following function/algorithms learns the probabilities $P(w_k|c_j)$ describing the probability that a randomly drawn word from a document with class c_j is the English word w_k. It also learn the class priors $P(c_j)$.

Learn_Naive_Bayes_texte(Examples, C)

Input: Examples is a set of document with classes. C is the set of classes.

1. Collect all words, punctuations and tokens occuring in the Examples. Let the pertinent vocabulary be V.
2. Calculate $P(c_j)$ and $P(w_k/c_j)$.
   - For each class c_j in C
     - $docs_j \leftarrow$ the subset of documents from Examples for which the label=c_j
     - $P(c_j) \leftarrow \frac{|docs_j|}{|Examples|}$
     - $text_j \leftarrow$ a single document concatenation of all documents in doc_j
     - $n_j \leftarrow$ total number of distinct word positions in $text_j$
     - for each word w_k in V
       - $n_k \leftarrow$ number of times word w_k appears in $text_j$
       - $P(w_k/c_j) \leftarrow \frac{n_k+1}{n_j+|V|}$

Output: all $P(c_j)$ and $P(w_k/c_j)$.

Classify_Naive_Bayes_text(Doc)

Return the estimated label for the document Doc. a_i denotes the word found in the i^{th} position within Doc.

- positions $\leftarrow$ all word positions in Doc that contain token found in V.
- Return c_{Doc} where:

$$c_{Doc} = \arg\max_{c_j \in C} P(c_j) \prod_{i \in positions} P(a_i/c_j)$$

Example

Classification of Radio and TV sentences.

TV:

1. TV programs are not interesting – TV is annoying.
2. Kids like TV.
3. We receive TV by radio waves.

Radio:

1. It is interesting to listen to the radio.
2. On the waves, kids programs are rare.
3. The kids listen to the radio; it is rare!

Vocabulary:

V = {TV, program, interesting, kids, radio, wave, listen, rare}

$p(C_{Tv}) = 3/6 = 0.5 \quad p(C_{Radio}) = 3/6 = 0.5$

$n_{TV} = 9 \quad n_{Radio} = 11$

$w \in \mathcal{V}$	Class "TV"			Class "Radio"		
	n_{TV}	n_w	$p(w\|C_{TV})$	n_{Radio}	n_w	$p(w\|C_{radio})$
TV	9	4	(4+1)/(9+8)	11	0	1/(11+8)
program	9	1	(1+1)/(9+8)	11	1	2/(11+8)
interesting	9	1	(1+1)/(9+8)	11	1	2/(11+8)
kids	9	1	(1+1)/(9+8)	11	2	3/(11+8)
radio	9	1	(1+1)/(9+8)	11	2	3/(11+8)
wave	9	1	(1+1)/(9+8)	11	1	2/(11+8)
listen	9	0	(0+1)/(9+8)	11	2	3/(11+8)
rare	9	0	(0+1)/(9+8)	11	2	3/(11+8)

Language Models

- We just saw that language is complex, there is no single meaning, we disagree on the grammar and there is not set of definitive sentences
- Instead of talking of one single meaning of a sentence, we talk of **probability distribution over meaning**
- A language model is an approximation of language
- Aim: Model natural language

- **Build a probabilistic language model that assigns a:**
  - probability to each next possible word: **predict** the next word

    P(*mother*|*Did you call your*...)

    P(*dinosaur*|*Did you call your*...)

    P(*doctor*|*Did you call your*...)
  - probability to a complete sentence (sequence of words): **predict** the probability to see this sentence in a text

    P(*Open your book on page six*)

    P(*book open ten your on page*)

Language models are crucial in many NLP applications:

- **Spell correction**

  "Once upon a time" versus "Ounce upon a time"
- **Statistical machine translation**

  "Out of sight, out of mind" translation to either (1) "Invisible, imbecile" or (2) "Hors de vue, hors de l' esprit".
- **Seek information** (text classification, information retrieval, information extraction).
- **Speech recognition**
- **Language identification**

N-gram models

- Estimate *P*(page|open your book on) using frequencies in a large corpus:

$$P(\texttt{page}|\texttt{open your book on}) = \frac{\texttt{count(open your book on page)}}{\texttt{count(open your book on)}}$$

- Estimate *P*(open your book on page) using frequencies in a large corpus:

$$P(\texttt{open your book on page}) = \frac{\texttt{count(open your book on page)}}{\texttt{count(sentences of 5 words)}}$$

- The corpus has to be very very large!
- Poor model. Will be zero for a sentence that does not appear in the corpus.

N-gram models

- **Problem:** How to estimate the joint probability?

$$P(w_1, w_2, , \cdots, w_n)$$

- **Solution:** decompose the joint probability using **chain rule of probability**

$$P(w_1, \cdots, w_n) = p(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\cdots P(w_n|w_1 \cdots w_{n-1})$$

$$P(w_1, \cdots, w_n) = \prod_{k=1}^{n} P(w_k|w_1 \cdots w_{k-1})$$

N-gram models

- **Idea:** Instead of using the whole chain, approximate using the last words.
- Bigram model: uses the **Markov assumption**

$$P(w_n|w_{n-1})$$

to approximate

$$P(w_n|w_1 \ldots w_{n-1})$$

e.g., $P(page|on)$.

- Trigram model: look two words in the past.
- N-gram model: look into $N-1$ words in the past.

N-gram models

- N-gram:

$$P(w_n|w_1 \cdots w_{n-1}) \approx P(w_n|w_{n-N+1} \cdots w_{n-1})$$

- Bigram:

$$P(w_1, \cdots, w_n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

- Use **Maximum Likelihood Estimate (MLE):**

$$P(w_n|w_{n-1}) = \frac{count(w_{n-1}w_n)}{\sum_w count(w_{n-1}w)}$$

$$P(w_n|w_{n-1}) = \frac{count(w_{n-1}w_n)}{count(w_{n-1})}$$

N-gram models

- Use Maximum Likelihood Estimate (MLE) for N-gram:

$$P(w_n|w_{n-N+1} \cdots w_{n-1}) = \frac{count(w_{n-N+1} \cdots w_{n-1}w_n)}{count(w_{n-N+1} \cdots w_{n-1})}$$

- Bigrams capture syntactic dependencies such as a **noun** comes after **eat**, and that a **verb** comes after **to** etc.
- In practice, using 3-grams and 4-grams are common. We also use log probability to get larger numbers instead of probabilities.

Example of bigrams

- Bigram probabilities
  1. * I love cheese STOP
  2. * Cheese and crackers are delicious STOP
  3. * I prefer swiss cheese STOP

$P(\texttt{I}|*) = \frac{2}{3}$
$P(\texttt{Cheese}|*) = \frac{1}{3}$
$P(\texttt{STOP}|\texttt{Cheese}) = \frac{2}{3}$
$P(\texttt{prefer}|\texttt{I}) = \frac{1}{2}$

- Probability of a sentence can be obtained by multiplying the the bigram probabilities.

$$P(\texttt{*I eat cheese STOP}) = P(\texttt{I}|*)P(\texttt{eat}|\texttt{I})P(\texttt{cheese}|\texttt{eat})P(\texttt{STOP}|\texttt{cheese})$$

Evaluation

- Use a **training corpus** and **test corpus**
- To compare two language models, calculate the probability of the test corpus with both models. Pick the one with a higher probability
- Use **Perplexity**: Inverse probability of the test corpus normalized by the number of words in the test, N.

$$\text{Perplexity}(w_1w_2 \cdots w_N) = P(w_1w_2 \cdots w_N)^{-\frac{1}{N}}$$

- Perplexity

$$\text{Perplexity}(w_1w_2 \cdots w_N) = P(w_1w_2 \cdots w_N)^{-\frac{1}{N}}$$

$$\text{Perplexity}(w_1w_2 \cdots w_N) = (\prod_{i=1}^{N} P(w_i|w_1 \cdots w_{i-1}))^{-\frac{1}{N}}$$

- For bigrams:

$$\text{Perplexity}(w_1w_2 \cdots w_N) = (\prod_{i=1}^{N} P(w_i|w_{i-1}))^{-\frac{1}{N}}$$

- The higher the conditional probability, the lower the perplexity.
- Empirically, the more information provided by the N-gram, the lower the perplexity (the word sequence is captured).

Smoothing

$$P(\texttt{*I eat cheese STOP}) = P(\texttt{I}|\texttt{*})P(\texttt{eat}|\texttt{I})P(\texttt{cheese}|\texttt{eat})P(\texttt{STOP}|\texttt{cheese})$$

- Some probabilities may be zero!
- We modify the N-gram counts:

$$P(w_j) = \frac{\text{count}(w_j)}{N}$$

$$P_L(w_j) = \frac{\text{count}(w_j) + 1}{N + V}$$

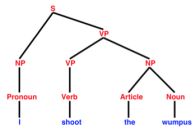
Progress in NLP

Big progress

- **Tagging**
  - Text ⇒ Tagged Text
  - – Part of Speech tagging
    - **I(P) shoot(V) the(A) wumpus(N)**
  - – Name Entity Recognition
    - **Yesterday(time) I(person) bought five(quantity) books from Amazon (Co.)**
- **Text classification (Spam filtering)**

Good progress

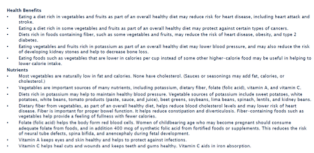
- **Parsing**
  - Exhibit the grammatical structure of a sentence: Text ⇒ Tree



- **Sentiment analysis**
  - √ Fantastic… truly a wonderful family movie
  - × I got so boooored…
- **Machine translation**
- **Information extraction**

Work in progress

- **Summarization**



Eating vegetables is healthy.

- **Question/Answering**
- **Dialog Systems: Siri, echo, etc.**