Natural Language Processing

Édouard Grave, Armand Joulin

Facebook AI Research
egrave@fb.com

What is natural language processing (NLP)?

- Process, analyze and/or produce natural language
- Interact with computers using natural language
- Natural language understanding:
  - language as input $\mapsto$ "information" as output
- Natural language generation:
  - "information" as input $\mapsto$ language as output
- Sometimes, both: machine translation, summarization
- Related fields:
  - machine learning,
  - artificial intelligence,
  - computational linguistics

What is natural language processing (NLP)?

- Why is NLP important? Lot of information stored as text!
- Text processing has many applications
  - text classification: spam detection, topic detection, etc
  - machine translation
  - web search / ranking
  - medical reports
  - etc

This course

- Goal: introduce core ideas at the basis of modern NLP algorithms
- Focus on machine learning applied to NLP
- In particular, we will mostly talk about applications, not so much intermediate linguistic structure
- We will cover techniques used in industry (Facebook, Google, ...)

Why is NLP hard?

- Ambiguity! Some examples of ambiguous headlines:

  Iraqi head seeks arms

  Enraged cow injures farmer with axe

  Miners refuse to work after death

  Two Soviet ships collide, one dies

  Dealers will hear car talk at noon

- Ambiguity can be lexical, syntactic, pragmatic

Why is NLP hard?

Human: Are there direct flights from Paris to Santiago?

Bot: Yes, there is an Air France flight leaving at 11:40PM.

Human: How long does it takes to go there?

Bot: The flight takes 14h35m.

Human: How much would that cost?

- Needs discourse knowledge, domain knowledge, linguistic knowledge

But also (sometimes) surprisingly easy!

- Many techniques might seem naive / simplistic
- But when used on large data: works surprisingly well!
- Always try simple methods first: often hard to beat, when well implemented.

Some milestones in NLP

- 1948: Shannon's "A Mathematical Theory of Communication"
- 1950: Turing introduces his famous test
- 1954: Georgetown–IBM translation experiment
- 1956: Dartmouth Workshop
- 1957: Chomsky's "Syntactic Structures"
- 1960s-1970s: Rule based systems (ELIZA, SHRDLU)
- 1980s-1990s: Statistical methods
- 2000s: Machine learning
- 2010s: Neural networks / Deep Learning

Disclaimer: this is a non-exhaustive selection.

A Mathematical Theory of Communication (1948)

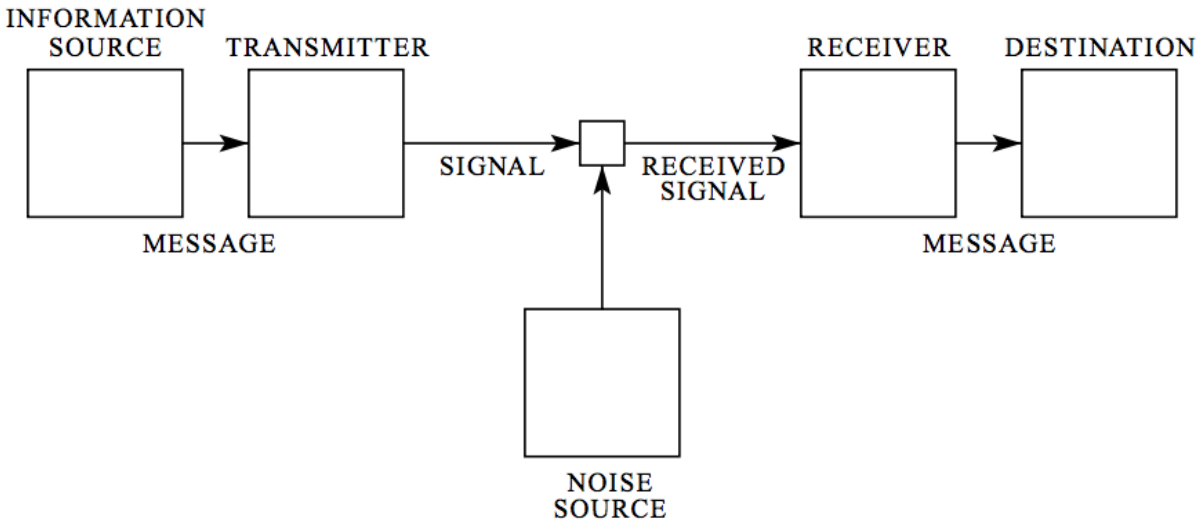


Fig. 1—Schematic diagram of a general communication system.

"The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point"

- Application to transmission (e.g. telephone) or compression

A Mathematical Theory of Communication (1948)

"We can think of a discrete source as generating the message, symbol by symbol. [...] Conversely, any stochastic process which produces a discrete sequence of symbols chosen from a finite set may be considered a discrete source. This will include such cases as:

1. *Natural written languages such as English, German, Chinese."*

- Examples of how to approximate English with stochastic processes:
  - First-order (independent symbols with English frequencies):

    `OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTP`

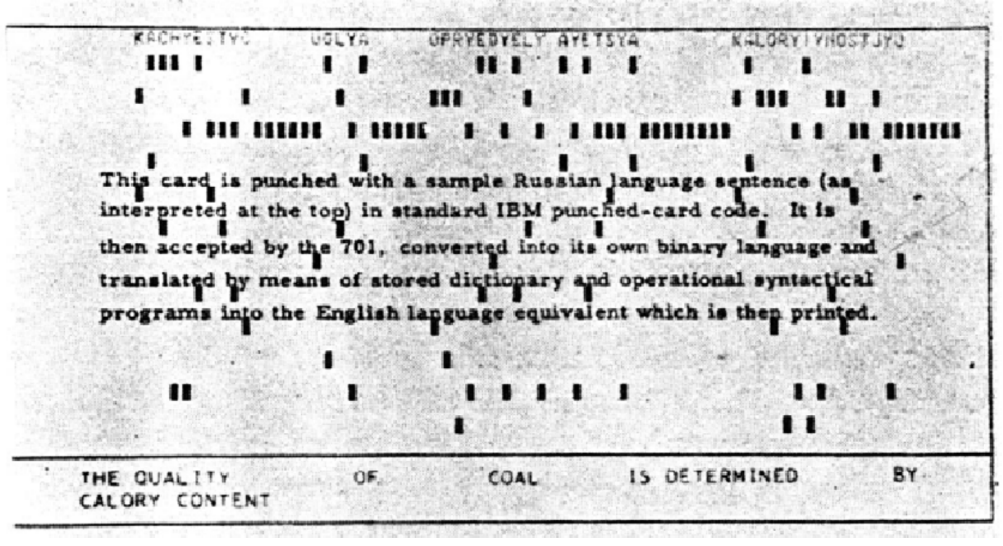
  - Third-order (trigram structure as in English):

    `IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE.`

- First example of language model (we'll cover this in Lecture 3).

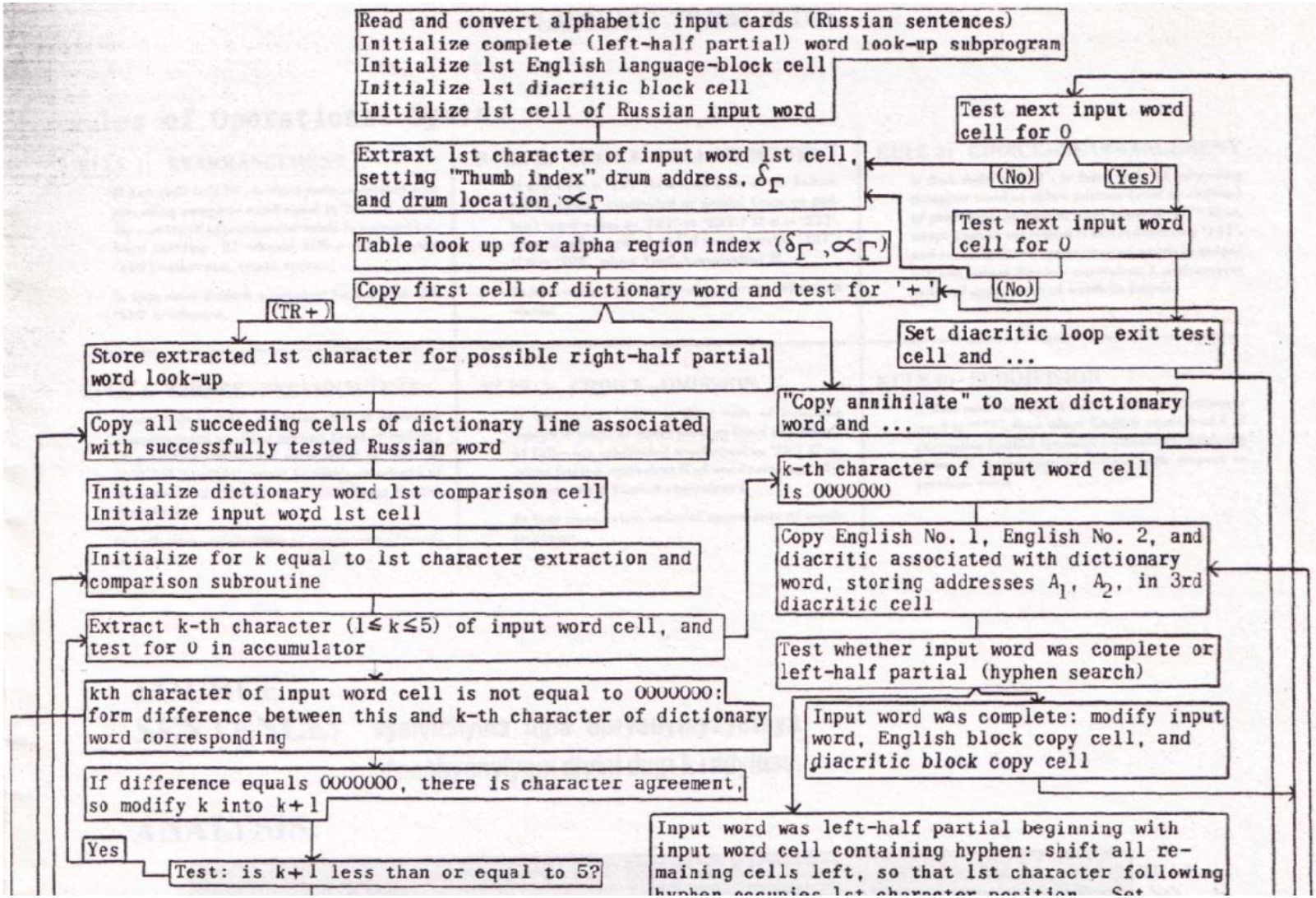
Georgetown-IBM experiment (1954)

- Demonstration of Russian-English translation system at IBM NY
- Some headlines:
  - "Electronic brain translates Russian"
  - "The bilingual machine"
  - "Robot brain translates Russian into King's English"
- Demonstration included approx. 60 sentences:
  - mainly about organic chemistry
  - few general interest sentences



Georgetown-IBM experiment (1954)

- Rule based system:



Georgetown-IBM experiment (1954)

- System had a vocabulary of size 250 words.
- Each Russian word had multiple codes and English translations

russian word	translation 1	translation 2	codes
doma	at home	houses	R5 C1
pravo	right	law	R4 C2

- "Word-for-word" translation
- Six rules (using word codes) to disambiguate/rearrange words

```
IF W[i] has code R1 and W[i-1] has code O2
  THEN reverse word order
  ELSE keep word order
```

Georgetown-IBM experiment (1954)

- Impressive results can be obtained in controlled settings

  But do not generalize well!

- Be very wary of claims of "superhuman" performance!

- After demonstration: claims that MT would be solved in 3-5 years.
- More than 60 years later, we are still not there

Credit: `xkcd.com/1425`

Three Models for the Description of Language (1956)

- Hierarchy of formal grammar, introduced by Noam Chomsky
- A grammar is a set of symbols and production rules
- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
        D -> the ;      N -> man ;
      N -> book ;      V -> took ;
```

Three Models for the Description of Language (1956)

- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
          D -> the ;      N -> man ;
         N -> book ;      V -> took ;
```

- Starting from symbol `S`:
  - `NP VP`
  - `NP V NP`
  - `D N V NP`
  - `D N V D N`
  - `the N V D N`
  - `the man V D N`
  - `the man took D N`
  - `the man took the N`
  - `the man took the book`

Three Models for the Description of Language (1956)

- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
          D -> the ;      N -> man ;
         N -> book ;      V -> took ;
```

- Starting from symbol S:
  - NP VP
  - NP V NP
  - D N V NP
  - D N V D N
  - the N V D N
  - the man V D N
  - the man took D N
  - the man took the N
  - the man took the book

Three Models for the Description of Language (1956)

- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
          D -> the ;      N -> man ;
         N -> book ;      V -> took ;
```

- Starting from symbol S:
  - NP VP
  - NP V NP
  - D N V NP
  - D N V D N
  - the N V D N
  - the man V D N
  - the man took D N
  - the man took the N
  - the man took the book

Three Models for the Description of Language (1956)

- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
        D -> the ;      N -> man ;
      N -> book ;      V -> took ;
```

- Starting from symbol S:
  - NP VP
  - NP V NP
  - D N V NP
  - D N V D N
  - the N V D N
  - the man V D N
  - the man took D N
  - the man took the N
  - the man took the book

Three Models for the Description of Language (1956)

- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
        D -> the ;      N -> man ;
      N -> book ;      V -> took ;
```

- Starting from symbol S:
  - NP VP
  - NP V NP
  - D N V NP
  - D N V D N
  - the N V D N
  - the man V D N
  - the man took D N
  - the man took the N
  - the man took the book

Three Models for the Description of Language (1956)

- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
        D -> the ;        N -> man ;
      N -> book ;       V -> took ;
```

- Starting from symbol S:
  - NP VP
  - NP V NP
  - D N V NP
  - D N V D N
  - the N V D N
  - the man V D N
  - the man took D N
  - the man took the N
  - the man took the book

Three Models for the Description of Language (1956)

- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
        D -> the ;        N -> man ;
      N -> book ;       V -> took ;
```

- Starting from symbol S:
  - NP VP
  - NP V NP
  - D N V NP
  - D N V D N
  - the N V D N
  - the man V D N
  - the man took D N
  - the man took the N
  - the man took the book

Three Models for the Description of Language (1956)

- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
        D -> the ;      N -> man ;
      N -> book ;      V -> took ;
```

- Starting from symbol S:
  - NP VP
  - NP V NP
  - D N V NP
  - D N V D N
  - the N V D N
  - the man V D N
  - the man took D N
  - the man took the N
  - the man took the book

Three Models for the Description of Language (1956)

- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
        D -> the ;        N -> man ;
      N -> book ;       V -> took ;
```

- Starting from symbol S:
  - NP VP
  - NP V NP
  - D N V NP
  - D N V D N
  - the N V D N
  - the man V D N
  - the man took D N
  - the man took the N
  - the man took the book

Three Models for the Description of Language (1956)

- Small grammar for English:

```
S -> NP VP ;      VP -> V NP ;      NP -> D N ;
        D -> the ;        N -> man ;
      N -> book ;       V -> took ;
```

- Starting from symbol S:
  - NP VP
  - NP V NP
  - D N V NP
  - D N V D N
  - the N V D N
  - the man V D N
  - the man took D N
  - the man took the N
  - the man took the book

Three Models for the Description of Language (1956)

- Many (other) applications in computer science
  - Context free grammar are use to described computer language
  - Regular languages are used for regular expressions (`sed`, `grep`...)

Syntactic Structures (1957)

- Book about the syntax of language: based on symbols and rules

(1) Colorless green ideas sleep furiously.

*(2) *Furiously sleep ideas green colorless.*

It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical model for grammaticalness, these sentences will be ruled out on identical grounds as equally "remote" from English. Yet (1), though nonsensical, is grammatical, while (2) is not grammatical.

ELIZA (1966)

- Created at MIT AI Lab by Joseph Weizenbaum
- One of the first chatbot, simulated a Rogerian psychotherapist

Human: Men are all alike.
ELIZA: IN WHAT WAY
Human: They're always bugging us about something or another.
ELIZA: CAN YOU BE MORE SPECIFIC
Human: Well, my boyfriend made me come here.
ELIZA: YOUR BOYFRIEND MADE YOU COME HERE
Human: He says I'm depressed much of the time.
ELIZA: I AM SORRY TO HEAR YOU ARE DEPRESSED

ELIZA (1966)

- Used pattern matching and substitution to simulate understanding:
  - a list of keywords are searched in the input
  - if keyword found:
    input is transformed, using a rule associated to keyword
  - if no keyword found:
    content-free remark (or earlier transformation) is used.

- Examples of rules:
  - `X always Y -> Can you think of a specific example`
  - `X because Y -> Is that the real reason?`
  - `X dream about Y -> How do you feel about Y in reality?`
  - `X I want Y -> What would it mean if you got Y?`
  - `X is like Y -> In what way is it that X is like Y?`

ELIZA (1966)

- Take away: intelligent "behavior" can emerge from simplistic rules!
- For specific domains, rules are still very useful
  - regular expressions (regexp) are a powerful tool for NLP
  - e.g. extract and parse dates/time from documents
- Some chatbots are still rule-based

A Statistical Approach to Language Translation (1988)

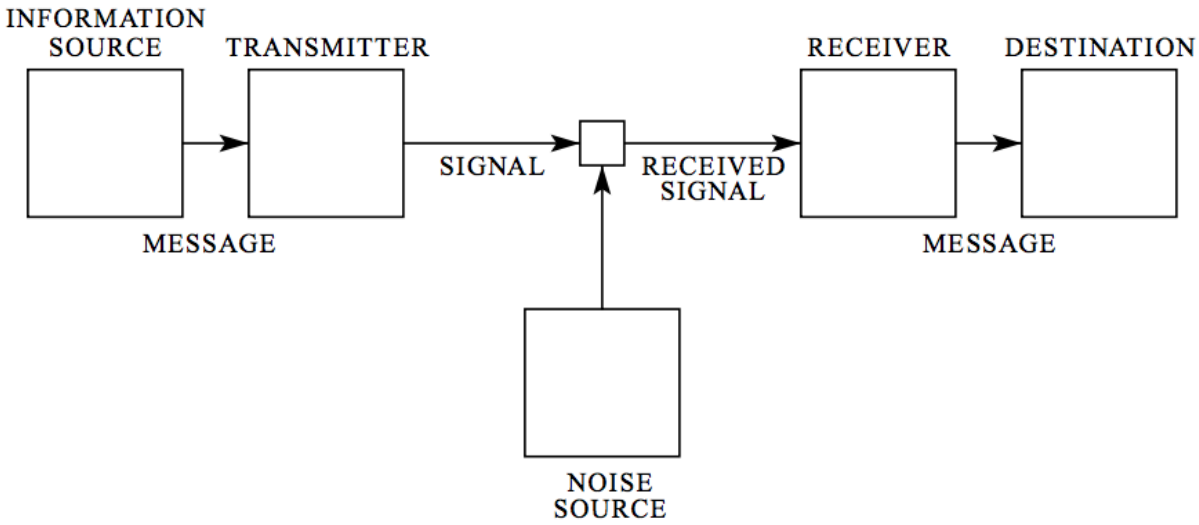


Fig. 1—Schematic diagram of a general communication system.

- Back to information theory ideas from Shannon
- Technique initially developed for speech recognition (1974)
- Main idea, noisy channel model:

$$\texttt{source} = \texttt{text} \longrightarrow \texttt{noisy channel} \longrightarrow \texttt{observation} = \texttt{speech}$$

A Statistical Approach to Language Translation (1988)

- Main idea, noisy channel model:

$$\texttt{source} = \texttt{text} \longrightarrow \texttt{noisy channel} \longrightarrow \texttt{observation} = \texttt{speech}$$

- "Easier" to model the process from text to speech
- Similar idea for translation (e.g. from Russian to English):

$$\texttt{source} = \texttt{English} \longrightarrow \texttt{noisy channel} \longrightarrow \texttt{observation} = \texttt{Russian}$$

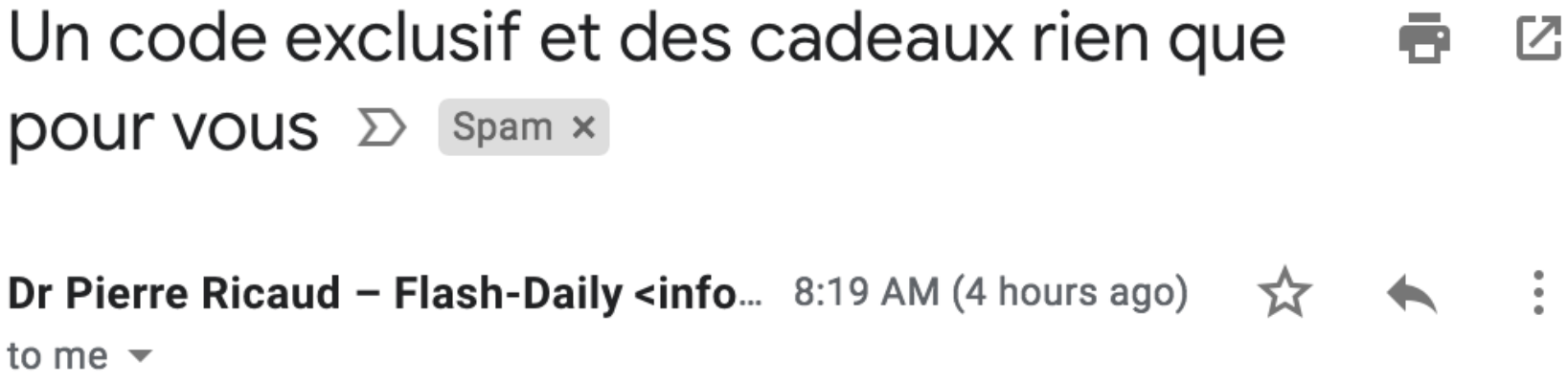
- or for optical character recognition:

$$\texttt{source} = \texttt{text} \longrightarrow \texttt{noisy channel} \longrightarrow \texttt{observation} = \texttt{image}$$

This lecture: text classification

- Introduction
- Naive Bayes
- Text Preprocessing
- Perceptron
- Logistic Regression

Is this spam?


Un code exclusif et des cadeaux rien que pour vous
Spam ×
Dr Pierre Ricaud – Flash-Daily <info…
8:19 AM (4 hours ago)
to me

Is this review positive?

01/02/2019

I'm gonna be dreaming about that bacon scone. Usually I find scones to be really dry but this one was so moist and had so much flavor packed in a small package. Smoky bacon, sweet maple and all that savory buttery salty goodness. I just wish I got another.

What is this article about?

Enriching Word Vectors with Subword Information

Piotr Bojanowski* and **Edouard Grave*** and **Armand Joulin** and **Tomas Mikolov**
Facebook AI Research
{bojanowski,egrave,ajoulin,tmikolov}@fb.com

Abstract

Continuous word representations, trained on large unlabeled corpora are useful for many natural language processing tasks. Popular models that learn such representations ignore the morphology of words, by assigning a distinct vector to each word. This is a limitation, especially for languages with large vocabularies and many rare words. In this paper, we propose a new approach based on the skipgram

et al., 2010; Baroni and Lenci, 2010). In the neural network community, Collobert and Weston (2008) proposed to learn word embeddings using a feed-forward neural network, by predicting a word based on the two words on the left and two words on the right. More recently, Mikolov et al. (2013b) proposed simple log-bilinear models to learn continuous representations of words on very large corpora efficiently.

What is text classification?

- Given a piece of text $\mathbf{w} = (w_1, ..., w_n)$, assign it a class $y \in \mathcal{Y}$.
- Spam detection:

$$\mathcal{Y} = \{\texttt{spam}, \texttt{notspam}\}$$

- Sentiment analysis:

$$\mathcal{Y} = \{\texttt{positive}, \texttt{negative}\} \text{ or } \mathcal{Y} = \{*, **, ***, ****\}$$

- Language identification:

$$\mathcal{Y} = \{\texttt{English}, \texttt{French}, \texttt{German}, ...\}$$

- Topic classification:

$$\mathcal{Y} = \{\texttt{1960s\_comedy\_films}, \texttt{Australian\_cricketers}, ...\}$$

- Class are sometimes also called labels or categories.

What is text classification?

- From now, we map classes to integers, so $\mathcal{Y} = \{1, ..., C\}$.
- Potential solution: rule based classifier:

```
IF "the" IN w AND "is" IN w
  THEN lang = "English"
```

- This works well, but misses many English documents
- Also: many rules to write, hard and expensive to maintain
- Instead: use machine learning
- Assume training/learning data: $(\mathbf{w}_i, y_i)$

First classifier: naive Bayes

- Represent documents as bags of words:

$$\{ \texttt{what a great movie} \} = \{ \texttt{a movie great what} \}$$

- Order does not matter

- A word can appear multiple time (bags != sets)

$$\{ \texttt{what a great movie} \} \neq \{ \texttt{what a great great movie} \}$$

First classifier: naive Bayes

- We want a probabilistic model of classes given a document:

$$P(y \mid w_1, ..., w_n)$$

- Hard to define such a model (space of document is very large!)
- Use the Bayes rule:

$$\begin{aligned} P(y \mid w_1, ..., w_n) &= \frac{P(y, w_1, ..., w_n)}{P(w_1, ..., w_n)} \\ &= \frac{P(w_1, ..., w_n \mid y)P(y)}{P(w_1, ..., w_n)} \\ &\propto P(w_1, ..., w_n \mid y)P(y). \end{aligned}$$

- $P(y)$ is the prior, $P(w_1, ..., w_n \mid y)$ is the data likelihood
- We will see that learning the prior and likelihood is easy

First classifier: naive Bayes

- How to estimate $P(w_1, ..., w_n \mid y)$?
- We can (always) factorize this probability as:

$$P(w_1, ..., w_n \mid y) = P(w_1 \mid y)P(w_2 \mid w_1, y)...P(w_n \mid w_1, ...w_{n-1}, y)$$

- Now, make conditional indepence assumption:

$$w_i \text{ is independent of } w_j \text{ given class } y$$

- Then, we get:

$$P(w_1, ..., w_n \mid y) = P(w_1 \mid y)P(w_2 \mid y)...P(w_n \mid y)$$

- Much simpler model to estimate.
- Note: harder to apply cond. assumption to $P(y \mid w_1, ..., w_n)$ directly.

First classifier: naive Bayes

- $P(Y = k)$ and $P(W_i = j \mid Y = k)$: categorical distributions
- Parametrized by π and p:

$$P(Y = k) = \pi_k \quad \text{and} \quad P(W_i = j \mid Y = k) = p_{j,k}$$

- How to estimate the parameters π_k and $p_{j,k}$?
- Estimate the parameters with maximum of likelihood:

$$\max \prod_i P(\mathbf{w}_i \mid y_i) P(y_i)$$

- Equivalent to

$$\max \log \left(\prod_i P(\mathbf{w}_i \mid y_i) p(y_i) \right) = \sum_i \log P(\mathbf{w}_i \mid y_i) + \sum_i \log p(y_i).$$

- We can estimate the parameter of documents given the class for each class independently, as well as the probability of classes.

Reminder: maximum of likelihood for categorical distributions

- Let Y a random variable taking its value in $\{1, ..., K\}$. Distribution is parametrized by vector $\pi \in \mathbb{R}^K$, such that $\sum_k \pi_k = 1$ and $\pi_k \geq 0$.
- We have

$$\begin{aligned}\sum_i \log p(y_i) &= \sum_i \sum_k \log \pi_k^{\delta(y_i=k)} \\ &= \sum_k \sum_i \delta(y_i = k) \log \pi_k \\ &= \sum_k C(k) \log \pi_k.\end{aligned}$$

- To maximize over π, let's introduce the Lagrangian

$$\sum_k C(k) \log \pi_k - \lambda \left(\sum_k \pi_k - 1 \right)$$

- Setting the derivatives w.r.t π and λ to 0, we get

$$\frac{C(k)}{\pi_k} - \lambda = 0$$
$$\sum_k \pi_k - 1 = 0$$

- By summing the K first equality, we get

$$\sum_k C(k) = \lambda \sum_k \pi_k$$

- Thus, $\lambda = \sum_k C(k)$ and $\pi_k = \frac{C(k)}{\sum_{k'} C(k')}$.

First classifier: naive Bayes

- Back to naive Bayes... we thus have:

$$\pi_k = \frac{C(k)}{\sum_{k'} C(k')} \quad \text{and} \quad p_{j,k} = \frac{C(j,k)}{\sum_{j'} C(j',k)}$$

- Very simple to learn: just count words!

Example

food was amazing
great food
food was bland
amazing restaurant
restaurant is not great

	food	was	amazing	great	restaurant	bland	is	not	total
+	2	1	2	1	1	0	0	0	7
-	1	1	0	1	1	1	1	1	7

Example

	food	was	amazing	great	restaurant	bland	is	not	total
+	2	1	2	1	1	0	0	0	7
-	1	1	0	1	1	1	1	1	7

$$\mathbf{w} = \texttt{food was great}$$

$$\begin{aligned} P(y = + \mid \mathbf{w}) &\propto P(y = +)P(\texttt{food} \mid +)P(\texttt{was} \mid +)P(\texttt{great} \mid +) \\ &\propto \frac{3}{5} \times \frac{2}{7} \times \frac{1}{7} \times \frac{1}{7} = \frac{6}{1715} \\ P(y = - \mid \mathbf{w}) &\propto P(y = -)P(\texttt{food} \mid -)P(\texttt{was} \mid -)P(\texttt{great} \mid -) \\ &\propto \frac{2}{5} \times \frac{1}{7} \times \frac{1}{7} \times \frac{1}{7} = \frac{2}{1715} \end{aligned}$$

$$\hat{y} = +$$

Example

	food	was	amazing	great	restaurant	bland	is	not	total
+	2	1	2	1	1	0	0	0	7
-	1	1	0	1	1	1	1	1	7

$$\mathbf{w} = \texttt{food is great}$$

$$P(y = + \mid \mathbf{w}) \propto P(y = +)P(\texttt{food} \mid +)P(\texttt{is} \mid +)P(\texttt{great} \mid +)$$
$$\propto \frac{3}{5} \times \frac{2}{7} \times \frac{0}{7} \times \frac{1}{7} = \frac{0}{1715}$$
$$P(y = - \mid \mathbf{w}) \propto P(y = -)P(\texttt{food} \mid -)P(\texttt{is} \mid -)P(\texttt{great} \mid -)$$
$$\propto \frac{2}{5} \times \frac{1}{7} \times \frac{1}{7} \times \frac{1}{7} = \frac{2}{1715}$$

$$\hat{y} = -$$

First classifier: naive Bayes

- Practical solution to unseen words: Laplacian smoothing.
- Just add α to all counts:

$$p_{j,k} = \frac{\alpha + C(j,k)}{\alpha V + \sum_{j'} C(j',k)}.$$

- The parameter α acts as a regularizer.
- When α goes to infinity, our classifier becomes independent of data!
- How to pick α?
  - Use train / valid / test splits
  - Use k-fold cross validation

Alternative naive Bayes formulation

- We can now ignore word order: bag of words assumption.
- Represent documents as word counts $\mathbf{x}$:

  x_j is the number of times word j appears in document
- Then, distribution $P(X = \mathbf{x} \mid Y = k)$ is a multinomial distribution:

$$P(X = \mathbf{x} \mid Y = k) = \frac{n!}{\prod_j x_j!} \prod_j p_{j,k}.$$

- Main difference with previous model?

$$B(\mathbf{x}) = \frac{n!}{\prod_j x_j!}.$$

- But! For a given $\mathbf{x}$, this is constant. So, does not change the decision function (i.e. maximizing over classes)!
- In practice? two assumptions: independence and bag of words.

Alternative naive Bayes formulation II

- Instead of bag of words: consider sets of words.
- Represent documents as word presences $\mathbf{x}$:

$$x_j = \begin{cases} 1 & \text{if word } j \text{ appears in document,} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

- Then, the distribution $P(X = \mathbf{x} \mid Y = k)$ is given by

$$P(X = \mathbf{x} \mid Y = k) = \prod_j p_{j,k}^{x_j} (1 - p_{j,k})^{1 - x_j}.$$

where

$$p_{j,k} = \frac{C(j,k)}{C(k)}$$

Multinomial versus Bernouilli naive Bayes

	Features	# of features	frequency or presence?	NB	ME	SVM
(1)	unigrams	16165	freq.	**78.7**	N/A	72.8
(2)	unigrams	"	pres.	81.0	80.4	**82.9**

Credit: **B. Pang, L. Lee, S. Vaithyanathan.** *Thumbs up? Sentiment Classification using Machine Learning Techniques*. EMNLP 2002

Naive Bayes as log linear model

- In the binary case, decision function

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, P(\mathbf{w} \mid y) P(y)$$

- Positive class equivalent to positive log probability ratio:

$$\log \left(\frac{P(\mathbf{w} \mid y = 1) P(y = 1)}{P(\mathbf{w} \mid y = 0) P(y = 0)} \right) > 0$$

- In case of multinomial naive Bayes:

$$\begin{aligned} \log \left(\frac{\pi_1 \prod_j p_{j,1}^{x_j}}{\pi_0 \prod_j p_{j,0}^{x_j}} \right) &= \sum_j x_j \log \frac{p_{j,1}}{p_{j,0}} + \log \frac{\pi_1}{\pi_0} \\ &= \mathbf{x}^\top \mathbf{w} + b \end{aligned}$$

- We'll come back to this later.

Naive Bayes: recap

- Very fast, easy to implement: just count words!
- Generative probabilistic model (i.e. $P(Y, W)$)
- Assumption: words are independent of each others (given class)
- Represent text as bag or set of words (try both!)
- Strong baseline for text classification

Evaluation

$$\begin{aligned}\texttt{accuracy} &= \frac{1}{n}\sum_{i=1}^{n} 1\{\hat{y}_i = y_i\} \\ &= \frac{\#\ \texttt{correct predictions}}{\#\ \texttt{text examples}}\end{aligned}$$

- **Problem 1:** all mistakes are not equal

  Spam filtering: worst to classify email as spam, than spam as email

- **Problem 2:** imbalanced datasets.

  Classifying everything as negative might have (almost) same accuracy as more useful models: hard to rank models

Evaluation

- Finer look at mistakes: confusion matrix

	$y = 1$	$y = -1$
$\hat{y} = 1$	TP	FP
$\hat{y} = -1$	FN	TN

- True Positive: positive example classified as positive
- True Negative: negative example classified as negative
- False Positive: negative example classified as positive
- False Negative: positive example classified as negative

Evaluation

- Finer look at mistakes: confusion matrix

	$y = 1$	$y = -1$
$\hat{y} = 1$	$\texttt{TP}$	$\texttt{FP}$
$\hat{y} = -1$	$\texttt{FN}$	$\texttt{TN}$

- Precision: $\texttt{PR} = \dfrac{\texttt{TP}}{\texttt{TP} + \texttt{FP}}$
- Recall: $\texttt{RE} = \dfrac{\texttt{TP}}{\texttt{TP} + \texttt{FN}}$
- F1-score: $\texttt{F1} = 2\,\dfrac{\texttt{PR} \times \texttt{RE}}{\texttt{PR} + \texttt{RE}}$

Evaluation

- Finer look at mistakes: confusion matrix

	$y = 1$	$y = -1$
$\hat{y} = 1$	TP	FP
$\hat{y} = -1$	FN	TN

- Precision: $\mathtt{PR} = \dfrac{\mathtt{TP}}{\mathtt{TP} + \mathtt{FP}}$
- Recall: $\mathtt{RE} = \dfrac{\mathtt{TP}}{\mathtt{TP} + \mathtt{FN}}$
- F1-score: $\mathtt{F1} = 2\,\dfrac{\mathtt{PR} \times \mathtt{RE}}{\mathtt{PR} + \mathtt{RE}}$

Evaluation

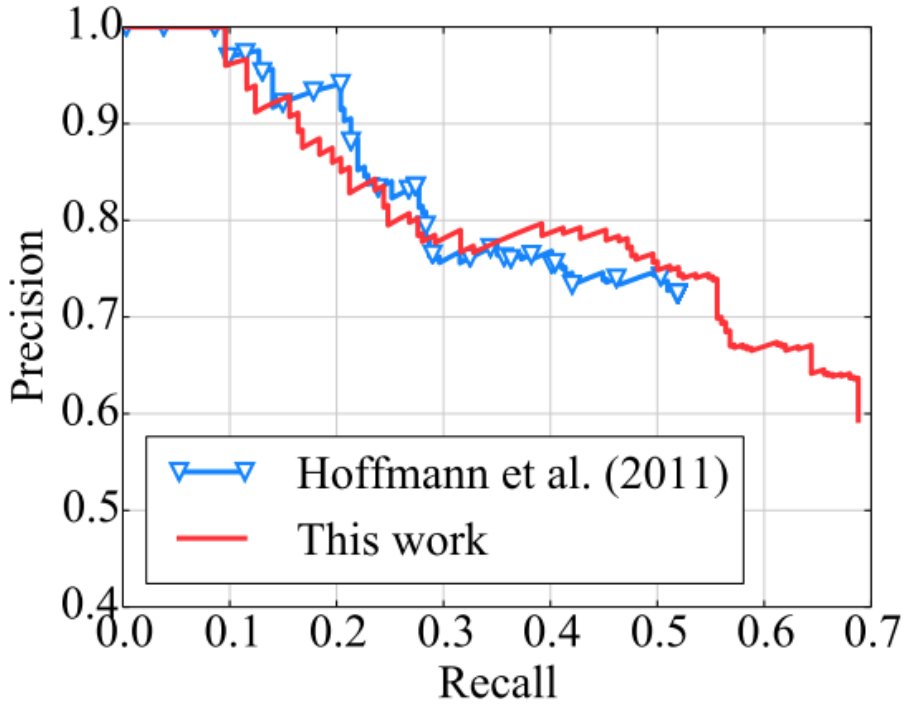
- Finer look at mistakes: confusion matrix

	$y = 1$	$y = -1$
$\hat{y} = 1$	TP	FP
$\hat{y} = -1$	FN	TN

- Precision: $\texttt{PR} = \dfrac{\texttt{TP}}{\texttt{TP} + \texttt{FP}}$
- Recall: $\texttt{RE} = \dfrac{\texttt{TP}}{\texttt{TP} + \texttt{FN}}$
- F1-score: $\texttt{F1} = 2\,\dfrac{\texttt{PR} \times \texttt{RE}}{\texttt{PR} + \texttt{RE}}$

Evaluation

- Use classifier score to plot precision/recall curve:



- Different application: different point on curve (business decision)

What is a word?

The Bengal and Siberian tigers are amongst the tallest cats in shoulder height.

- How many words? 12? 13? 14?
- Type v.s. Tokens:
  - Type is an element of the vocabulary
  - Token is an instance of type in running text
- How to split this input in tokens? tokenization
- Are *tigers* and *tiger* the same type? lemmatization
- What about *The* and *the*? normalization

Tokenization

The tiger's closest living relatives were previously thought to be the lion, leopard and jaguar.

- Split text into tokens: cannot just split on spaces

  `[The] [tiger] ['s] [closest] [living] [relatives] [were] [previously] [thought] [to] [be] [the] [lion] [,] [leopard] [and] [jaguar] [.]`

- Issues for tokenization:
  - *tiger's* → `[tiger] ['s]`
  - *isn't* → `[isn't]` or `[is] [not]` or `[is] [n't]` ?
  - *low-frequency* → `[low-frequency]` or `[low] [frequency]` ?
  - *Mr.* → `[Mr.]` or `[Mr] [.]` ?
- Some arbitrary choices: always use same tokenizer!

Tokenization is language dependent

- How to tokenize French word *l'avion*?

  `[l'] [avion]` or `[le] [avion]`

- How to tokenize Finnish word *rautatieasema* (railway station)?

  `[rauta] [tie] [asema]` (`[iron] [road] [station]`)

- What about Japanese or Chinese?

  虎為亚洲特有种类。

  [虎] [為] [洲亚] [特有] [种类] [。]

- How to tokenize Vietnamese words *cà phê* (coffee)?

  `[cà_phê]`

How to tokenize?

- Use an existing tokenizer (e.g. Moses tokenizer):

```
> git clone https://github.com/moses-smt/mosesdecoder.git
> cat data | perl mosesdecoder/scripts/tokenizer/tokenizer.perl
```

- Alternatives (in python): spaCy, NLTK, Stanford CoreNLP

How does tokenization work?

- Rule based tokenizers (e.g. spaCy tokenizer):
  - Split input on spaces
  - For each substring, does it matches exception rule?
  - For each substring, can a suffix or prefix be split off?

- Example, from spaCy documentation
  - `[‘‘Let’s] [go] [to] [N.Y.!’’]`

How does tokenization work?

- Rule based tokenizers (e.g. spaCy tokenizer):
  - Split input on spaces
  - For each substring, does it matches exception rule?
  - For each substring, can a suffix or prefix be split off?

- Example, from spaCy documentation
  - [‘‘Let’s] [go] [to] [N.Y.!’’] PREFIX
  - [‘‘] [Let’s] [go] [to] [N.Y.!’’]

How does tokenization work?

- Rule based tokenizers (e.g. spaCy tokenizer):
  - Split input on spaces
  - For each substring, does it matches exception rule?
  - For each substring, can a suffix or prefix be split off?

- Example, from spaCy documentation
  - [‘‘Let’s] [go] [to] [N.Y.!’’] PREFIX
  - [‘‘] [Let’s] [go] [to] [N.Y.!’’] EXCEPTION
  - [‘‘] [Let] [’s] [go] [to] [N.Y.!’’]

How does tokenization work?

- Rule based tokenizers (e.g. spaCy tokenizer):
  - Split input on spaces
  - For each substring, does it matches exception rule?
  - For each substring, can a suffix or prefix be split off?

- Example, from spaCy documentation
  - `[‘‘Let’s] [go] [to] [N.Y.!’’]` PREFIX
  - `[‘‘] [Let’s] [go] [to] [N.Y.!’’]` EXCEPTION
  - `[‘‘] [Let] [’s] [go] [to] [N.Y.!’’]` SUFFIX
  - `[‘‘] [Let] [’s] [go] [to] [N.Y.!] [’’]`

How does tokenization work?

- Rule based tokenizers (e.g. spaCy tokenizer):
  - Split input on spaces
  - For each substring, does it matches exception rule?
  - For each substring, can a suffix or prefix be split off?

- Example, from spaCy documentation
  - [‘‘Let’s] [go] [to] [N.Y.!’’] PREFIX
  - [‘‘] [Let’s] [go] [to] [N.Y.!’’] EXCEPTION
  - [‘‘] [Let] [’s] [go] [to] [N.Y.!’’] SUFFIX
  - [‘‘] [Let] [’s] [go] [to] [N.Y.!] [’’] SUFFIX
  - [‘‘] [Let] [’s] [go] [to] [N.Y.] [!] [’’]

How does tokenization work?

- Rule based tokenizers (e.g. spaCy tokenizer):
  - Split input on spaces
  - For each substring, does it matches exception rule?
  - For each substring, can a suffix or prefix be split off?

- Example, from spaCy documentation
  - `[‘‘Let’s] [go] [to] [N.Y.!’’]` PREFIX
  - `[‘‘] [Let’s] [go] [to] [N.Y.!’’]` EXCEPTION
  - `[‘‘] [Let] [’s] [go] [to] [N.Y.!’’]` SUFFIX
  - `[‘‘] [Let] [’s] [go] [to] [N.Y.!] [’’]` SUFFIX
  - `[‘‘] [Let] [’s] [go] [to] [N.Y.] [!] [’’]` EXCEPTION
  - `[‘‘] [Let] [’s] [go] [to] [N.Y.] [!] [’’]`

How does tokenization work?

- Rule based tokenizers (e.g. spaCy tokenizer):
  - Split input on spaces
  - For each substring, does it matches exception rule?
  - For each substring, can a suffix or prefix be split off?

- Example, from spaCy documentation
  - `[‘‘Let’s] [go] [to] [N.Y.!’’]` PREFIX
  - `[‘‘] [Let’s] [go] [to] [N.Y.!’’]` EXCEPTION
  - `[‘‘] [Let] [’s] [go] [to] [N.Y.!’’]` SUFFIX
  - `[‘‘] [Let] [’s] [go] [to] [N.Y.!] [’’]` SUFFIX
  - `[‘‘] [Let] [’s] [go] [to] [N.Y.] [!] [’’]` EXCEPTION
  - `[‘‘] [Let] [’s] [go] [to] [N.Y.] [!] [’’]` DONE

How does tokenization work?

- Alternative formulation as sequence tagging problem:

  for each character, predict if space should be inserted after
- State-of-the-art approach for languages such as Chinese, Japanese, Thai or Vietnamese

Normalization

- Case folding: *The* v.s. *the*
- Periods in words: *U.S.A.* v.s. *USA*
- Keep or remove punctuation? Stopwords?
- Inflected variants: *walk* v.s. *walking*, *is* v.s. *are*
- Date? Numers?
  - *2018-03-21* $\rightarrow$ `YYYY-MM-DD`
  - *21 March 2018* $\rightarrow$ `2018-03-21`
  - *3.1415* $\rightarrow$ `D.DDDD`
  - *300,000* $\rightarrow$ `DDD,DDD`
- Normalization is application dependent:
  - *U.S.* v.s. *us* might be important (e.g. named entity detection)
  - Keeping punctuation might be important (e.g. sentiment analysis)

Lemmatization

- *Cars* and *car* are two inflected (word)forms of the same lemma
- Some languages have rich morphology and many inflected forms
- For example, verbs in French or Italian...

  `apprendre` = { *apprenaient*, *apprenais*, *apprenait*, *apprend*, *apprendra*, *apprendrai*, *apprendraient*, *apprendrais*, *apprendrait*, *apprendras*, *apprendre*, *apprendrez*, *apprendriez*, *apprendrions*, *apprendrons*, *apprendront*, *apprends*, *apprenez*, *appreniez*... }

- ... or nouns in Polish

  `uniwersytet` = { *uniwersytecie*, *uniwersytet*, *uniwersytetach*, *uniwersytetami*, *uniwersytetem*, *uniwersytetom*, *uniwersytetowi*, *uniwersytetu*, *uniwersytety*, *uniwersytetów* }

- Lemmatization: replacing wordforms by headword form

Lemmatization

The tiger's closest living relatives were previously thought to be the lion, leopard and jaguar.

`the tiger 's closest living relative be previously think to be the lion , leopard and jaguar .`

- Ambiguity:
  - adjective *living* has headword *living*
  - verb *living* has headword *live*
- Resource intensive (need a part-of-speech tagger and dictionnary!)
- Formulated as sequence tagging problem

Text preprocessing: recap

- Always start with tokenization
- Text normalization: application dependent.

  Try different preprocessing
- ML models, very sensitive to preprocessing:

  Always use same preprocessing for train and test!

Generative v.s. discriminative learning

- In naive Bayes, we have conditional independence assumption:

$$P(w_i \mid y, w_1, ..., w_{i-1}) = P(w_i \mid y)$$

- What if we want to add features such as prefix?
- Clearly, we don't have

$$P(\text{un-} \mid y, \text{undefinite}) = P(\text{un-} \mid y)$$

- Same is true for bigrams, etc
- Solution: directly learn the decision function $x \mapsto y$

Perceptron

- Each document is represented by a vector $\mathbf{x}$ (e.g. word count).
- Linear classifier: $y = \text{sign}(\mathbf{w}^\top \mathbf{x})$.
- How to learn the parameter vector $\mathbf{w}$ from examples?

Perceptron: algorithm

- At each time step, sample an example $\mathbf{x}_i, y_i$.
- Compute the predicted class $\hat{y} = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_i)$
- If no error (i.e. $\hat{y} = y_i$):

  do not change the parameters: $\mathbf{w}_{t+1} = \mathbf{w}_t$
- If error (i.e. $\hat{y} \neq y_i$):

  update the parameters: $\mathbf{w}_{t+1} = \mathbf{w}_t + y_i \mathbf{x}_i$

Perceptron: intuition

- Why the update rule $\mathbf{w}_{t+1} = \mathbf{w}_t + y_i \mathbf{x}_i$?
- If the example is positive (i.e. $y_i = 1$), then

$$\mathbf{w}_{t+1}^\top \mathbf{x}_i = \mathbf{w}_t^\top \mathbf{x}_i + \mathbf{x}_i^\top \mathbf{x}_i$$

- Similarly, for negative examples (i.e. $y_i = -1$), then

$$\mathbf{w}_{t+1}^\top \mathbf{x}_i = \mathbf{w}_t^\top \mathbf{x}_i - \mathbf{x}_i^\top \mathbf{x}_i$$

Perceptron: convergence

- Does this algorith converges?
- Under some assumptions, yes.

- Let's assume that the examples are linearly separable, i.e. there exists a constant $\gamma > 0$ and a unit vector $\mathbf{w}_*$ such that

$$\forall i,\ y_i \mathbf{w}_*^\top \mathbf{x}_i \geq \gamma \qquad \text{and} \qquad \|\mathbf{w}_*\| = 1$$

- Let's also assume that examples are bounded, i.e. $\|\mathbf{x}_i\| \leq R$.

- Then, the perceptron algorithm makes at most $\dfrac{R^2}{\gamma^2}$ errors.

Perceptron: convergence

- When algorithm makes error k at time t, we have

$$\begin{aligned}\mathbf{w}_{t+1}^\top \mathbf{w}_* &= \mathbf{w}_t^\top \mathbf{w}_* + y_t \mathbf{x}_t^\top \mathbf{w}_* \\ &\geq \mathbf{w}_t^\top \mathbf{w}_* + \gamma\end{aligned}$$

- Thus, by induction, after k errors, $\mathbf{w}_{t+1}^\top \mathbf{w}_* \geq k\gamma$.
- Then, after k errors, $\|\mathbf{w}_{t+1}\| \geq k\gamma$.

- When algorithm makes error k at time t, we also have

$$\begin{aligned}\|\mathbf{w}_{t+1}\|^2 &= \|\mathbf{w}_t\|^2 + 2y_t \mathbf{w}_t^\top \mathbf{x}_t + \|\mathbf{x}_t\|^2 \\ &\leq \|\mathbf{w}_t\|^2 + R^2.\end{aligned}$$

- Then, after k errors, $\|\mathbf{w}_{t+1}\|^2 \leq kR^2$.

- Combining the two bounds, we get $k^2\gamma^2 \leq kR^2$, hence the result.

Perceptron

- Cheap and easy to implement
- Online algorithm: scales to large datasets
- Can use any features easily

- Limitation: all classifiers that separate the data are equivalent.
- Limitation: convergence depends on the margin γ.

  If classes are not well separated, slow convergence.

Logistic regression

- Reminder from naive Bayes: log probability ratio is linear.
- What if we directly use this?

$$\log\left(\frac{P(Y=1 \mid \mathbf{x})}{P(Y=-1 \mid \mathbf{x})}\right) = \mathbf{w}^\top \mathbf{x}$$

- Then, we get

$$\log\left(\frac{P(Y=1 \mid \mathbf{x})}{1-P(Y=1 \mid \mathbf{x})}\right) = \mathbf{w}^\top \mathbf{x}$$

$$\frac{P(Y=1 \mid \mathbf{x})}{1-P(Y=1 \mid \mathbf{x})} = \exp(\mathbf{w}^\top \mathbf{x})$$

$$P(Y=1 \mid \mathbf{x}) = \exp(\mathbf{w}^\top \mathbf{x}) - \exp(\mathbf{w}^\top \mathbf{x}) P(Y=1 \mid \mathbf{x})$$

$$P(Y=1 \mid \mathbf{x}) = \frac{\exp(\mathbf{w}^\top \mathbf{x})}{1+\exp(\mathbf{w}^\top \mathbf{x})}$$

Sigmoid function a.k.a. logistic function

- The logistic or sigmoid function is defined by

$$\sigma(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{1+\exp(-x)}$$

- We also have

$$P(Y=-1 \mid X=\mathbf{x}) = \frac{1}{1+\exp(\mathbf{w}^\top \mathbf{x})}$$

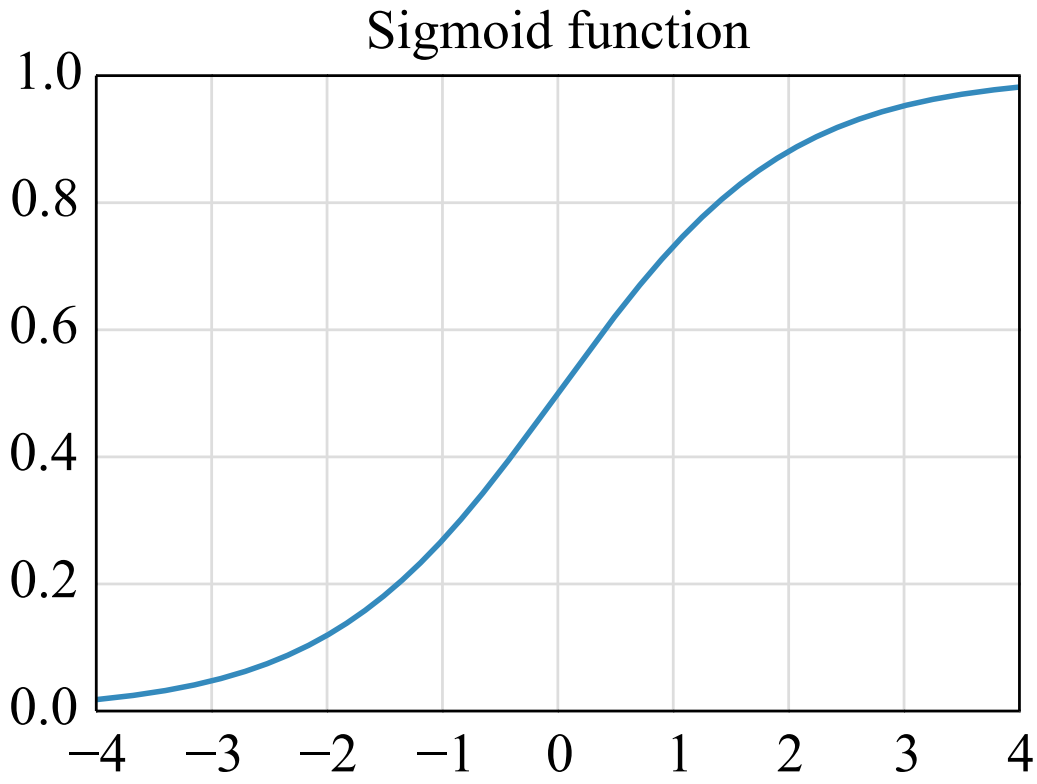
- Hence

$$P(Y=1 \mid X=\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$$
$$P(Y=-1 \mid X=\mathbf{x}) = \sigma(-\mathbf{w}^\top \mathbf{x})$$

- Finally:

$$P(Y=y \mid X=\mathbf{x}) = \sigma(y\mathbf{w}^\top \mathbf{x})$$

Sigmoid function a.k.a. logistic function



Logistic regression: objective function

- As for naive Bayes, maximum (log) likelihood estimator:

$$\max_{\mathbf{w}} \frac{1}{n} \log \left(\prod_i P(Y = y_i \mid X = \mathbf{x}_i) \right)$$

- We get

$$\max_{\mathbf{w}} \frac{1}{n} \sum_i \log P(Y = y_i \mid X = \mathbf{x}_i)$$

- Using the logistic function

$$\max_{\mathbf{w}} \frac{1}{n} \sum_i \log \sigma(y_i \mathbf{w}^\top \mathbf{x}_i)$$

- And we get

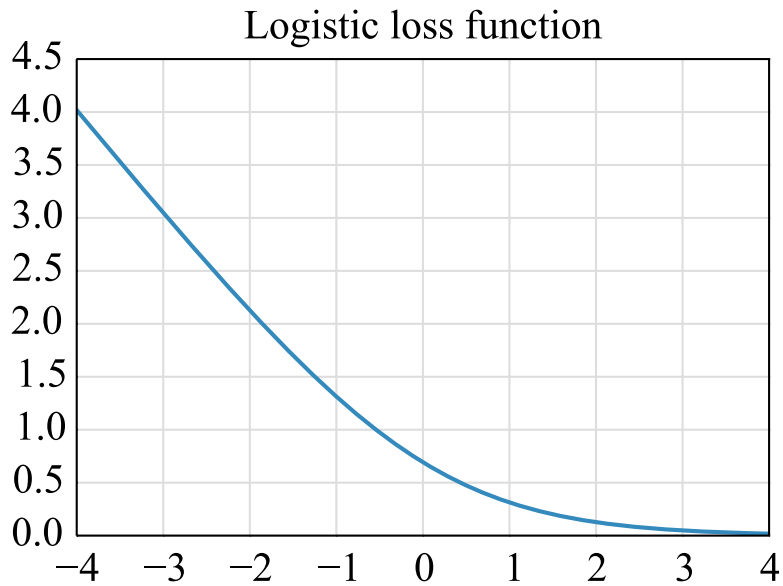
$$\max_{\mathbf{w}} \frac{1}{n} \sum_i -\log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i))$$

Logistic regression: objective function

- Replacing the max by min, we finally get

$$\min_{\mathbf{w}} \frac{1}{n} \sum_i \log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i))$$

- The function $x \mapsto \log(1 + \exp(-x))$ is called the logistic loss.



Logistic regression: optimization

- How to find the min?

  As opposed to naive Bayes, no close form solution!

- Use tool from convex optimization: gradient descent.
- We have

$$\sigma'(x) = \frac{\exp(-x)}{(1+\exp(-x))^2} = \frac{1}{1+\exp(-x)} \times \frac{\exp(-x)}{1+\exp(-x)}$$

- Thus

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

- And

$$\frac{\partial \log \sigma(x)}{\partial x} = 1 - \sigma(x)$$

- Derivative is non-increasing, hence log-likelihood is concave (and loss is convex)

Logistic regression: optimization

- Thus, the gradient of our objective function is

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = -\frac{1}{n} \sum_{i=1}^{n} (1 - \sigma(y_i \mathbf{w}^\top \mathbf{x}_i)) y_i \mathbf{x}_i$$

- We can optimize our objective function with gradient descent:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_t)$$

- This algorithm requires to do a full pass over the data at each step.
- This algorithm is also slow to converge: error $O(1/t)$

Logistic regression: optimization

- Instead of first order optimization (i.e. use only gradient), use second order information (i.e. hessian)
- We have

$$\begin{aligned}\nabla^2_{\mathbf{w}}\mathcal{L}(\mathbf{w}) &= \frac{1}{n}\sum_i (1-\sigma(y_i\mathbf{w}^\top\mathbf{x}_i))\sigma(y_i\mathbf{w}^\top\mathbf{x}_i)\mathbf{x}_i\mathbf{x}_i^\top \\ &= \mathbf{X}^\top\mathbf{D}\mathbf{X}\end{aligned}$$

  where $\mathbf{D} = \text{diag}\left[(1-\sigma(y_i\mathbf{w}^\top\mathbf{x}_i))\sigma(y_i\mathbf{w}^\top\mathbf{x}_i)\right]$
- Then, we apply Newton step:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - (\mathbf{X}^\top\mathbf{D}\mathbf{X})^{-1}\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}_t)$$

- Newton method has a quadratic convergence rate: $O(\exp(-\rho 2^t))$
- But! $d \times d$ matrix inversion: $O(d^3)$. Does not scale to large models

Logistic regression: optimization

- Method of choice in machine learning: stochastic gradient descent.
- Instead of computing gradient on full dataset, use only one example:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t(1 - \sigma(y_i \mathbf{w}^\top \mathbf{x}_i))y_i \mathbf{x}_i$$

- Slow convergence, but very efficient to evaluate.
- In machine learning, do not need to optimize below estimation error (see Bottou and Bousquet (2008), The Tradeoffs of Large Scale Learning)

- Some references about optimization:
  - M. Hardt course at Berkeley, `ee227c.github.io`
  - F. Bach tutorial, `www.di.ens.fr/~fbach/fbach_mlss_2018.pdf`

Logistic regression: optimization

Algorithm (hinge loss, λ=1e-4)	Training Time*	Primal cost	Test Error
SMO (SVMLight)	≃ 16000 secs[1]	0.2275	6.02%
Cutting Plane (SVMPerf)	≃ 45 secs[2]	0.2278	6.03%
Hinge Loss SDCA (LibLinear -s 3 -B 1)	2.5 secs	-	6.02%
SGD (`svmsgd`)	< 1 sec.	0.2275	6.02%
ASGD (`svmasgd`)	< 1 sec.	0.2275	6.02%

[1] Extrapolated from a 23642 seconds run on a 30% slower machine.
[2] Extrapolated from a 66 seconds run on a 30% slower machine.
* All timings exclude data loading time.

Credit: `https://leon.bottou.org/projects/sgd`

Logistic regression versus perceptron

- Reminder, perceptron update rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + (1 - H(y_i \mathbf{w}^\top \mathbf{x}_i)) y_i \mathbf{x}_i$$

  where H is the Heaviside step function (equal to 0 for negative argument and 1 otherwise).

- Ignoring the learning rate η_t, for logistic regression we have

$$\mathbf{w}_{t+1} = \mathbf{w}_t + (1 - \sigma(y_i \mathbf{w}^\top \mathbf{x}_i)) y_i \mathbf{x}_i$$

Logistic regression versus perceptron

- Reminder, perceptron update rule:

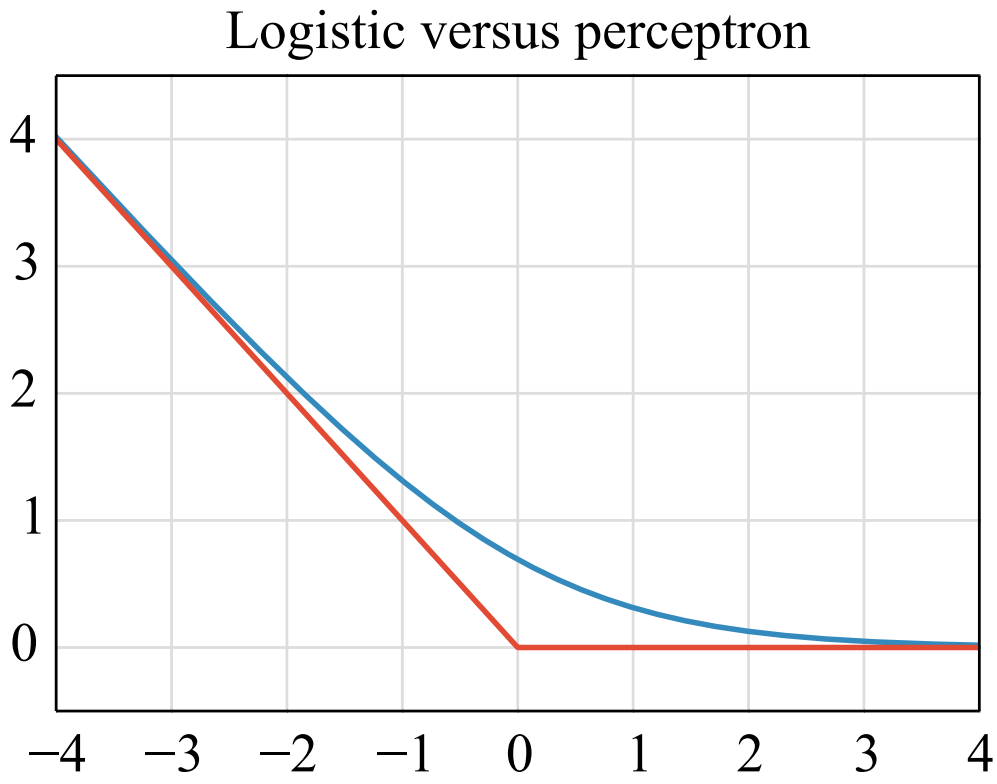
$$\mathbf{w}_{t+1} = \mathbf{w}_t + (1 - H(y_i \mathbf{w}^\top \mathbf{x}_i)) y_i \mathbf{x}_i$$

  where H is the Heaviside step function (equal to 0 for negative argument and 1 otherwise).

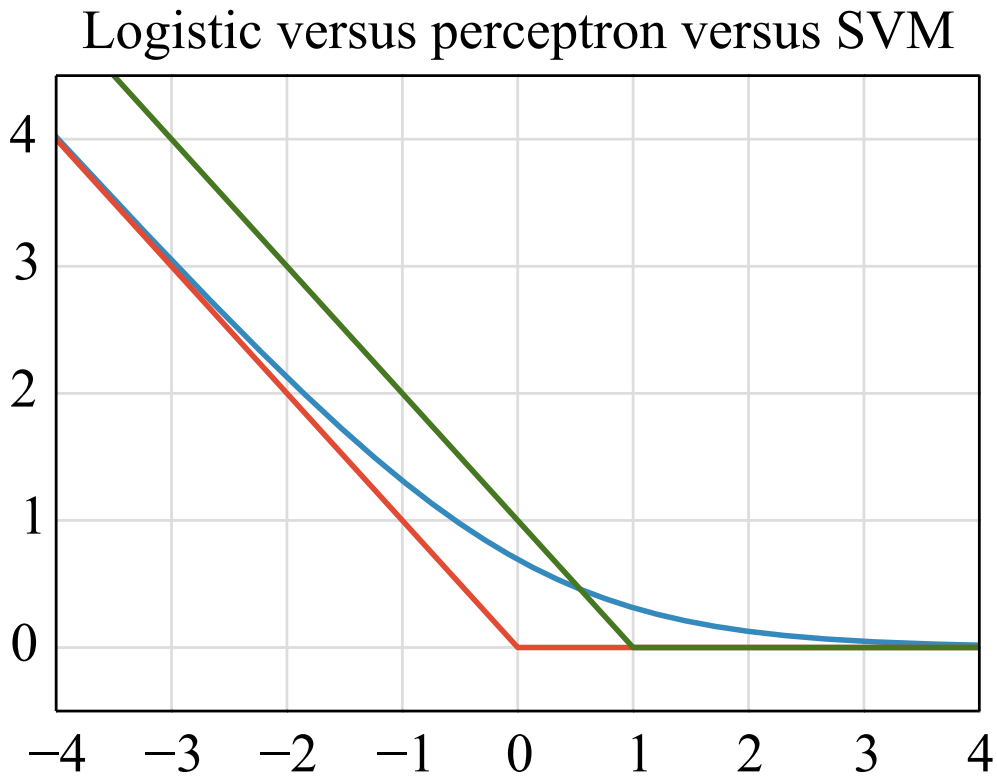
- Ignoring the learning rate η_t, for logistic regression we have

$$\mathbf{w}_{t+1} = \mathbf{w}_t + (1 - \sigma(y_i \mathbf{w}^\top \mathbf{x}_i)) y_i \mathbf{x}_i$$

Logistic regression versus perceptron



Logistic regression versus perceptron versus SVM



Logistic regression: multiclass case

- Logistic regression has straightforward extension to multiple classes

$$P(Y = k \mid X = \mathbf{x}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{k'} \exp(\mathbf{w}_{k'}^\top \mathbf{x})}$$

- Need to learn one parameter vectors per class.
- Note: since $\sum_k P(Y = k \mid X = \mathbf{x}) = 1$, we could set $\mathbf{w}_1 = 0$

$$\begin{aligned} P(Y = k \mid X = \mathbf{x}) &= \frac{\exp(-\mathbf{w}_1^\top \mathbf{x}) \exp(\mathbf{w}_k^\top \mathbf{x})}{\exp(-\mathbf{w}_1^\top \mathbf{x}) \sum_{k'} \exp(\mathbf{w}_{k'}^\top \mathbf{x})} \\ &= \frac{\exp((\mathbf{w}_k - \mathbf{w}_1)^\top \mathbf{x})}{\sum_{k'} \exp((\mathbf{w}_{k'} - \mathbf{w}_1)^\top \mathbf{x})} \end{aligned}$$

- But does not change in practice.

Softmax function

- The multivariate function f defined by

$$f_i(\mathbf{s}) = \frac{\exp(s_i)}{\sum_{i'} \exp(s_{i'})}$$

  is called the softmax.
- This function maps $\mathbb{R}^K$ to the $K-1$ simplex:

$$f_i(\mathbf{s}) > 0 \text{ and } \sum_{i=1}^{K} f_i(\mathbf{s}) = 1$$

- Appears in many applications: linear discriminant analysis, neural networks, etc.

Softmax function

- The gradient is equal to:

$$\frac{\partial f_i}{\partial s_j}(\mathbf{s}) = f_i(\mathbf{s})(\delta_{ij} - f_j(\mathbf{s}))$$

- Then

$$\frac{\partial \log(f_i)}{\partial s_j}(\mathbf{s}) = (\delta_{ij} - f_j(\mathbf{s}))$$

Softmax function

- Actually, should be called softargmax:

$$\lim_{\lambda \to \infty} f(\lambda \mathbf{s}) = \delta_k$$

  where $k = \text{argmax}_i s_i$

- Computational stability: remove the max $m = \max_i s_i$

$$\frac{\exp(s_i - m)}{\sum_{i'} \exp(s_{i'} - m)}$$

Class-based softmax

- Limitation: expensive to compute when large number of classes.
- Can we do better? Yes, by approximating the softmax function.

- Idea: partition the classes to L subsets, c_k is subset of label k

$$P(Y = k \mid X = \mathbf{x}, C = c_k)P(C = c_k \mid X = \mathbf{x})$$

- If each subset contains $\sqrt{K}$ classes, then, train complexity if $O(\sqrt{K})$

- How to partition the classes? Random, frequency, similarity

Hierarchical softmax

- Idea: organize classes in binary tree, classes corresponding to leaves
- Then: learn binary classifiers for nodes of tree (left or right?)
- A class k is encoded by the path $\mathcal{P}_k$ from root to corresponding leaf, and corresponding decisions y_{ik} (go left or right):

$$P(Y = k \mid X = \mathbf{x}) = \prod_{i \in \mathcal{P}_k} \sigma(y_{ik} \mathbf{w}_i^\top \mathbf{x})$$

- If tree is balanced: depth is $O(\log(K))$

Hierarchical softmax

- The complexity at train time is $\log(K)$.
- What about prediction? How to find the most probable label?
- We can do depth first search.
- Each time a path has a smaller probability than best so far, stop.
- Start exploring the most probable branch.

Hierarchical softmax

- Python inspired pseudo code:

```
def dfs(x, score, node, best):
  if score < best.score:
    return best

  if node.is_leaf:
    return (score, node.label)

  s = sigmoid(dot(node.w, x))
  best = dfs(x, score * s, node.left, best)
  best = dfs(x, score * (1-s), node.right, best)
  return best
```

Hierarchical softmax

- Use log to prevent underflow

```
def dfs(x, score, node, best):
  if score < best.score:
    return best

  if node.is_leaf:
    return (score, node.label)

  s = sigmoid(dot(node.w, x))
  best = dfs(x, score + log(s), node.left, best)
  best = dfs(x, score + log(1-s), node.right, best)
  return best
```

Hierarchical softmax

- Predict categories from stack exchange questions
- 735 labels, 10,000 word types

Model	PR at 1	RE at 1	Time (sec)
Full softmax	56.8	24.6	29.1
Hierarchical softmax	57.1	24.7	5.1

- On dataset with 300,000 labels: from hours to minutes

Logistic regression: multiclass case

- Reminder of multiclass logistic regression:

$$P(Y = k \mid X = \mathbf{x}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{k'} \exp(\mathbf{w}_{k'}^\top \mathbf{x})}$$

- Need to learn one parameter vectors per class.

- The distribution over the labels given $\mathbf{x}$ is thus:

$$p(\mathbf{x}) = \text{softmax}(\mathbf{W}\mathbf{x})$$

Bigram features

- *the cat ate the mouse* versus *the mouse ate the cat*
- Same bag of words representations, very different meanings!

- Important for sentiment analysis:
  - *I did not wait and I like[d] the restaurant*
  - *I wait[ed] and I did not like the restaurant*

- What can we do? Use *n*-gram features!

`[the]` `[cat]` `[ate]` `[the]` `[mouse]`
`[the_cat]` `[cat_ate]` `[ate_the]` `[the_mouse]`

Model	AG	DBP	Yelp F.	Yah. A.	Amz. F.
unigram features	91.5	98.1	60.4	72.0	55.8
bigram features	92.5	98.6	63.9	72.3	60.2

Table: Test accuracy [%] on classification datasets.

Bigram features

- *the cat ate the mouse* versus *the mouse ate the cat*
- Same bag of words representations, very different meanings!

- Important for sentiment analysis:
  - *I did not wait and I like[d] the restaurant*
  - *I wait[ed] and I did not like the restaurant*

- What can we do? Use *n*-gram features!

`[the]` `[cat]` `[ate]` `[the]` `[mouse]`
`[the_cat]` `[cat_ate]` `[ate_the]` `[the_mouse]`

Model	AG	DBP	Yelp F.	Yah. A.	Amz. F.
unigram features	91.5	98.1	60.4	72.0	55.8
bigram features	92.5	98.6	63.9	72.3	60.2

Table: Test accuracy [%] on classification datasets.

Bigram features

- *the cat ate the mouse* versus *the mouse ate the cat*
- Same bag of words representations, very different meanings!

- Important for sentiment analysis:
  - *I did not wait and I like[d] the restaurant*
  - *I wait[ed] and I did not like the restaurant*

- What can we do? Use *n*-gram features!

`[the]` `[cat]` `[ate]` `[the]` `[mouse]`
`[the_cat]` `[cat_ate]` `[ate_the]` `[the_mouse]`

Model	AG	DBP	Yelp F.	Yah. A.	Amz. F.
unigram features	91.5	98.1	60.4	72.0	55.8
bigram features	92.5	98.6	63.9	72.3	60.2

Table: Test accuracy [%] on classification datasets.

Matrix factorization

- If number of classes and features is large:

  large number of parameters.
- The size of the weight matrix $\mathbf{W}$ is $K \times V$!
- Might not fit in memory for some problems.
- Instead, replace $\mathbf{W}$ by low rank matrix $\mathbf{U}\mathbf{V}^\top$,

$$\mathbf{U} \in \mathbb{R}^{K \times d} \qquad \text{and} \qquad \mathbf{V} \in \mathbb{R}^{V \times d}$$

- Then, $\mathbf{W}\mathbf{x} = \mathbf{U}(\mathbf{V}^\top \mathbf{x})$.
- If $\mathbf{x}$ is the count of words, then, $\mathbf{V}^\top \mathbf{x} = \sum_i \mathbf{v}_{w_i}$
- Average the vectors of words appearing in document.
- Next lecture, we will see how to learn such vectors from raw text.

fastText

- Open source implementation of text classification

- Multiclass logistic regression, binary hierarchical softmax
- *n*-gram word features and subword features (see lecture 2)
- Stochastic gradient descent
- Matrix factorization
- Model compression (e.g. langid in less than 1MB)

```
> fasttext supervised -input data.train.txt -output model
> fasttext predict model.bin data.test.txt
```

- More information on `www.fasttext.cc`

fastText – Experiments

Model	AG	DBP	Yelp F.	Yah. A.	Amz. F.
BoW (Zhang et al., 2015a)	88.8	96.6	58.0	68.9	54.6
ngrams (Zhang et al., 2015a)	92.0	98.6	56.3	68.5	54.3
ngrams TFIDF (Zhang et al., 2015a)	92.4	98.7	54.8	68.5	52.4
char-CNN (Zhang et al., 2015a)	87.2	98.3	62.0	71.2	59.5
char-CRNN (Xiao and Cho, 2016)	91.4	98.6	61.8	71.7	59.2
VDCNN (Conneau et al., 2016)	91.3	98.7	64.7	73.4	63.0
`fastText`, $h = 10$	91.5	98.1	60.4	72.0	55.8
`fastText`, $h = 10$, bigram	92.5	98.6	63.9	72.3	60.2

Table: Test accuracy [%] on classification datasets.

Compressing *fastText* models (advanced)

- Parameters of *fastText*: embeddings **U** and classifiers **V**
- With large vocabulary and/or large output space:

  U and **V** can (still) be large

- Use compression technique to reduce memory footprint
- Product quantization approximates a vector $\mathbf{x} \in \mathbb{R}^d$ by

$$\hat{\mathbf{x}} = [q_1(\mathbf{x}^1), ..., q_k(\mathbf{x}^k)],$$

  where $\mathbf{x}^i$ are subvectors in $\mathbb{R}^{d/k}$ and q_i are k-means quantizers.

k-means quantization (advanced)

- Given a set of vectors $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$, run k-means
- Approximate a vector $\mathbf{x}_i$ by the nearest centroid

- If $k = 256$:
  - need $4 \times 256 \times d$ bytes to store centroids
  - need one byte per vector, instead of $4 \times d$

  - Total memory: $1024 \times d + n$ v.s. $4 \times n \times d$

- For NLP application: use $d = 2$ or $d = 4$ in product quantization
- Leads to memory savings of $8\times$ to $16\times$

Dictionary pruning (advanced)

Second strategy to reduce memory footprint: feature selection.

In our case, select K words / ngrams from a trained model.

Find the closest sparse model:

$$\min_{\hat{\mathbf{U}}} \|\mathbf{U} - \hat{\mathbf{U}}\|_2 \quad \text{s.t.} \quad \hat{\mathbf{U}} \in \mathcal{S}_K,$$

where $\mathcal{S}_K$ is the set of matrices with at most K nonzero columns.

Corresponds to keeping the K columns with largest norms.

Dictionary pruning (advanced)

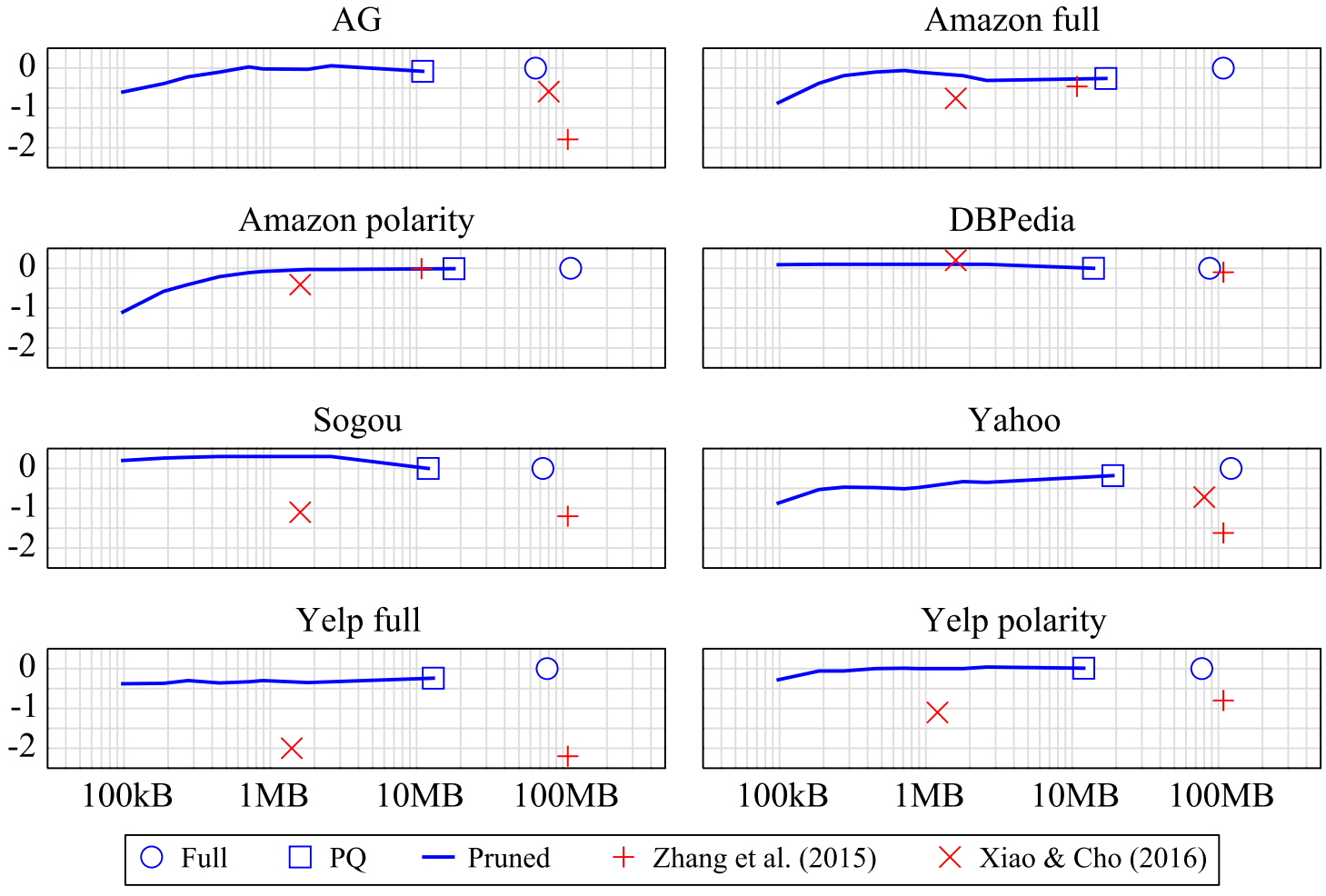


Figure: Loss of accuracy vs model size. We compare models with different level of pruning and the full `fastText` model. We also compare with (Zhang et al., 2015b) and (Xiao and Cho, 2016). Note: the size is in log scale.

fastText – Model compression (advanced)

Dataset	full		64KiB	32KiB	16 KiB
AG	65M	92.1	91.4	90.6	89.1
Amazon full	108M	60.0	58.8	56.0	52.9
Amazon pol.	113M	94.5	93.3	92.1	89.3
DBPedia	87M	98.4	98.2	98.1	97.4
Sogou	73M	96.4	96.4	96.3	95.5
Yahoo	122M	72.1	70.0	69.0	69.2
Yelp full	78M	63.8	63.2	62.4	58.7
Yelp pol.	77M	95.7	95.3	94.9	93.2
Average diff. [%]		0	-0.8	-1.7	-3.5

Table: Performance of very small models. We use a quantization with $k = 1$, hashing and an extreme pruning. The last row shows the average drop of performance for different size.

References I

Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. (2016). Very deep convolutional networks for natural language processing. *arXiv preprint arXiv:1606.01781*.

Xiao, Y. and Cho, K. (2016). Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367*.

Zhang, X., Zhao, J., and LeCun, Y. (2015a). Character-level convolutional networks for text classification. In *Adv. NIPS*.

Zhang, X., Zhao, J., and LeCun, Y. (2015b). Character-level convolutional networks for text classification. In *NIPS*.