# Classifying Documents and Words using Statistical Methods

Tanel Alumäe

# Is this spam?

**Saatja:** Cindy Pinda <membership@mrjbf.org>
**Reply-to:** sweetcindyj68@gmail.com
**Kellele:** tanel.alumae@phon.ioc.ee
**Teema:** I am Cindy
**Kuupäev:** Fri, 17 Nov 2017 06:59:27 +0200 (EET)

Hi,

I am Cindy and my surname is Pinda.I am 18 years old. My father was a very wealthy oil merchant and businessman in Turkey. My father was poisoned to death by his brothers because they want to take over his properties and wealth.My Mother died after giving birth to me.I am currently in my secondary school where i stay in their boarding house for students.

My father left family valuables worth the sum of Two Million United States Dollars in a security company for me which i want you to stand as my trustee to claim this from the security company. They contacted me and informed me that the deposit is due for claim and demand that i should come forward to claim or send a trustee. I cannot handle this so i am contacting you to help me as i am too young and cannot follow the process. I will authorize you to them and they will release the fund to you.i wait to read from you.Write back to me in my email

sweetcindyj68@gmail.com

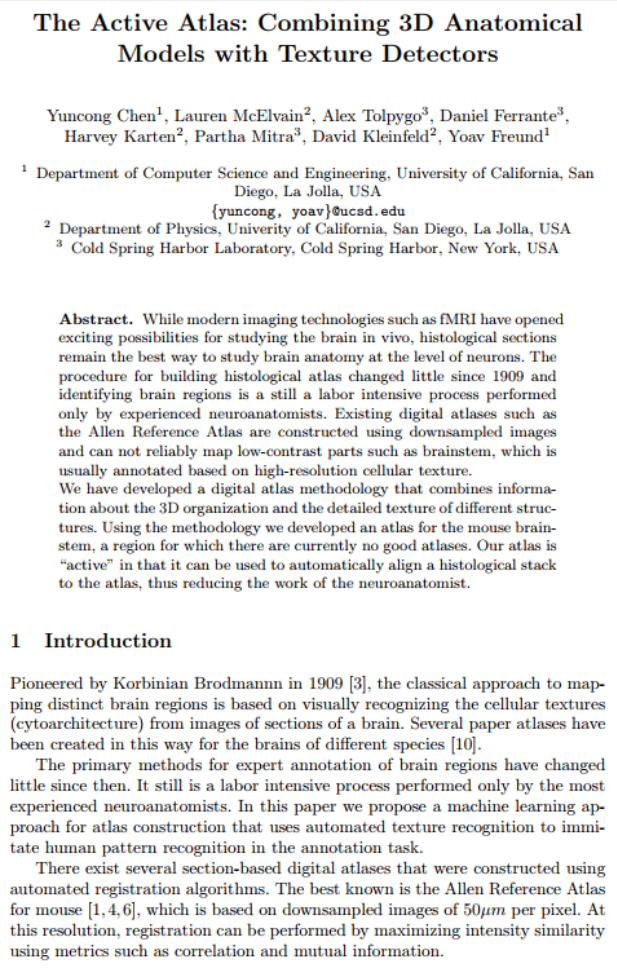
Thanks and God bless.

Cindy Pinda

# Positive or negative review?

- unbelievably disappointing
- Full of zany characters and richly applied satire, and some great plot twists
- this is the greatest screwball comedy ever filmed
- It was pathetic. The worst part about it was the boxing scenes.

# Document categorization



**The Active Atlas: Combining 3D Anatomical Models with Texture Detectors**

Yuncong Chen[1], Lauren McElvain[2], Alex Tolpygo[3], Daniel Ferrante[3], Harvey Karten[2], Partha Mitra[3], David Kleinfeld[2], Yoav Freund[1]

[1] Department of Computer Science and Engineering, University of California, San Diego, La Jolla, USA
{yuncong, yoav}@ucsd.edu
[2] Department of Physics, Univerity of California, San Diego, La Jolla, USA
[3] Cold Spring Harbor Laboratory, Cold Spring Harbor, New York, USA

**Abstract.** While modern imaging technologies such as fMRI have opened exciting possibilities for studying the brain in vivo, histological sections remain the best way to study brain anatomy at the level of neurons. The procedure for building histological atlas changed little since 1909 and identifying brain regions is a still a labor intensive process performed only by experienced neuroanatomists. Existing digital atlases such as the Allen Reference Atlas are constructed using downsampled images and can not reliably map low-contrast parts such as brainstem, which is usually annotated based on high-resolution cellular texture.
We have developed a digital atlas methodology that combines information about the 3D organization and the detailed texture of different structures. Using the methodology we developed an atlas for the mouse brainstem, a region for which there are currently no good atlases. Our atlas is "active" in that it can be used to automatically align a histological stack to the atlas, thus reducing the work of the neuroanatomist.

**1 Introduction**

Pioneered by Korbinian Brodmannn in 1909 [3], the classical approach to mapping distinct brain regions is based on visually recognizing the cellular textures (cytoarchitecture) from images of sections of a brain. Several paper atlases have been created in this way for the brains of different species [10].

The primary methods for expert annotation of brain regions have changed little since then. It still is a labor intensive process performed only by the most experienced neuroanatomists. In this paper we propose a machine learning approach for atlas construction that uses automated texture recognition to immitate human pattern recognition in the annotation task.

There exist several section-based digital atlases that were constructed using automated registration algorithms. The best known is the Allen Reference Atlas for mouse [1, 4, 6], which is based on downsampled images of $50\mu m$ per pixel. At this resolution, registration can be performed by maximizing intensity similarity using metrics such as correlation and mutual information.

?

- Medicine
- Computer science
- Mathematics
- Physics
- Chemistry
- Social science
- Linguistics
- ...

# Formal definition

- Input:
  - Document (text) $d$
  - Finite set of classes $C=\{c_1, c_2, \ldots, c_j\}$
- Output: $c \in C$

# Hand-crafted rules

- Rules based on combinations of words or other features
  - spam: black-listed-address OR (“*dollars*” AND “*have been selected*”)
- Accuracy can be high
  - If rules carefully refined by expert
- But building and maintaining these rules is expensive

# Classification using machine learning

- Input:
  - Document (text) $d$
  - Finite set of classes $C=\{c_1, c_2, \ldots, c_j\}$
  - A training set of $m$ hand-labeled documents $(d_1,c_1),\ldots,(d_m,c_m)$
- Output: a classifier learned from the training corpus $f: d \rightarrow c$

# Advantages of using machine learning

- Advantages:
  - No need to create and manage hand-crafted rules
  - Simple, relatively easy to achieve good accuracy
  - No deep knowledge of the domain or language required
- Disadvantages
  - A large training corpus needed
  - Need to keep the training corpus up-to date and retrain models
  - Difficult to guarantee very high accuracy
  - Difficult to fix a classifier to avoid certain mistakes (black box)

# Machine learning methods

- Possible to use many different machine learning methods for text classification:
  - **Naive Bayes**
  - **Logistic regression** (other names: log-linear model, maximum entropy model)
  - Support vector machine
  - Decision trees, random forests
  - K-Nearest neighbor
  - (Deep) neural networks
  - Recurrent neural networks
  - Transformer-based neural networks
  - Ensembles of the above

# Naive Bayes method

- Simple (although formally naive) classifier
- Uses Bayes rule (probability theory)
- Naive, as it assumes that the words occurring in the document are independent (e.g., occurrence of the word “*economy*” should not have any effect on the probability of seeing “*money*”)
- In reality, the words are not independent
- However, the method works very well for text classification

# Bag of words

Kaks nädalat tagasi tellisin sülearvuti. Ütlesid, et läheb nädal aega. Kui lubatud nädal oli möödas, saatsin kirja ja küsisin, kas on kohal, nad ütlesid, et läheb nädal veel. Siis saatsid kirja umbes tund hiljem, et siiski läheb nädal ja üks päev veel. Täna pidi asi kohal olema. [...] ja ma ei taha enam seda korrata, mitte kunagi enam ei osta neilt mitte midagi. Täielik pettumus. Hinnang: Lauspask

```
nädal: 3  veel: 2      kunagi: 1
läheb: 3  ütlesid: 2   pettumus: 1
ja: 3     ei: 2        tagasi: 1
...
```

# Theory behind Naive Bayes

- Document *d* and class *c*

$$c_{max} = argmax_c P(c|d)$$

$$= argmax_c \frac{P(d|c) \times P(c)}{P(d)}$$

$$= argmax_c P(d|c) \times P(c)$$

Probability that document *d* *belongs to class c*

Bayes rule

Probability that class *c* generated such document

*Prior* probability of class *c*

Dropping the denominator: we don't care about the actual probability, rather than the class for which it is the largest. Therefore, we can just drop the denominator, as it is the same for all classes: the largest probability will still result in the largest pseudo-probability

# Theory continues

- Document consists of words $w_1 w_2 .. w_n$

$$c_{max} = argmax_c P(d|c) \times P(c)$$
$$= argmax_c P(w_1 w_2 .. w_n|c) \times P(c)$$

The second term, *P(c)* is the prior probability of *c*: what's the probability that the next document that we see will belong to class *c*?

# Theory, continues

- In reality, document consists of a sequence of words
- Naive Bayes makes two simplifications (not true in reality):
  - *Bag-of-words:* the order of words is not important
  - *Naive Bayes:* the words are independent
- This allows us to break the probability calculation into subcomponents

$$P(w_1, w_2, ..., w_n|c) = P(w_1|c) \times P(w_2|c) \times ... \times P(w_2|c)$$

$$= \prod_w P(w_i|c)$$

# How to find word-given class probabilities?

- How to calculate $P(w_i|c)$?
- Maximum likelihood estimate:

$$\hat{P}(w_i|c_j) = \frac{Count(w_i, c_j)}{\sum_{w \in V} Count(w_i, c_j)}$$

  - Fraction of times word $w_i$ appears among all words in documents of topic $c_j$
  - Create mega-document for class $j$ by concatenating all docs in this topic
  - Use frequency of $w_i$ in the mega-document

# Problems with maximum likelihood

- What if we have seen no training documents with the word *fantastic* and classified in the topic *positive*?

$$P(fantastic|positive)=\frac{Count(fantastic,positive)}{\sum_{w\in V} Count(w_i,positive)}=\frac{0}{\sum_{w\in V} Count(w_i,positive)}=0$$

- Since the probability of a P(fantastic|positive)=0, the whole probability of the document being positive becomes zero

$$P(positive|w_{1,\ldots},w_n)\propto P(positive)\prod_i P(w_i|positive)=0$$

# Laplace (add-1) smoothing

- Add one to every word-category pair, to avoid zeros

$$\hat{P}(w_i|c_j) = \frac{Count(w_i, c_j)+1}{\sum_{w \in V} (Count(w_i, c_j)+1)} = \frac{Count(w_i, c_j)+1}{(\sum_{w \in V} Count(w_i, c_j))+|V|}$$

Number of words in the model's vocabulary

# Summary: training Naive Bayes model

Given: documents $d_i$, each belonging to a class $c_i$

- Extract vocabulary $V$ (e.g., all words)
- Calculate $P(c_j)$ terms: for each $c_j$ do:

$$P(c_j) = \frac{N_{d \in c_j}}{N_d}$$

- Calculate $P(w_k|c_j)$ terms

  $N_j \leftarrow$ # of tokens in the whole subcorpus of $c_j$

  - For each word $w_k$ in vocabulary:

    $n_{kj} \leftarrow$ # of occurrences of $w_k$ in the whole subcorpus of $c_j$

$$P(w_k|c_j) = \frac{n_{kj} + \alpha}{N_j + \alpha|Vocabulary|}$$

# Example

- Training corpus
  - “jalgpall eelarve värav” - S (sport)
  - “korvpall võit” - S
  - “jalgpall võit raha” - S
  - “eelarve defitsiit raha” - M (majandus)
- Test document
  - “eelarve raha võit” - ?

$$P(c_j) = \frac{Count(c_j)}{N_{doc}} \qquad P(w_i|c_j) = \frac{Count(w_i, c_j) + 1}{\sum_w Count(w_i, c_j) + |V|}$$

$$\begin{aligned} c_{max} &= argmax_c P(d|c) \times P(c) \\ &= argmax_c P(w_1 w_2 .. w_n|c) \times P(c) \end{aligned}$$

$$\begin{aligned} P(w_1, w_2, ..., w_n|c) &= P(w_1|c) \times P(w_2|c) \times ... \times P(w_2|c) \\ &= \prod_w P(w_i|c) \end{aligned}$$

# Solution

$$
\begin{aligned}
P(c = S) &= 3/4 = 0.75 \\
P(c = M) &= 1/4 = 0.25 \\
P(eelarve|S) &= (1+1)/(8+7) = 2/15 \\
P(raha|S) &= (1+1)/(8+7) = 2/15 \\
P(voit|S) &= (2+1)/(8+7) = 3/15 \\
P(eelarve|S) &= (1+1)/(3+7) = 0.2 \\
P(raha|S) &= (1+1)/(3+7) = 0.2 \\
P(voit|S) &= (0+1)/(3+7) = 0.1 \\
P(S|D_{test}) &\propto 0.75 * 2/15 * 2/15 * 3/15 = 0.00267 \\
P(M|D_{test}) &\propto 0.25 * 0.2 * 0.2 * 0.1 = 0.001
\end{aligned}
$$

# Naive Bayes and features

- Using the bag-of-words as a document representation is **feature extraction**
- We don’t have to use word occurrence features
  - e.g., for Estonian, using **lemmas** (*spordi* → *sport*) might be useful
- But features can be “anything” that can be derived from the document and that could be helpful for classification
- Other features possibly used by a spam filter:
  - Is *Subject* written in ALL CAPITAL LETTERS?
  - Does the e-mail originate from a known spam host?
  - Is the sender in recipients address book?
  - Does the e-mail contain a link to a known malicious web site?

# Feature extraction exercise

- You decide to train a classifier to predict whether you will '**like**' a Facebook post (by your friends)
- What features will you try?

# Feature extraction exercise

- Example features:
  - Number of photos
  - Number on existing comments
  - Number of likes
  - Number of likes/comments by your friends
  - Are there people on photos?
  - Number of emoticons in the post
  - Language of the post
  -

# More about features

- Main features do not have to be words
- Word n-grams (bigrams, trigrams) often work better and can capture more complex meanings (e.g. “not like”)
- Often, character n-grams work even better
  - For example, for language identification

# Multi-class classification

- What if a document can belong to many categories?
- Solution: train a one-against-all classifier for all classes
- Given a test doc:
  - Evaluate it for membership in each class using each classifier

Tennise Austraalia lahtiste viimases tänases üksikmängus oli võidukas tiitlikaitsjast šveitslane Roger Federer (ATP 2.), kes alistas sakslase Jan-Lennard Struffi (ATP 55.) 6:4, 6:4, 7:6 (7:4).

Järgmises ringis ootab teda prantslane Richard Gasquet (ATP 31.). Nad on kohtunud 18 korda ja vaid kahel juhul on edu saatnud Gasquet'd.

Viimati sai ta Federerist jagu 2011. aastal Roomas, seejärel on kaheksa korda järjest võitnud šveitslane.

Lisaks mullusele triumfile on Federer võitnud Austraalia lahtised ka 2004., 2006., 2007. ja 2010. aastal.

Toimetaja: Siim Boikov

austraalia lahtised roger federer

# Quality metrics for classification

- How good is our classifier?
- Test on data that was not used for training
- Compare predicted classes to the real (human-annotated) classes
- **Accuracy**: how many documents in the test set were classified correctly?

$$A = \frac{N_{correct}}{N_{total}}$$

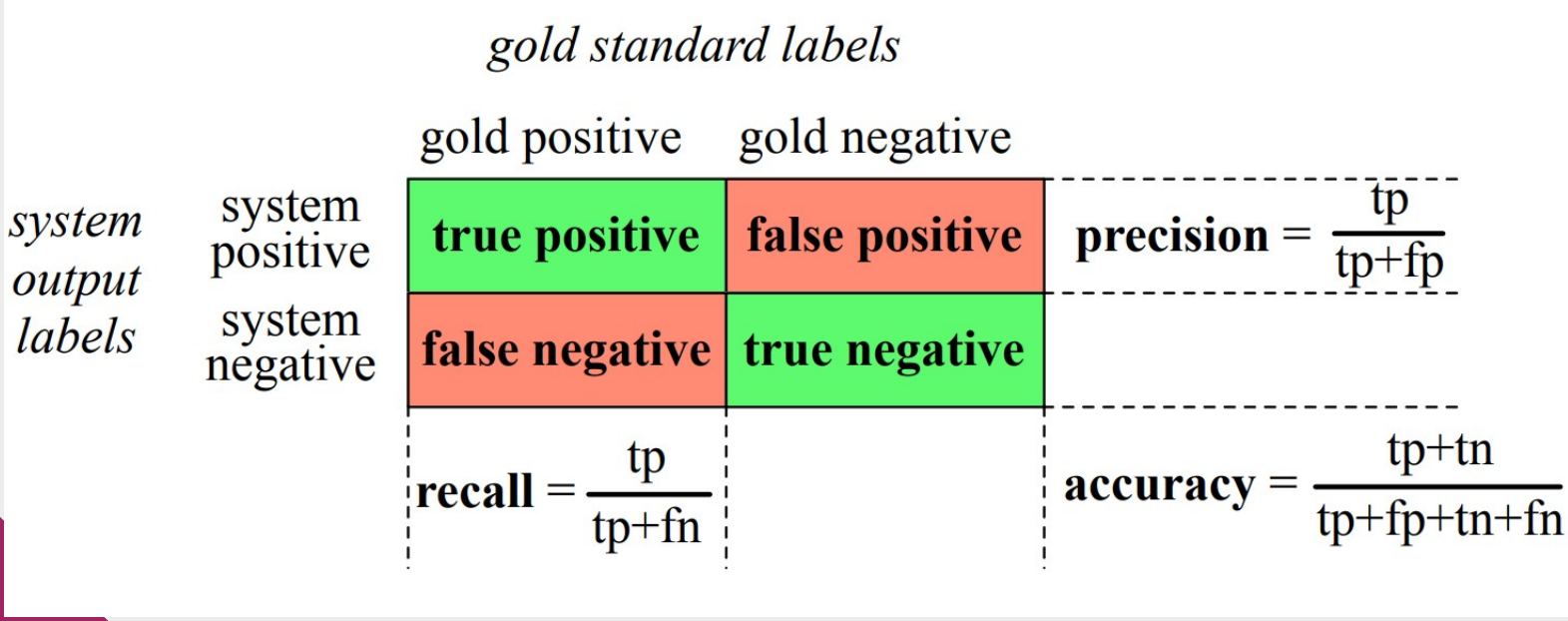
# Problem with accuracy

- Accuracy metric doesn’t work well for inbalanced classes
- For example, if test corpus contains
  - 990 documents about economy
  - 10 documents about sport
- If the classifier assigns “economy” to all documents, the accuracy is 990/1000 = 99%
  - 99% sounds good, but in reality the classifier is not useful at all!
- Often, error rate (ratio of errors in all data) is used instead of accuracy: 10/1000= 1%

# Precision and recall

Precision and recall are metrics for binary classifiers, where we are mostly interested in finding items of a certain class (e.g., Facebook posts that we like)

- Precision: % of detected items that are in fact positive
- Recall: % of positive items that are detected by the system



# $F_1$-measure

- Precision and recall give different views to the classifiers performance:
  - Try to increase precision if you are really bothered about false alarms (and you can live with some items being missed)
  - Try to increase recall if you are care about really finding most positive items, even when many of the found items are negative (e.g., cancer test)
- Combined metrics: $F_1$ measure

$$F_1 = \frac{2 * Precison * Recall}{Precision + Recall}$$

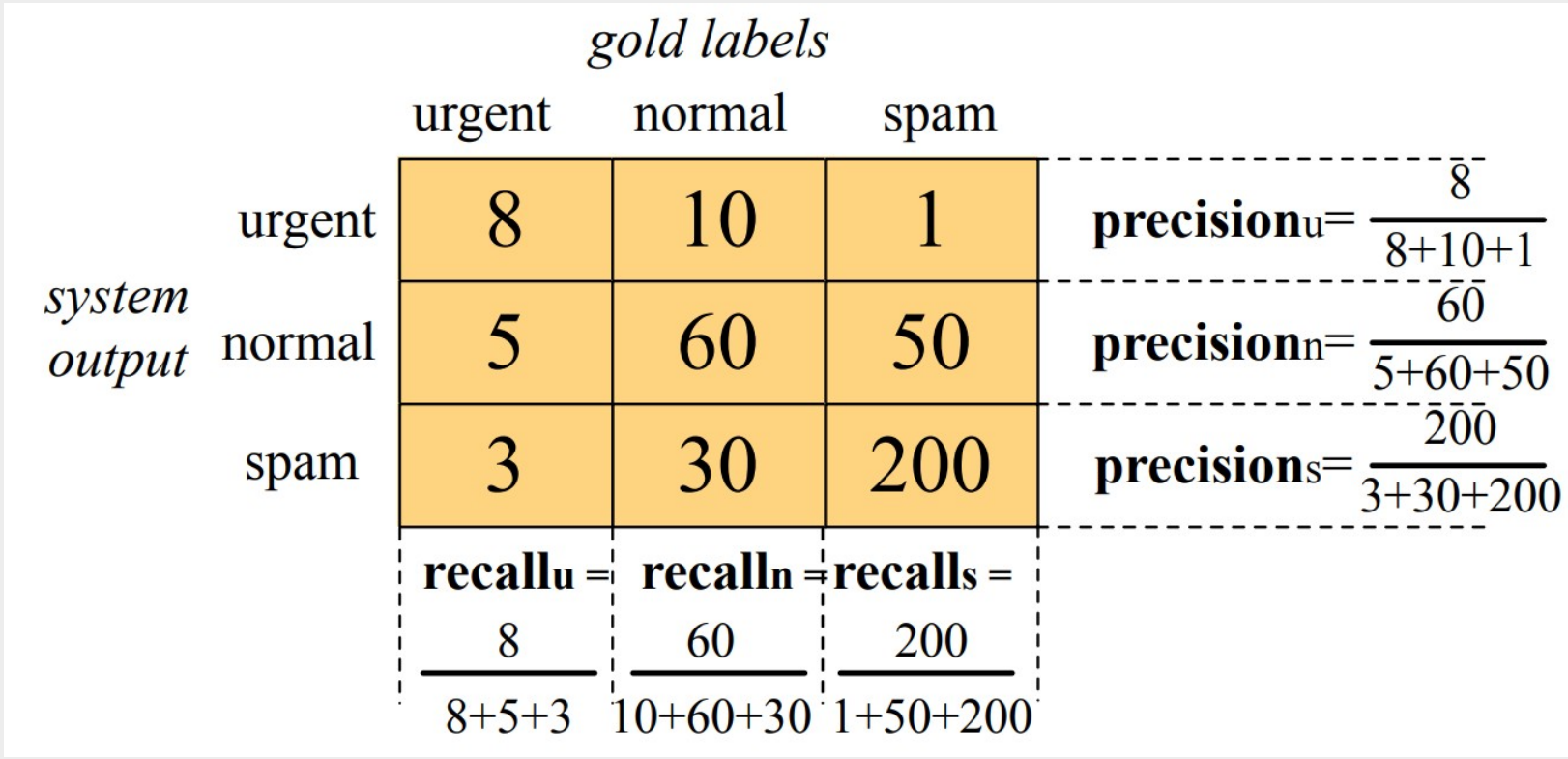
# Example

- Example: 100 doping tests, 10 are really positive
- Automatic doping test analyzer classified 7 positive samples as positive, 5 negative samples as positive, and 3 positive tests as negative.
- TP=7 FP=5 TN=85 FN=3
- *Accuracy*: 92/100 = 92%
- *Precision*: 7/(7+5) =58%
- *Recall*: 7/(7+3) =70%
- *F1-measure*: 2*0.58*0.7/(0.58+0.7)=0.63

# Micro-average and macro-average

- If we have more than one class, how do we combine multiple performance measures into one quantity?
- **Macro-averaging**: Compute performance for each class, then average.
- **Micro-averaging**: Collect decisions for all classes, compute contingency table, evaluate.

# Micro-average and macro-average: example



# Example, continued

**Class 1: Urgent**

| | true urgent | true not |
|---|---|---|
| system urgent | 8 | 11 |
| system not | 8 | 340 |

$$\text{precision} = \frac{8}{8+11} = .42$$

**Class 2: Normal**

| | true normal | true not |
|---|---|---|
| system normal | 60 | 55 |
| system not | 40 | 212 |

$$\text{precision} = \frac{60}{60+55} = .52$$

**Class 3: Spam**

| | true spam | true not |
|---|---|---|
| system spam | 200 | 33 |
| system not | 51 | 83 |

$$\text{precision} = \frac{200}{200+33} = .86$$

**Pooled**

| | true yes | true no |
|---|---|---|
| system yes | 268 | 99 |
| system no | 99 | 635 |

$$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$

# Training/development/test

- When training a classifier, split the data into training, development (evaluation) and test set
- Typically using a ratio 80% 10% 10%
- Training data if used for training the model
- Development data is used for optimizing the architecture and meta-parameters of the model
  - Which features to use?
  - What kind of smoothing to use?
- Test data is used for estimating the true performance of the classifier
  - Using development data for this is too optimistic, as the model’s hyperparameters have been already optimized on it

# Robustness

- It’s important that training data represents different domains, in order for the model to be robust
- For example, if you train a language identification model on the Wikipedia dumps of different languages, it’s not necessarily very good on Twitter
- Better: include news (web portals), books, social media (might have to label it yourself)

# Avoiding harms

- It’s important to know that a machine-learning based classification system could be biased
  - One research showed that most sentiment systems assigned lower sentiment and more negative emotion to sentences with African American names (like *Shaniqua* as opposed to *Stephanie*)
  - Toxicity detection (e.g. in social media comments): some widely used toxicity classifiers incorrectly flag as being toxic sentences that are non-toxic but simply mention minority identities like women, blind people or gay people
- Why?
  - Bias in training data
  - Bias in other resources (e.g. sentiment lexicons)
  - Bias in labels (due to biases in the human labelers)
- Thus, it is important to test the classifiers across different demographic or other groups and environmental situations
- Also, carefully consider the training data resources

# Classifying words

# Why classify words?

- Classifying words into categories is useful for many NLP tasks:
  - Part-of-speech tagging: classify words according their class (noun, verb, adjective)
  - Find names (persons, locations, companies, etc)
  - Find time expressions
- More generally, we want to tag (classify) each item (word) in a sequence (sentence)
  - Machine learning: sequence tagging problem

# Part-of-speech tagging

- Task: assign a syntactic category for each word

```
Mrs. Shaefer never got around to joining
NNP  NNP     RB    VBD RP      TO VBG
```

- Useful for downstream text processing tasks
  - Speech synthesis (*record, lead*)
  - Lemmatization (saw → see, saw → saw)
  - Parsing

# English POS tags

| | | |
|---|---|---|
| **CC** | conjunction, coordinating | and both but either or |
| **CD** | numeral, cardinal | mid-1890 nine-thirty 0.5 one |
| **DT** | determiner | a all an every no that the |
| **EX** | existential there | there |
| **FW** | foreign word | gemeinschaft hund ich jeux |
| **IN** | preposition or conjunction, subordinating | among whether out on by if |
| **JJ** | adjective or numeral, ordinal | third ill-mannered regrettable |
| **JJR** | adjective, comparative | braver cheaper taller |
| **JJS** | adjective, superlative | bravest cheapest tallest |
| **MD** | modal auxiliary | can may might will would |
| **NN** | noun, common, singular or mass | cabbage thermostat investment subhumanity |
| **NNP** | noun, proper, singular | Motown Cougar Yvette Liverpool |
| **NNPS** | noun, proper, plural | Americans Materials States |
| **NNS** | noun, common, plural | undergraduates bric-a-brac averages |
| **POS** | genitive marker | ' 's |
| **PRP** | pronoun, personal | hers himself it we them |
| **PRP$** | pronoun, possessive | her his mine my our ours their thy your |
| **RB** | adverb | occasionally maddeningly adventurously |
| **RBR** | adverb, comparative | further gloomier heavier less-perfectly |
| **RBS** | adverb, superlative | best biggest nearest worst |
| **RP** | particle | aboard away back by on open through |
| **TO** | "to" as preposition or infinitive marker | to |
| **UH** | interjection | huh howdy uh whammo shucks heck |
| **VB** | verb, base form | ask bring fire see take |
| **VBD** | verb, past tense | pleaded swiped registered saw |
| **VBG** | verb, present participle or gerund | stirring focusing approaching erasing |
| **VBN** | verb, past participle | dilapidated imitated reunifed unsettled |
| **VBP** | verb, present tense, not 3rd person singular | twist appear comprise mold postpone |
| **VBZ** | verb, present tense, 3rd person singular | bases reconstructs marks uses |
| **WDT** | WH-determiner | that what whatever which whichever |
| **WP** | WH-pronoun | that what whatever which who whom |
| **WP$** | WH-pronoun, possessive | whose |
| **WRB** | Wh-adverb | however whenever where why |

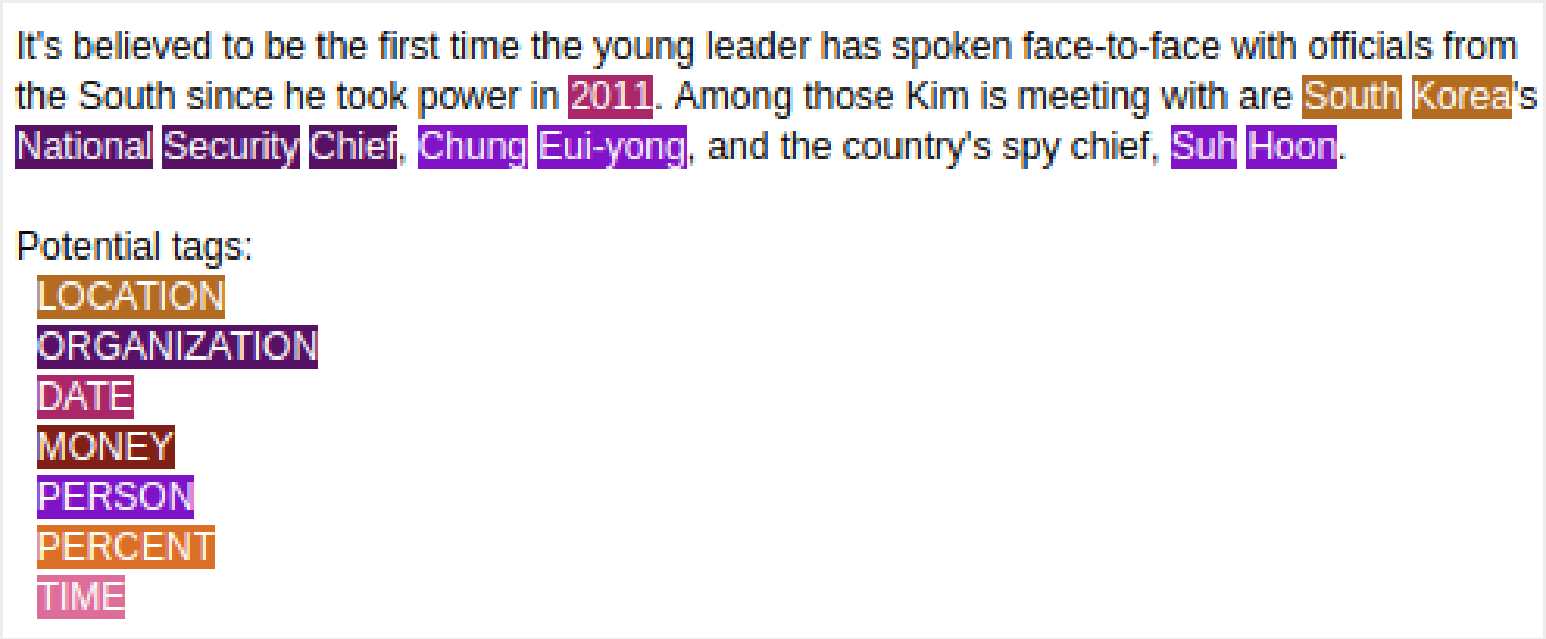
# Part-of-speech ambiguity

- Word can have multiple parts-of-speech:

| Fed | raises | interest | rates | 0.5 | percent |
|---|---|---|---|---|---|
| VBD | NNS | VB | VBZ | CD | NN |
| VBN | VBZ | VBP | NNS | | |
| NNP | | NN | | | |

# Named Entity Recognition (NER)

- Find and classify names in text, for example:



# Named Entity Recognition

- Applications:
  - Document indexing, linking
  - Sentiment can be attributed to companies and products
  - Information extraction: find associations between names
  - Question answering: answers to natural language questions are often named entities (e.g. *Who is the prime minister of Estonia? What is the largest country in Africa?*)

# Word classification task

- POS tagging

```
Mrs.      NNP
Shaefer   NNP
never     RB
got       VBD
around    RP
to        TO
joining   VBG
:         :
```

- Named Entity Recognition

```
Foreign    ORG
Ministry   ORG
spokesman  O
Shen       PER
Guofang    PER
told       O
Reuters    ORG
:          :
```

# Features for word classification

- Words
  - Current word itself (what class is assigned to this word in training data?)
  - Previous/next word (context)
- Other inferred linguistic classification
  - E.g. use inferred POS tags when doing NER
- Word features
  - Prefixes, suffixes, other substrings (e.g. surprising**ly** → RB)
  - Word shape (Is word all lowercase? Is word in Title-case? Is word in UPPERCASE? Is it all-digits?)
- Gazzetteers: dictionaries from external sources (e.g., for NER: make a collection of all company names in Estonia, and use a feature: Is the word in the collection?)
- Handcrafted features, looking at the word context (e.g., for NER: is the current word uppercase and followed within 3 words by *Co.*, *Inc.*, or *LLC*?)
- **Label (word class) context**
  - Class of the previous (and perhaps the next) word
  - Available during training, but how to perform decoding?

# Maximum entropy models

- The task of classifying words is similar to document classification
- However, here the features are typically even more correlated than in document classification
  - E.g. `word=surpringly, suffix1=y, suffix2=ly, suffix3=gly, prefix2=su`
- Therefore, Naive Bayes doesn't work well for word classification
- **Maximum entropy models** *(aka* multinomial logistic regression models*)* are popular machine learning models that allow feature dependence

# Maximum entropy classifiers

- Naive Bayes is a *generative* model: in order to estimate *P(y|x)*, we evaluate *P(x|y)* – the probability that the class *y* *generated* the observation *x*
- Maximum entropy classifer is a *discriminative* model: it estimates *P(y|x)* directly:

$$\hat{y} = argmax_y P(y|x)$$

# Linear classifier

- As Naive Bayes, MaxEnt is *linear classifier*
- Linear classifiers:
  - Extract some set of features from input
  - Multiply each active feature with its weight
  - Add up the weighted features
  - Apply some function to this combination

# Linear classifier

- The simplest linear classier is potentially like this:

$$P(y|x) = \sum_{i=1}^{N} w_i f_i(x, y)$$

DOESN'T WORK!

- However, the above doesn't produce a legal probability distribution
  - $w_i f_i(x, y)$ could be negative!
  - The sum over all classes doesn't sum to one!

# ‘Fixing’ the simple linear classifier

- The maximum entropy classifier is a simple linear classifier, “fixed” using two tricks
  - First, we exponentiate the weighted sum so that it’s always positive:

$$\exp \sum_{i=1}^{N} w_i f_i(x, y)$$

This is positive
But not between 0 and 1

  - Second, we normalize this expression (using the sum over all classes), so that the sum will be exactly 1, resulting in legal probability distribution

$$P(y|x) = \frac{\exp \sum_{i=1}^{N} w_i f_i(x, y)}{\sum_{y' \in Y} \exp \sum_{j=1}^{N} w_j f_j(x, y')}$$

This is positive and between 0 and 1
Also, the sum over all classes
Is exactly 1

# Features in MaxEnt classifier

- The features in the MaxEnt classifer are slightly different from the features in other machine learning model
- And a bit unintuitive
- It's actually better to call them *indicator functions*
- The indicator functions are typically functions of both a traditional features and a class
- That is, they link aspects of the observation with the class that we want to predict

- $f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$
- $f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$
- $f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$

# Indicator functions and weights in MaxEnt model

- $f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$
- $f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$
- $f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$


- Model assigns each indicator function a weight:
  - A positive weight “votes” that this feature-class combination is likely correct
  - A negative weight “votes” that this feature-class combination is likely incorrect
- Weights are learned from training data (we’ll see how)

# Classification example

- Task: document sentiment analysis
- Classes: -, +
- Features:

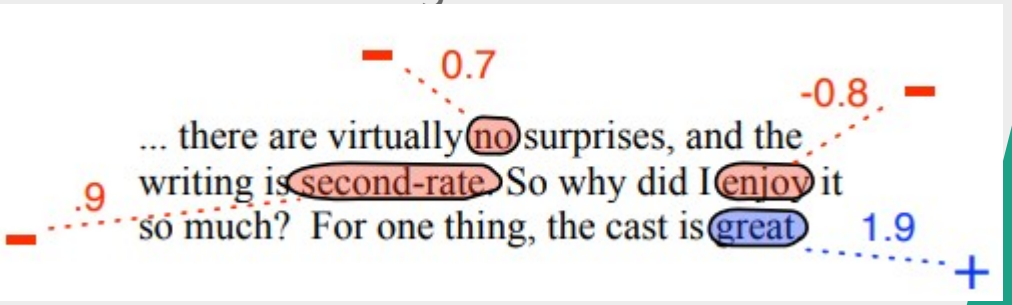
$$f_1(c,x) = \begin{cases} 1 & \text{if "great"} \in x \ \& \ c = + \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(c,x) = \begin{cases} 1 & \text{if "second-rate"} \in x \ \& \ c = - \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(c,x) = \begin{cases} 1 & \text{if "no"} \in x \ \& \ c = - \\ 0 & \text{otherwise} \end{cases}$$

$$f_4(c,x) = \begin{cases} 1 & \text{if "enjoy"} \in x \ \& \ c = - \\ 0 & \text{otherwise} \end{cases}$$

- Learned weights:
  - $w_1$=1.9
  - $w_2$=0.9
  - $w_3$=0.7
  - $w_4$=-0.8
- Find the class probabilities for the following document:




- Formula:

$$P(y|x) = \frac{\exp \sum_{i=1}^{N} w_i f_i(x,y)}{\sum_{y' \in Y} \exp \sum_{j=1}^{N} w_j f_j(x,y')}$$

# Classification example

- Task: document sentiment analysis
- Classes: -, +
- Features:

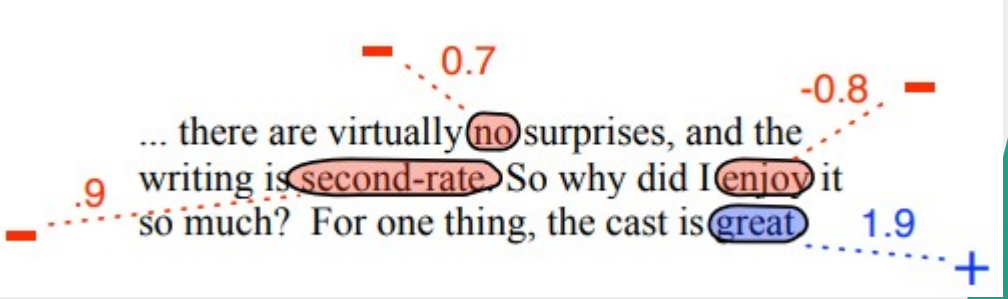
$$f_1(c,x) = \begin{cases} 1 & \text{if "great"} \in x \ \& \ c = + \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(c,x) = \begin{cases} 1 & \text{if "second-rate"} \in x \ \& \ c = - \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(c,x) = \begin{cases} 1 & \text{if "no"} \in x \ \& \ c = - \\ 0 & \text{otherwise} \end{cases}$$

$$f_4(c,x) = \begin{cases} 1 & \text{if "enjoy"} \in x \ \& \ c = - \\ 0 & \text{otherwise} \end{cases}$$

- Learned weights:
  - $w_1$=1.9
  - $w_2$=0.9
  - $w_3$=0.7
  - $w_4$=-0.8
- Find the class probabilities for the following document:




- Solution

$$P(y|x) = \frac{\exp \sum_{i=1}^{N} w_i f_i(x,y)}{\sum_{y' \in Y} \exp \sum_{j=1}^{N} w_j f_j(x,y')}$$

$$P(+|x) = \frac{e^{1.9}}{e^{1.9} + e^{.9+.7-.8}} = .82$$

$$P(-|x) = \frac{e^{.9+.7-.8}}{e^{1.9} + e^{.9+.7-.8}} = .18$$

# Training MaxEnt model

- Intuition: choose weights for indicator functions so that the classes observed in training data will be more likely
- That is: conditional maximum likelihood estimation
- That means, we choose weights that maximize the (log) probability of labels $y^{(j)}$ in the training data, given the observations $x^{(j)}$:

$$\hat{w} = argmax_w \sum_j \log P\left(y^{(j)} | x^{(j)}\right) = argmax_w \sum_j \log \frac{\exp \sum_{i=1}^{N} w_i f_i\left(y^{(j)}, x^{(j)}\right)}{\sum_{y' \in Y} \exp \sum_{k=1}^{N} w_k f_k\left(y', x^{(j)}\right)}$$

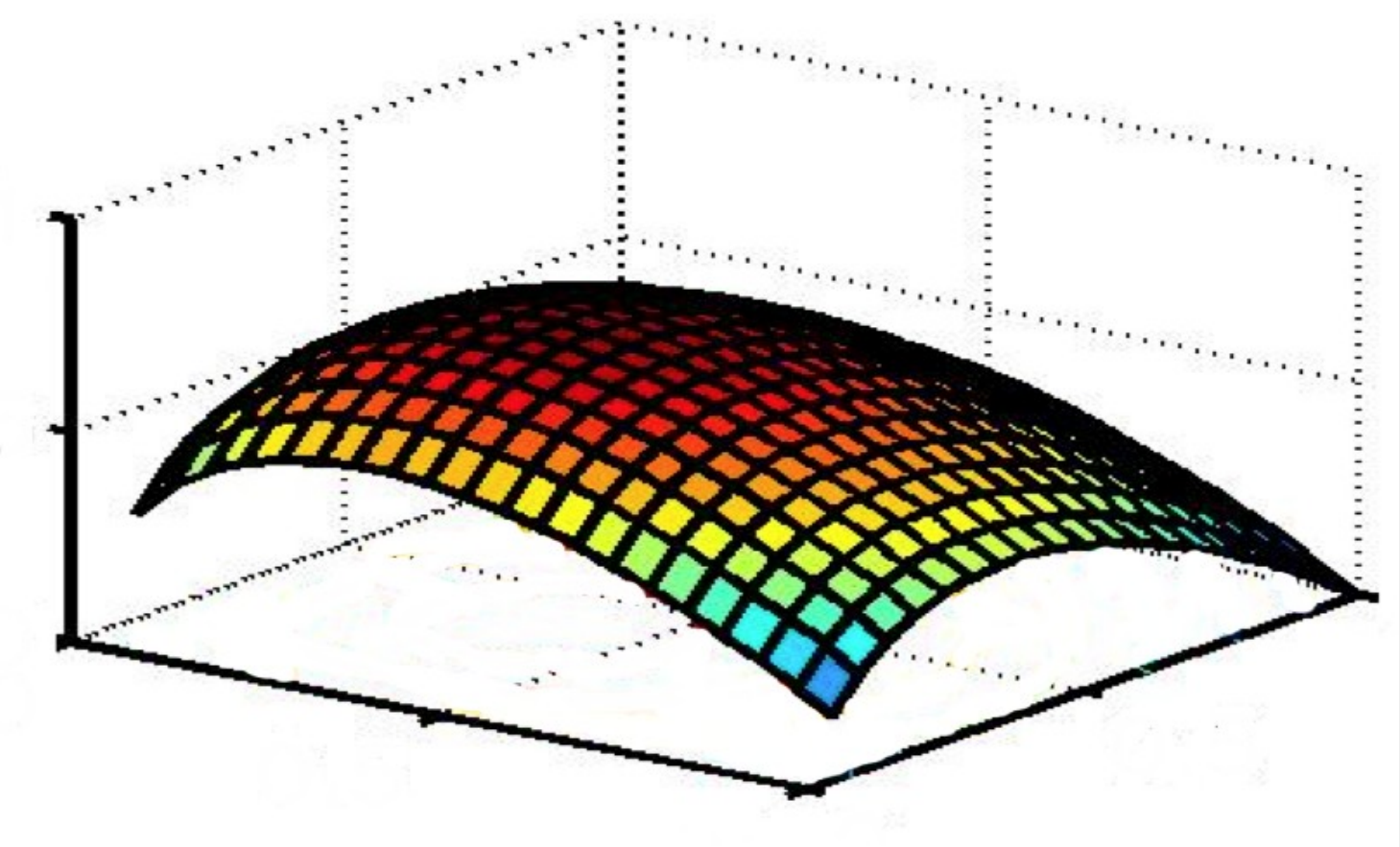
# Training MaxEnt model

- The function that we want to maximize is the objective function:

$$L(w) = \sum_j \log P\left(y^{(j)} | x^{(j)}\right)$$

- This is called **cross-entropy loss**
- In Naive Bayes, we found the parameters of the model analytically, by just counting the items in the training data
- The parameters of the MaxEnt models cannot be found analytically, instead we use “hill-climbing” methods like stochastic gradient ascent
- Such gradient ascent methods start with a zero weight vector and move in the direction of the gradient, *L’(w)* the partial derivative of the objective function *L(w)* with respect to the weights
- Neural networks are trained very similarly

# A likelihood surface



# Features

- Features correspond to word/context attributes that are distinctive enough to deserve model parameters
  - E.g. word itself, word suffix, suffix of previous word, etc
- Features are often added during development phase to target errors
  - Think of useful word/class combinations
  - Also, think of “bad combinations” that should get negative weights. E.g.: “word contains digit” combined with *Personal name* should get a useful negative weight in NER
- Usually, we use feature templates that automatically generate all features that occur in training data, according to some template:
  - word[i], word[i-1], word[i+1]
  - suffix1(word[i]), suffix2(word[i]), suffix3(word[i])…
- MaxEnt models do not automatically model feature combinations
- Therefore, it’s often beneficial to include feature conjunctions, e.g.:
  - word[i-1]+word[i], word[i]+word[i+1]
  - word[i]+suffix2(word[i-2])
- But be careful: too many combinations generate too much features → **overfitting**

# Regularization

- Usually it's not good if we learn weights that make the model perfectly match the training data
- The weights try to match the data too perfectly
- Often, the features start to model noisy factors in the training data that just accidentally correlate with the class
- This is called **overfitting**
- To avoid overfitting, we often add a regularization term to the objective function:

$$\hat{w} = argmax_w = \sum_j \log P\left(y^{(j)} | x^{(j)}\right) - \alpha R(w)$$

# L2 regularization

- The regularization term *R(w)* penalizes **large** weights
- Thus a setting of the weights that matches the training data perfectly, but uses weights with high values, will be penalized more than than a setting that matches the data less well, but does so using smaller weights
- One of the most common regularization methods is L2 regularization

$$\hat{w} = argmax_w = \sum_j \log P\left(y^{(j)} | x^{(j)}\right) - \alpha R(w)$$

$$R(w) = \sum_{j=1}^{N} w_j^2$$

# Why the surrounding classes improve performance

- Often, the classes of words can be inferred better if the classifier ‘sees’ the classes of the previous/next words:
  - POS:

    ```
    DT  NNS  VBD     VBN
    The flaws remained undetected
    ```

  - NER:

    ```
    O O   LOC   LOC
    I saw Mount Washington

    O O   PER    PER
    I saw George Washington
    ```

- However, we do not know the true classes of surrounding words at test time
- Solutions:
  - Don’t use classes of surrounding words (actually works pretty good)
  - Use classes of only previous words, and rely on the inferred classes during test time
  - Use a sequence-aware model (Conditional Random Fields)

End