Natural Language and Speech Processing

Lecture 6: Classifying words

Tanel Alumäe

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

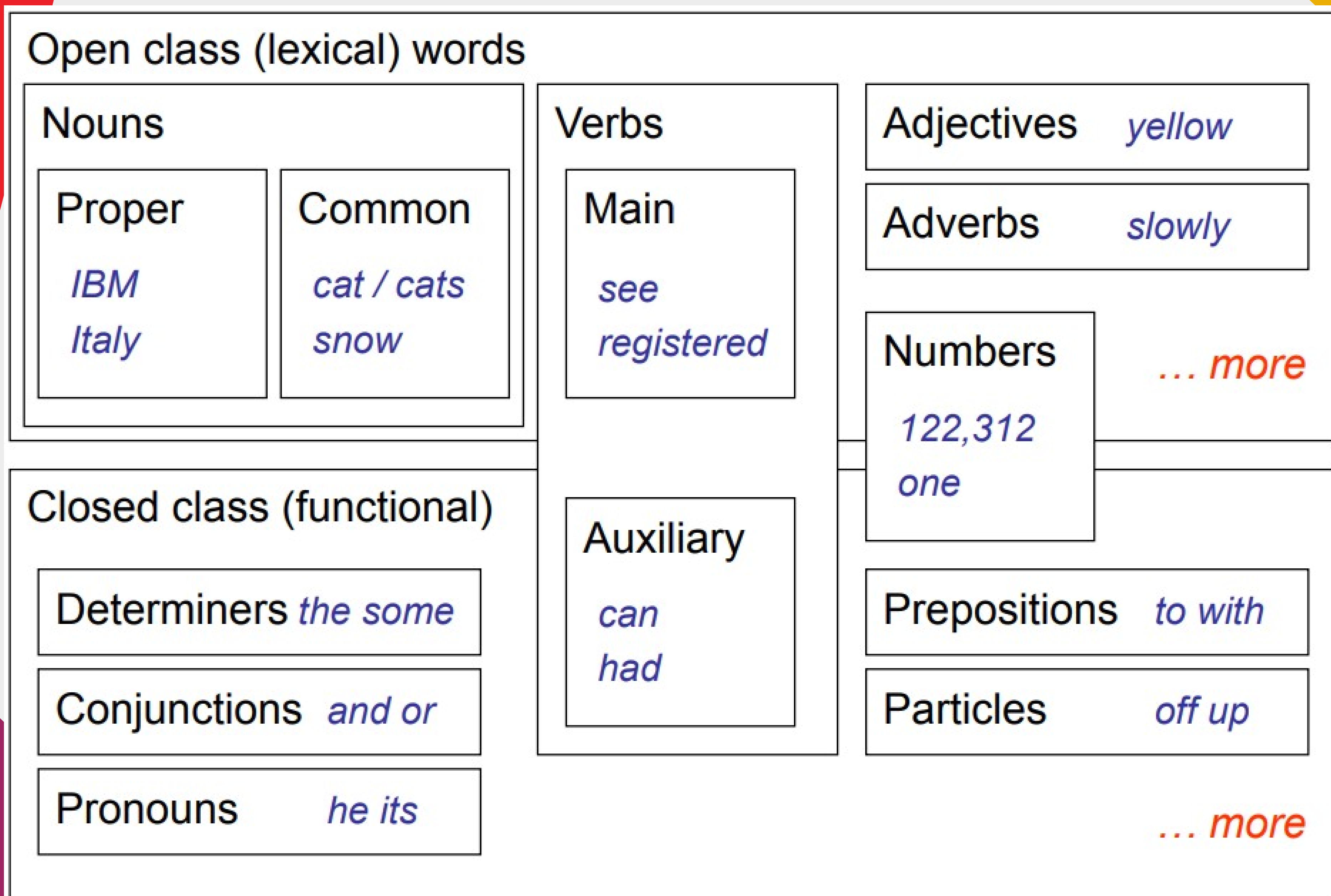
- Task: assign a syntactic category for each word

```
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



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:

It's believed to be the first time the young leader has spoken face-to-face with officials from the South since he took power in 2011. Among those Kim is meeting with are South Korea's National Security Chief, Chung Eui-yong, and the country's spy chief, Suh Hoon.

Potential tags:

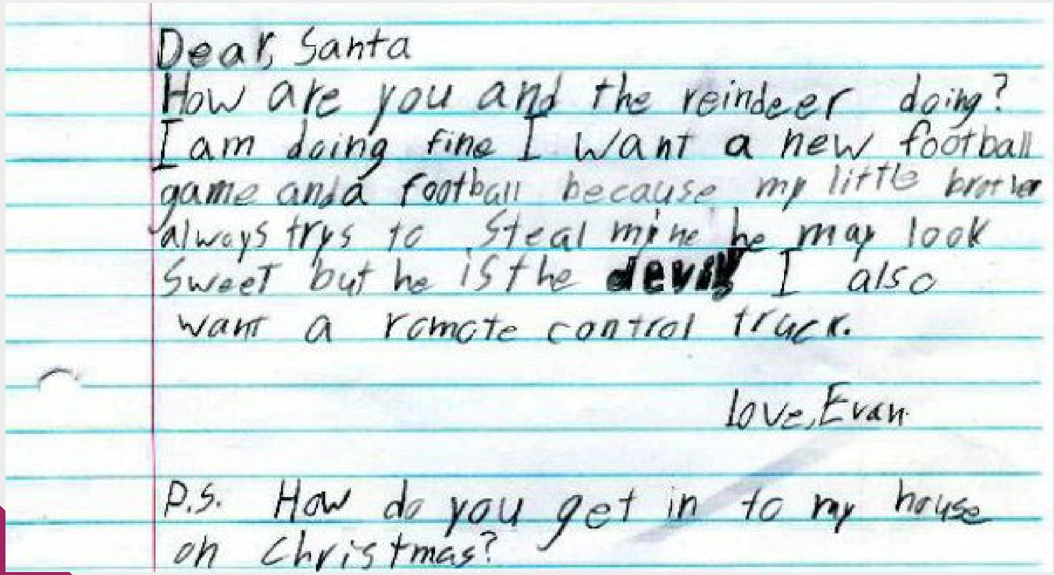
LOCATION
ORGANIZATION
DATE
MONEY
PERSON
PERCENT
TIME

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?*)

Other uses of word classification

- Extract specific noun phrases:
  - *Dear Santa! How are you and the reindeer doing? I am doing fine I want a new football game and a football because my little brother always tries to steal mine he may look sweet but he is the devil. I also want a remote control truck. Love, Evan. ...*



Dear Santa
How are you and the reindeer doing?
I am doing fine I want a new football
game and a football because my little brother
always trys to Steal mine he may look
Sweet but he is the devil I also
want a remote control truck.

love, Evan

P.S. How do you get in to my house
on Christmas?

→

Football game
Football
Remote control truck

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

- Naibe 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 classifer, “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:

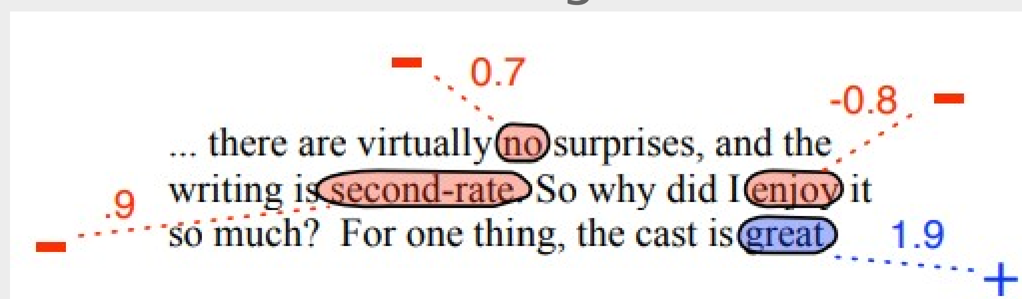
$$f_1(c,x) = \begin{cases} 1 & \text{if “great”} \in x \ \& \ c = + \\ 0 & \text{otherwise} \end{cases}$$

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

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

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

- 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:

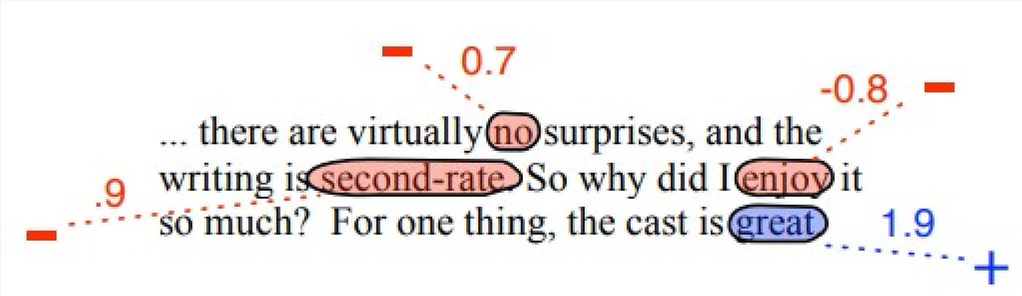
$$f_1(c,x) = \begin{cases} 1 & \text{if "great"} \in x \;\&\; c = + \\ 0 & \text{otherwise} \end{cases}$$

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

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

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

- 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

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

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

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

Exercise

3 class decision: LOCATION, PERSON, or DRUG; 3 features as below, what are:

- P(PERSON | *by Goéric*) =
- P(LOCATION | *by Goéric*) =
- P(DRUG | *by Goéric*) =
-
- 1.8 $f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$
- -0.6 $f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$
- 0.3 $f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$

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

PERSON *by Goéric*

LOCATION *by Goéric*

DRUG *by Goéric*

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 $x^{(j)}$:

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

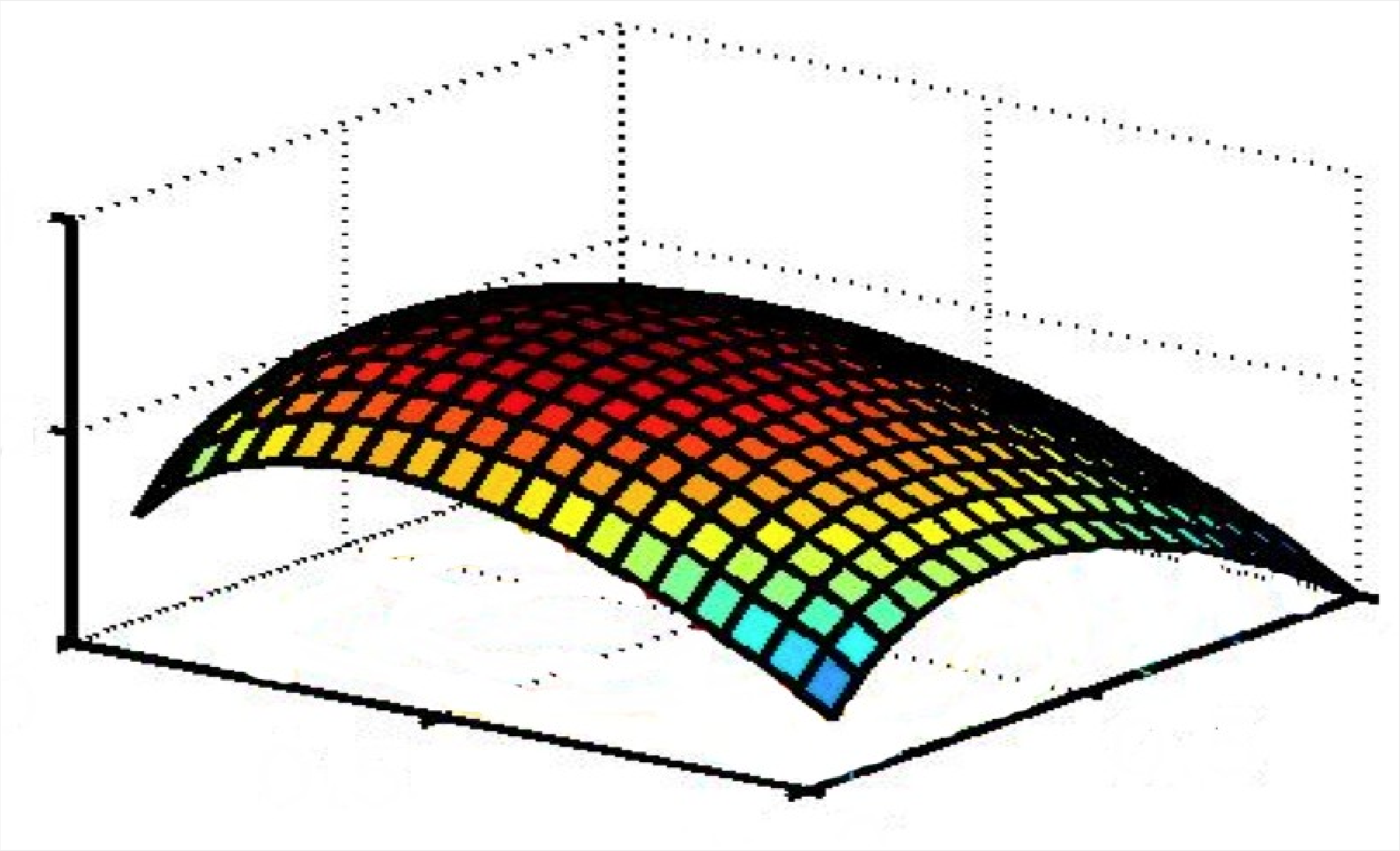
Training MaxEnt model

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

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

- 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 or L-BFGS
- 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:

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

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

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

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

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
    ```

Classifying sequences of words

- Training sequence models (that use the classes of previous words as features) is relatively straightforward: classes of previous words are given in training data
- But how to decode? The classes of previous words are not known during decoding!
- Solutions:
  - Greedy decoding,
  - Greedy decoding, and use model-inferred classes during training
  - Beam search
  - Use a model like *Conditional Random Field (CRF)* that models the sequence of classes jointly

Classifying sequences of words

- Training sequence models (that use the classes of previous words as features) is relatively straightforward: classes of previous words are given in training data
- But how to decode? The classes of previous words are not known during decoding!
- Solutions:
  - Greedy decoding,
  - Greedy decoding, and use model-inferred classes during training
  - Beam search
  - Use a model like *Conditional Random Field (CRF)* that models the sequence of classes jointly

Greedy decoding

- In greedy decoding, we choose the best class for each word, step-by-step
- When a feature needs the class of the previous word, we simply use the predicted class
- Problems:
  - Cannot use classes of future words
  - Errors accumulate: if we make a mistake in classifying a previous word, the next word is also likely to be classified incorrectly because of the hard (wrong) decision
    - ```
      Correct:   O O    0  LOC   LOC
      Predicted: 0 0    0  PER   ???
      Words:     I live on Grace Road
      ```
      ```

Greedy decoding, improved

- The problem with greedy decoding:
  - The model relies too much on the classes of previous words (because they are given in training data)
- A bit hacky solution: use model-predicted classes for previous words also during training
- This makes the training and decoding data more similar
- The model learns that the classes of previous words are not too reliable
- Works well in many cases, and it’s simple to implement

Beam search

- Beam search: maintain *N* best hypotheses during decoding
- Return the class sequence that gives the **best total score** (probability)
- Still only previous class predictions can be used
- But wrong predictions can be overturned when more evidence is encountered
- Example with *N = 2*
  - Classes: PER (peron), LOC (location), O (other)
  - `I live on Grace Road`
    - ```
      Word 1: I→O    (P=0.95)
              I→ORG  (P=0.05)
      ```
    - ```
      Word 2: I→O   live→O (P=0.95)
              I→ORG live→O (P=0.05)
      ```
    - ```
      Word 3: I→O   live→O on→O (P=0.95)
              I=ORG live→O on→O (P=0.05)
      ```
    - ```
      Word 4: I→O   live→O on→O Grace→PER (P=0.80)
              I→O   live→O on→O Grace→LOC (P=0.20)
      ```
    - ```
      Word 5: I→O   live→O on→O Grace→LOC Road→LOC (P=0.75)
              I→O   live→O on→O Grace→PER Road→PER (P=0.25)
      ```
      ```

Conditional Random Field

- CRF is a MaxEnt model over sequence
- Decoding finds the best class sequence for a given word sequence that gives the best **joint** probability
- Training requires more memory and computation than word-based models