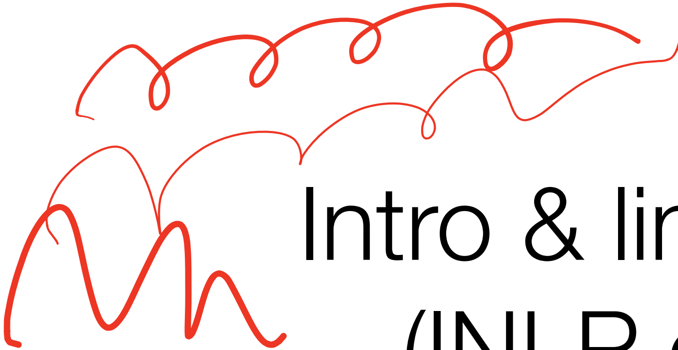


Intro & linear models (INLP ch. 1 & 2)

CS 685, Spring 2021

Advanced Topics in Natural Language Processing

http://brenocon.com/cs685

https://people.cs.umass.edu/~brenocon/cs685_s21/

Brendan O’Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

[Material from Eisenstein (2019)]

Levels of linguistic structure

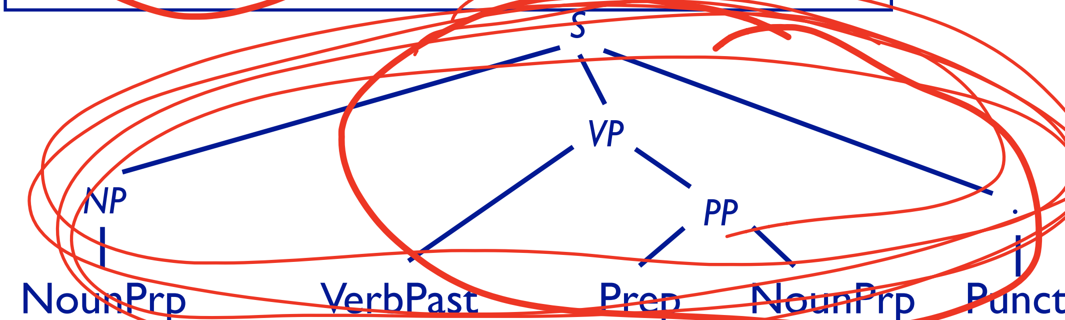
Discourse

Alice saw *Mary* too. **She** talked to **her**.

Semantics

CommunicationEvent(e)
Agent(e, Alice)
Recipient(e, Bob)
SpeakerContext(s)
TemporalBefore(e, s)

Syntax



Words

`Alice` `talked` `to` `Bob` `.`

Morphology

`talk` `-ed`

Characters

`A l i c e   t a l k e d   t o   B o b .`

NLP as linguistic prediction

Scorer (model)

Params

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})}{\operatorname{argmax}} \Psi(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$$

Predicted output

Possible outputs

Input (text)

Output candidate

$y \in \{POS, NEG\}$

$\mathcal{Y}(x) = \{POS, NEG\}$

- x: Text, y: Sentiment label
- x: Text, y: Syntax tree

y: Pronoun conf.

- x: English text, y: Chinese text translation
- x: Book, y: Major characters & their relationships

x: Question/Cmd y: Dialogue response

y: Summary

NLP modeling

- **Pred./Search:** Scorer (model) Params

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})}{\operatorname{argmax}} \Psi(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$$

Predicted output

Possible outputs

Input (text)

Output candidate

- **Learning:** find a good θ from data
(~~we~~ do we need learning at all?)

- **Modeling:** design important ling. phenomena into Ψ
  - Reuse search/learning optimization methods for for many different NLP problems

NLP modeling

- **Pred./Search:** Scorer (model) Params

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})}{\operatorname{argmax}} \Psi(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$$

Predicted output

Possible outputs

Input (text)

Output candidate

- Today: Linear models **Ψ**, BOW **x** & **f**, multiclass **y**
- Models: Naive Bayes and Logistic Regression
  - Learning: (regularized) MLE
- Wednesday: Neural network **Ψ**
- Later: sequential **x**
- Later: structured output **y**

Linear classification models

Y = { NEG, NEU, POS }

- Assume classification problem:
  - Input text **x**
  - Output discrete y∈**Y**, |**Y**|=K
- Scoring function is a dot product of weight vector θ and a vector-valued feature function **f**
  - **x** is a representation of the text. What to use?
  - **f** computes features to score the candidate output. What features to use?

Ψ("I love it", POS)

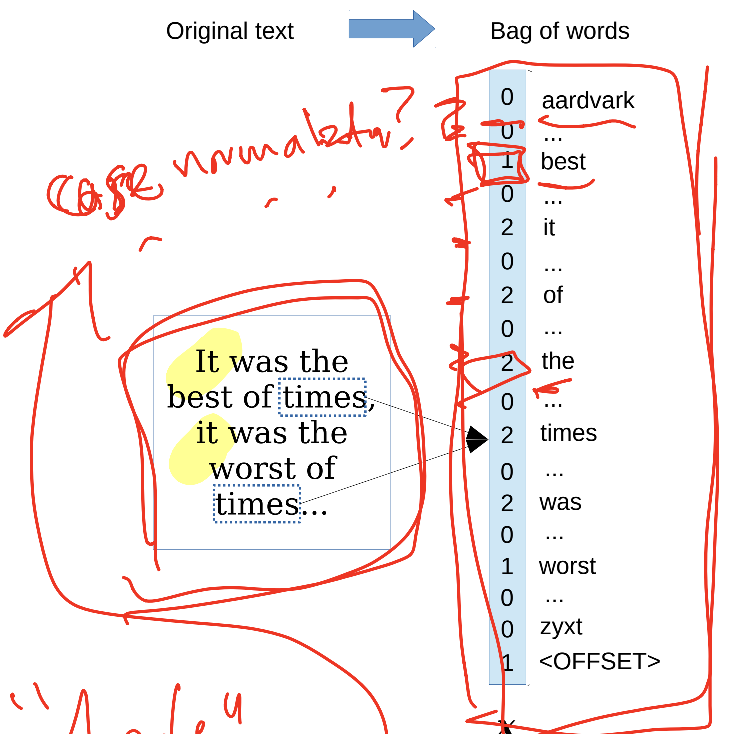
Input (text)

Output candidate

$$\Psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \sum_j \theta_j f_j(\boldsymbol{x}, y)$$

Bag of words representation

"Sparse vector"



- **x** is a vector, representing counts of words in the text
- Vocabulary **V**: set of all word types under consideration
  - Vocab size V = |**V**|
  - Len(x) = V
- BOW ignores order information!! Yet is useful….
- Idea: each word can a weights for each possible output category

tf Topic ?? Sentiment

from Baseline

$$\Psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \sum_j \theta_j f_j(\boldsymbol{x}, y)$$

Bag of words: linear model

- One feature for each word and class pair.

word count

$$f_j(\boldsymbol{x}, y) = \begin{cases} x_{whale}, & \text{if } y = \textsc{Fiction} \\ 0, & \text{otherwise} \end{cases}$$

feat j from $f(x, y)$

- And another for y=NEWS, y=GOSSIP, y=SPORTS (and all other output classes)
- Thus
  - **f**: **X** x **Y** -> $\mathbf{R}^{VK}$
  - θ ϵ $\mathbf{R}^{VK}$

V: vocab size

K: num. classes

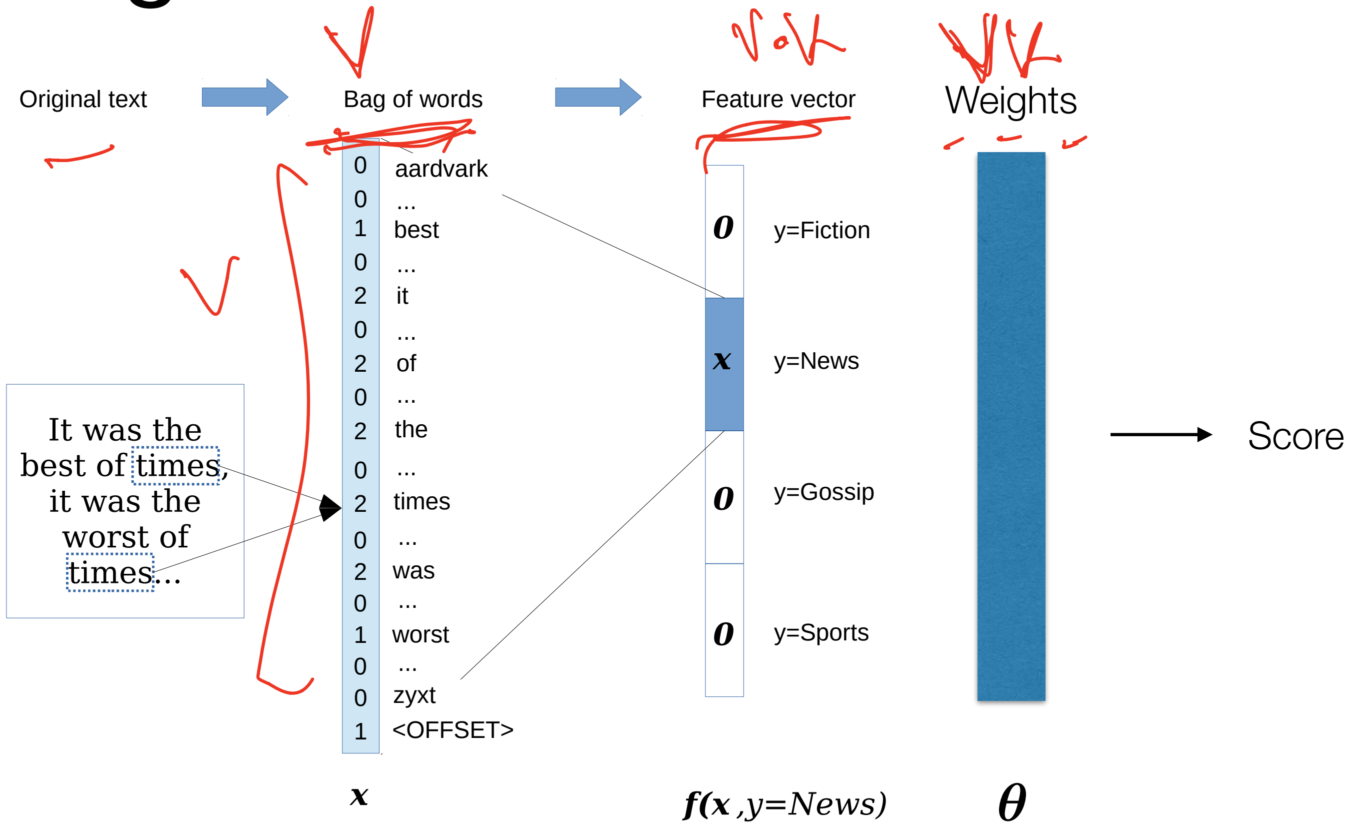
$$\Psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \sum_j \theta_j f_j(\boldsymbol{x}, y)$$

$$f(x,y)^{\text{after } n} = [\underbrace{0\,0\,0\,0\,0\,0\,0\,0\,0}_{\text{NEWS}} \mid \underbrace{0\,0\,\overset{x_{\text{whole}}}{\boxed{3}}\,0\,0\,0\,2\,0\,1\,0}_{\text{FICTION}} \mid 0\,0\,0\,0\,0\,0 \ldots]$$

Length VK

Very long feat vec!

Bag of words: linear model



$$\Psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \sum_j \theta_j f_j(\boldsymbol{x}, y)$$

How to set parameters?

- Could use deterministic 1 and 0 weights — implicit in lexicon/dictionary/keyword methods (e.g. racial slur blacklist, sentiment lexicons, etc.)
- But if you have labeled data, typically **supervised learning** is better.
- Labeled data: "gold-standard" (text, label) pairs

$$\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$$

- Two linear, probabilistic models today
  - Naive Bayes
  - Logistic Regression

Naive Bayes

- Assume a model of both text and labels (each doc i.i.d.)

$$p(\boldsymbol{x}^{(1:N)}, y^{(1:N)}) = \prod_{i=1}^{N} p_{X,Y}(\boldsymbol{x}^{(i)}, y^{(i)})$$

- p(x,y) is a **generative model**: has a story of how both the label and document text was generated
  - generating text is a.k.a. **language model** — other LMs are used a lot in NLP
- Once we have a model, we can do:
  1. Learning: fit p(x,y)'s parameters to training data (using MLE)
  2. Prediction ("Search"): infer labels on new documents (using Bayes Rule)

$\underbrace{\text{It was it not good}}$

5 tokens (long

4 unique word types

NB generative model

Multinom(M_i; $\phi_{y^{(i)}}$)

Algorithm 1 Generative process for the Naïve Bayes classification model

for Instance $i \in \{1, 2, \ldots, N\}$ **do**:
  Draw the label $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$;
  Draw the word counts $\boldsymbol{x}^{(i)} \mid y^{(i)} \sim \text{Multinomial}(\boldsymbol{\phi}_{y^{(i)}})$.

$P(y=k) = \mu_k$ $\phi_k \in \mathbb{R}^V$

Algorithm 2 Alternative generative process for the Naïve Bayes classification model

for Instance $i \in \{1, 2, \ldots, N\}$ **do**:
  Draw the label $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$;
  for Token $m \in \{1, 2, \ldots, M_i\}$ **do**:
    Draw the token $w_m^{(i)} \mid y^{(i)} \sim \text{Categorical}(\boldsymbol{\phi}_{y^{(i)}})$.

$P(w_m = \text{"cat"} \mid y=k) = \phi_{k,\text{"cat"}}$

- Types vs. Tokens ~> M_i
- Generative probability notation:
  - a ~ Distrib(theta): "Random variable a is sampled according to distribution Distrib, parameterized by theta." names, symbol
- Parameters
  - μ_k: prior probability of class k $\mu \in \mathbb{R}^K$: Simplex(K)
  - $\phi_{k,w}$: probability word w gets generated under doc class k
- "Naive": each word token is generated independently.

"Multinom NB"

NB prediction

- First assume we have parameters. How do we predict the label given text? Chose the one with *highest posterior probability* p(y | **x**) = p(**x**, y) / p(**x**)

$$p(y|x) = \frac{p(x|y)\,p(y)}{p(x)}$$

$$\hat{y} = \operatorname*{argmax}_y \log \mathrm{p}(\boldsymbol{x}, y; \boldsymbol{\mu}, \boldsymbol{\phi})$$

$$= \operatorname*{argmax}_y \log \mathrm{p}(\boldsymbol{x} \mid y; \boldsymbol{\phi}) + \log \mathrm{p}(y; \boldsymbol{\mu})$$

$$\log \mathrm{p}(\boldsymbol{x} \mid y; \boldsymbol{\phi}) + \log \mathrm{p}(y; \boldsymbol{\mu}) = \log \left[\cancel{B(\boldsymbol{x})} \prod_{j=1}^{V} \phi_{y,j}^{x_j} \right] + \log \mu_y$$

$$= \log \cancel{B(\boldsymbol{x})} + \sum_{j=1}^{V} x_j \log \phi_{y,j} + \log \mu_y$$

- This can be shown to be a linear model (see text). $\vec{\theta}^{T} \vec{f}(\vec{x}, y)$
  - Parameters = log probs of words and class priors
  - Features = count of word under candidate class; candidate class

NB learning

- Intuitively, **relative frequency estimation** sounds good:

$$\phi_{y,j} = \frac{\text{count}(y,j)}{\sum_{j'=1}^{V} \text{count}(y,j')} = \frac{\sum_{i:y^{(i)}=y} x_j^{(i)}}{\sum_{j'=1}^{V} \sum_{i:y^{(i)}=y} x_{j'}^{(i)}}$$

- This has a deeper theoretical basis!

NB learning: MLE

- **Maximum Likelihood Estimation**: choose params that give highest likelihood to the training data

$\theta = (\phi, \mu)$

$\phi \in \mathbb{R}^{VK}$, $\mu \in \mathbb{R}^{K}$

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathrm{p}(\boldsymbol{x}^{(1:N)}, y^{(1:N)}; \boldsymbol{\theta})$$

$$= \operatorname*{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^{N} \mathrm{p}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

$$= \operatorname*{argmax}_{\boldsymbol{\theta}} \underbrace{\sum_{i=1}^{N} \log \mathrm{p}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})}_{\mathcal{L}(\boldsymbol{\theta})}$$

log likelihood function

NB learning: MLE

$\partial =$

- Choose **ϕ,μ** to maximize log-likelihood

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\mu}) = \sum_{i=1}^{N} \log \mathrm{p}_{\mathrm{mult}}(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}_{y^{(i)}}) + \log \mathrm{p}_{\mathrm{cat}}(y^{(i)}; \boldsymbol{\mu})$$

- Under the sum-to-1 constraints

$$\sum_{j}^{V} \phi_{y,j} = 1 \quad \forall y \qquad\qquad \sum_{k}^{K} \mu_k = 1$$

- Calculus with Lagrange multipliers
===> intuitive relative frequency estimates!

Bias-variance tradeoffs

words are sparse!

Dirichlet prior → Dir-Mult

- Does MLE overfit or underfit?
- Laplace smoothing: add a pseudocount,

$$\phi_{y,j} = \frac{\alpha + \text{count}(y,j)}{V\alpha + \sum_{j'=1}^{V} \text{count}(y,j')}$$

- (Why V**α** ?)
- **α**=>large ==> ?

$\alpha = .1$

$\alpha = 100$

$\alpha = 10{,}000$

$\phi_k \to \left[\frac{1}{V}, \frac{1}{V}, \dots, \frac{1}{V}\right]$ $\alpha = 1{,}000{,}000$

What about class priors?

Same thing!

- Choose **ϕ,μ** to maximize log-likelihood

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\mu}) = \sum_{i=1}^{N} \log \mathrm{p}_{\text{mult}}(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}_{y^{(i)}}) + \log \mathrm{p}_{\text{cat}}(y^{(i)}; \boldsymbol{\mu})$$

- **μ** is set to class frequencies in training data. Is this realistic?
    - [See our paper! Keith and O'Connor, 2018]

Cond. indep. is a problem

- We can do better than BOW features by **feature engineering** lots of little variants of words and phrases
  - e.g. ngram features … character n-grams … words with or without lowercasing … number of digits in the text … number of punctuation marks in the text … etc.
  - Even overlapping features can have useful predictive value
- But… does NB do well with repetitive features?

$$\Psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \sum_j \theta_j f_j(\boldsymbol{x}, y)$$

Discriminative learning

- NB (generative) learning chooses params to maximize p(X^{train},Y^{train}), then indirectly gives the linear prediction model.
- But if we just care about prediction, why not directly learn to minimize prediction errors?
  - Perceptron: choose params to minimize **1-0 loss**.
  - SVM: choose it to minimize **hinge loss**
- Logistic regression: choose theta to maximize **conditional log-likelihood** (a.k.a. minimize logistic loss a.k.a. cross-entropy)

Logistic regression

- Directly define the conditional probability of label given text via the **softmax** of the linear scoring function

$$\mathrm{p}(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp\left(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y')\right)}$$

exp and normalize

- Learning: choose theta to maximize the **conditional log-likelihood**,

$$\log \mathrm{p}(\boldsymbol{y}^{(1:N)} \mid \boldsymbol{x}^{(1:N)}; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \mathrm{p}(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= \sum_{i=1}^{N} \underbrace{\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)})}_{\text{Give high scores to observed y(i)}} - \log \sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y')\right)$$

Logistic loss

Negative log-likelihood for one example

$$\ell_{\text{LOGREG}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = -\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y'))$$

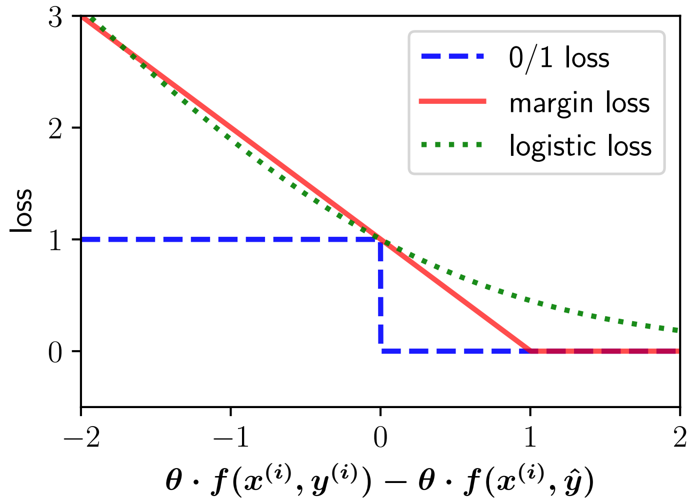


Figure 2.2: Margin, zero-one, and logistic loss functions.

- "Soft" grading of errors
  - 99% prob for y(i) => 😀
  - 1% prob for y(i) => 😂
  - 0%?

LogReg learning

- There is no closed form MLE
- But, fortunately, the log-lik is concave (NLL convex)
- Use gradient descent! or LBFGS
  - 1. Calculate gradient equations
  - 2. Use a batch or online gradient algorithm

Regularization

- For many NLP feature functions, training data is often linearly separable. Weights diverge to +/- inf
- Regularization is essential. Typically use the L2 norm of weights, resulting in a regularized loss:

$$L_{\textsc{LogReg}} = \frac{\lambda}{2}||\boldsymbol{\theta}||_2^2 - \sum_{i=1}^{N}\left(\boldsymbol{\theta}\cdot\boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \log\sum_{y'\in\mathcal{Y}}\exp\boldsymbol{\theta}\cdot\boldsymbol{f}(\boldsymbol{x}^{(i)}, y')\right)$$

Summary: NLP prediction

Scorer (model)

Params

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} \Psi(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$$

Predicted output

Possible outputs

Input (text)

Output candidate

- Today: Linear models **Ψ**, BOW **x** & **f**, multiclass **y**
- Models: Naive Bayes and Logistic Regression
  - Learning: (regularized) MLE
- Wednesday: Neural network **Ψ**
- Later: sequential **x**
- Later: structured output **y**