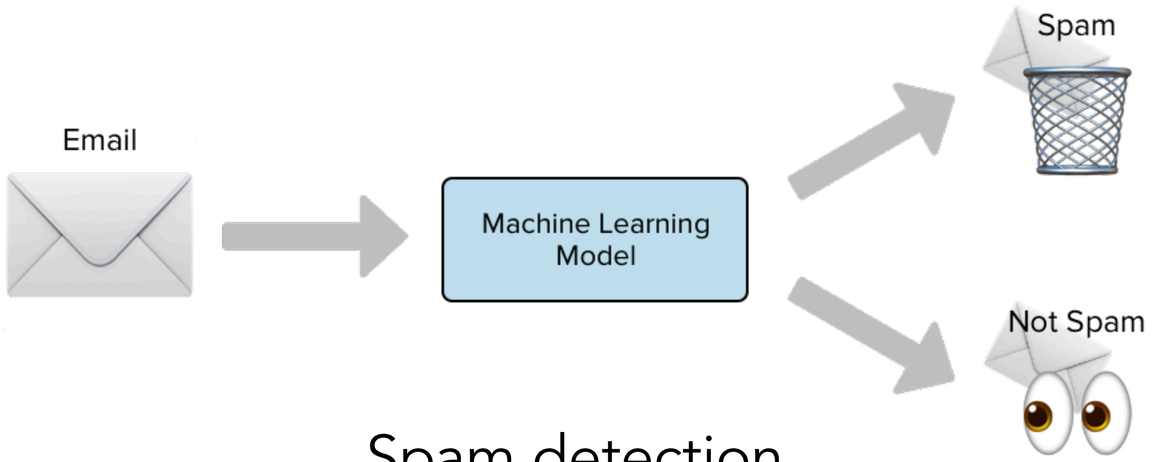
COS 484

L3: Text Classification

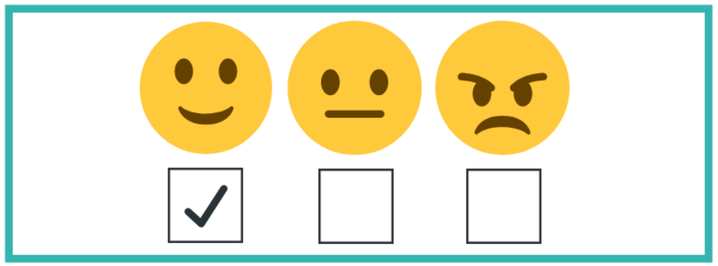
Spring 2022

Assignment 1 will be out later today — due Tuesday, Feb 15, 9:30am (in 2 weeks)

Why classify?



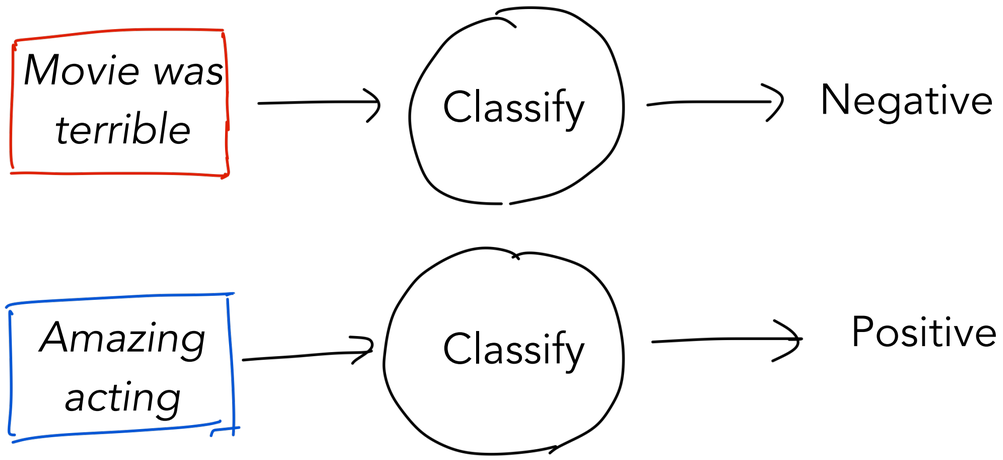
Spam detection



Sentiment analysis

- Authorship attribution
- Language detection
- News categorization
- …. and many more!

Text classification



- Inputs:
  - A document d
  - A set of classes $C = \{c_1, c_2, c_3, \ldots, c_m\}$
- Output:
  - Predicted class c for document d

Rule-based classification

- Combinations of features on words in document, meta-data

IF there exists word w in document d such that w in [good, great, extra-ordinary, …],

THEN output Positive

IF email address ends in [ithelpdesk.com, makemoney.com, spinthewheel.com, …]

THEN output SPAM

+ Can be very accurate

- Rules may be hard to define (and some even unknown to us!)

- Expensive

- Not easily generalizable

VADER-Sentiment-Analysis

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is *specifically attuned to sentiments expressed in social media*. It is fully open-sourced under the [MIT

Supervised Learning: Let's use statistics!

Let the machine figure out the best patterns using data

Inputs:

- Set of m classes $C = \{c_1, c_2, \ldots, c_m\}$
- Set of n 'labeled' documents: $\{(d_1, c_1), (d_2, c_2), \ldots, (d_n, c_n)\}$

Output:

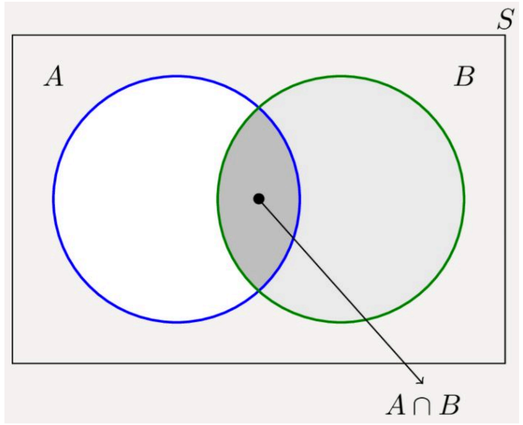
- Trained classifier, $F : d \rightarrow c$

Key questions:

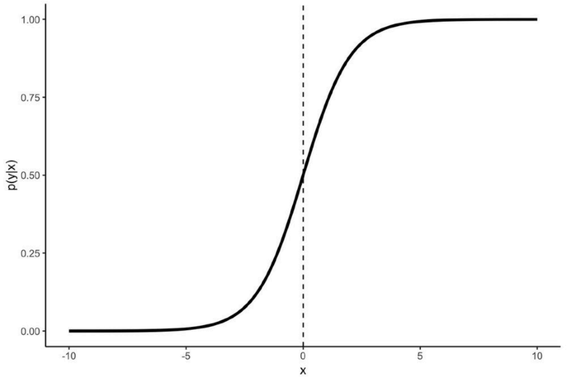
a) What is the form of F?

b) How do we learn F?

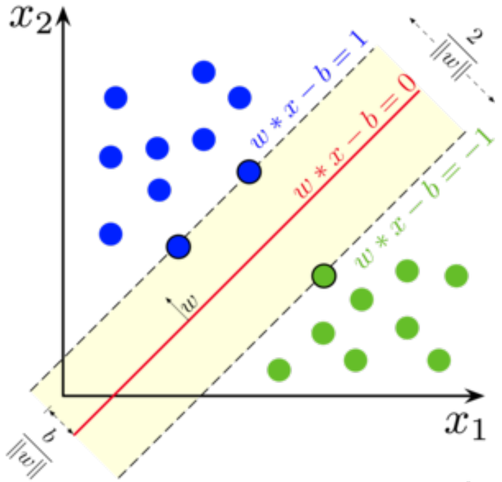
Types of supervised classifiers



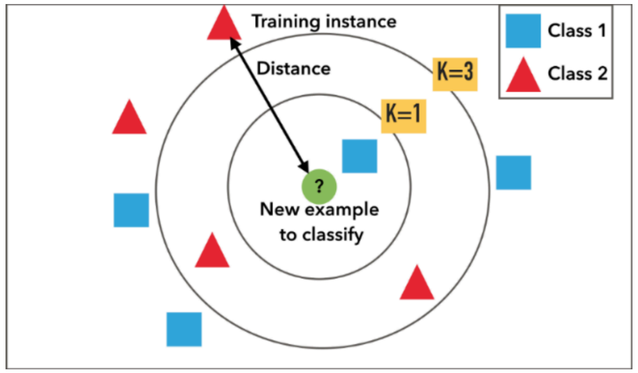
Naive Bayes



Logistic regression



Support vector machines



k-nearest neighbors

Multinomial Naive Bayes

- Simple classification model making use of Bayes rule
  - Bayes Rule:

d – document
c – class

$$P(c \mid d) = \frac{P(c)\ P(d \mid c)}{P(d)}$$

Predicting a class

d – document
c – class

- Best class, $c_{MAP} = \underset{c \in C}{\operatorname{argmax}} \; p(c|d)$

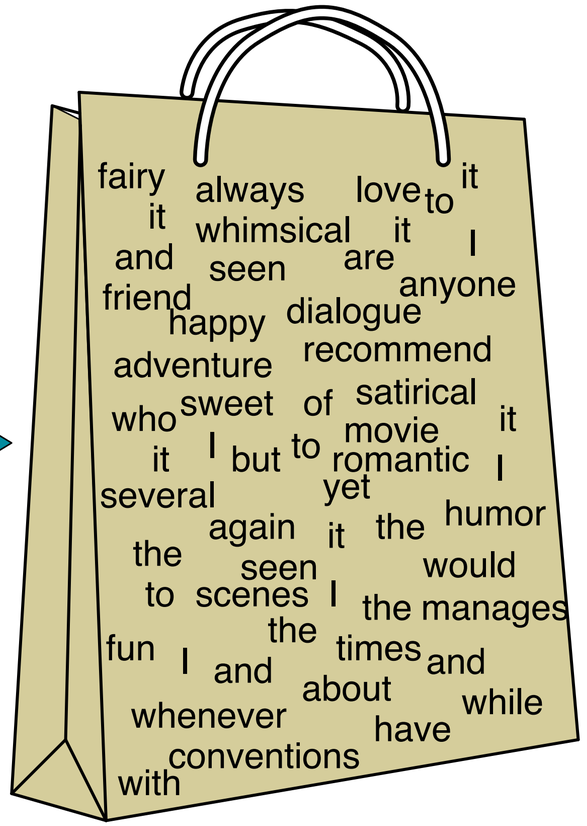
Maximum a posteriori (MAP) estimate

How to represent P(d | c)?

- Option 1: represent the entire sequence of words
  - $P(w_1, w_2, \ldots, w_K | c)$ *(too many sequences!)*
- Option 2: Bag of words
  - Assume position of each word is irrelevant (both absolute and relative)
  - $P(w_1, w_2, \ldots, w_K | c) = P(w_1 | c)P(w_2 | c) \ldots P(w_k | c)$
  - Probability of each word is *conditionally independent* of the other words given class **c**

Bag of words (BoW)

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

Predicting with Naive Bayes

- We now have:

$$c_{MAP} = \underset{c}{\operatorname{argmax}} \; P(d \mid c) \, P(c)$$

Predicting with Naive Bayes

- We now have:

$$c_{MAP} = \underset{c}{\operatorname{argmax}} \; P(d \mid c) \, P(c)$$

$$= \underset{c}{\operatorname{argmax}} \; P(w_1, w_2, \ldots w_k \mid c) \; P(c)$$

Predicting with Naive Bayes

- We now have:

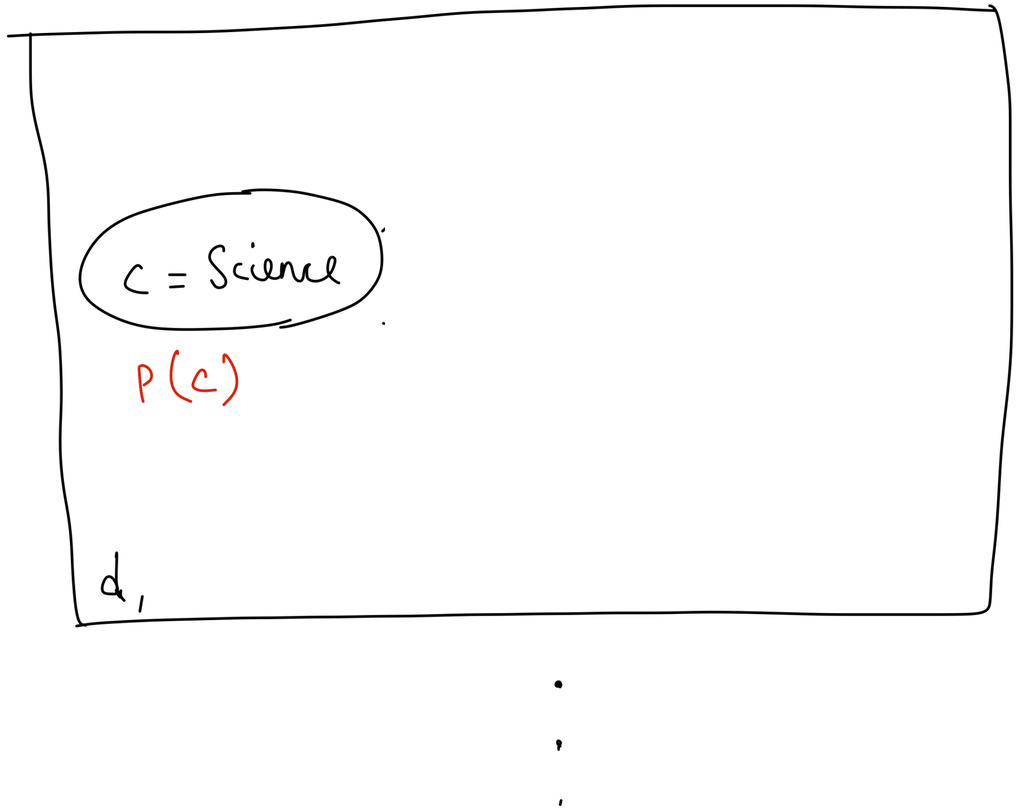
$$c_{MAP} = \underset{c}{\operatorname{argmax}} \; P(d \mid c)\, P(c)$$

$$= \underset{c}{\operatorname{argmax}} \; P(w_1, w_2, \ldots w_k \mid c)\; P(c)$$

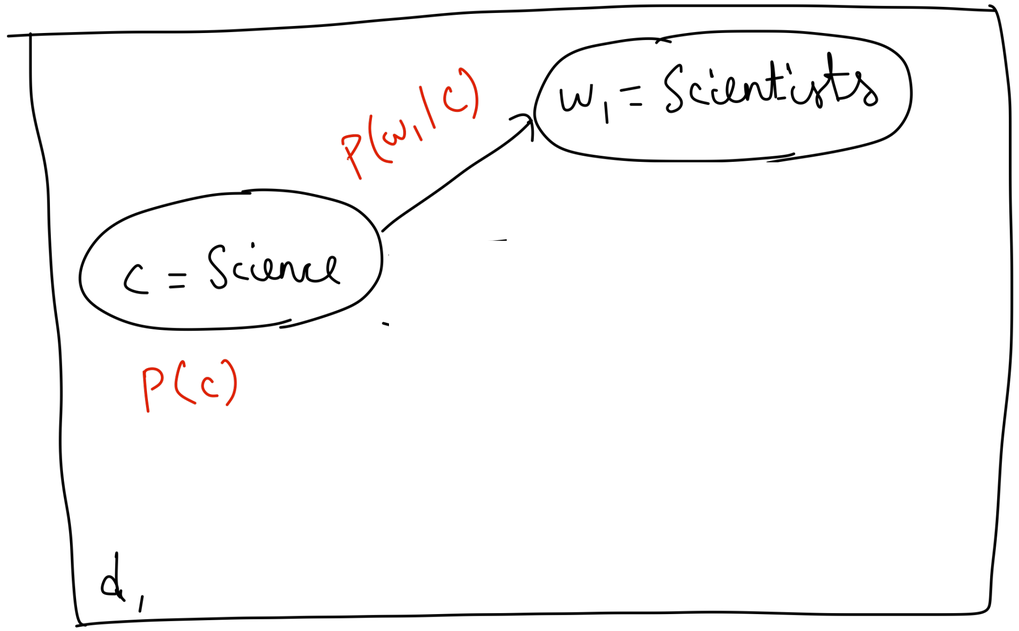
$$= \underset{c}{\operatorname{argmax}} \; P(c) \prod_{i=1}^{k} P(w_i \mid c)$$

(using BOW assumption)

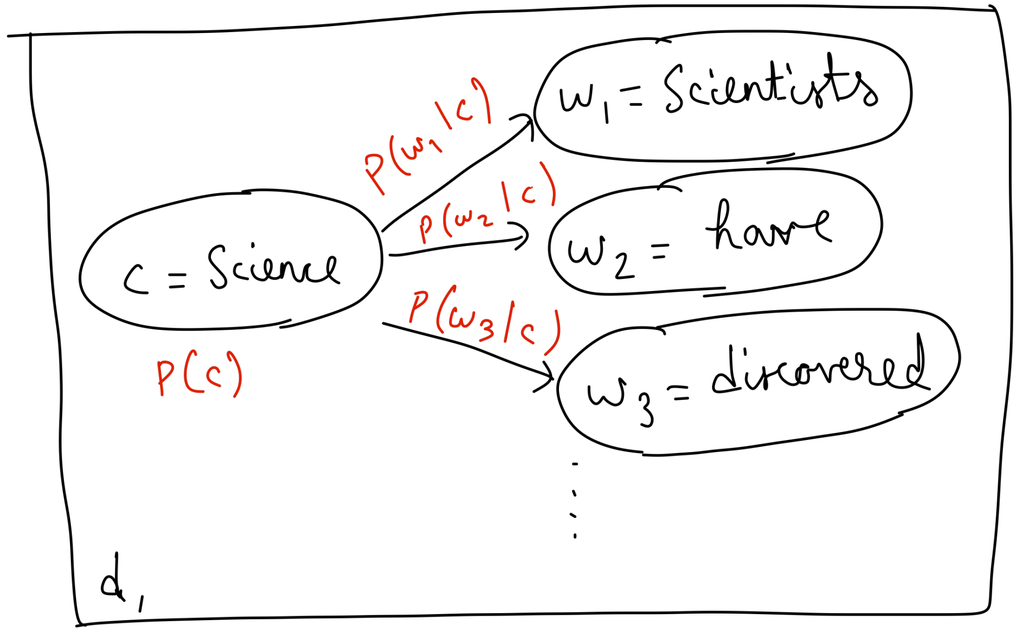
Naive Bayes as a generative model



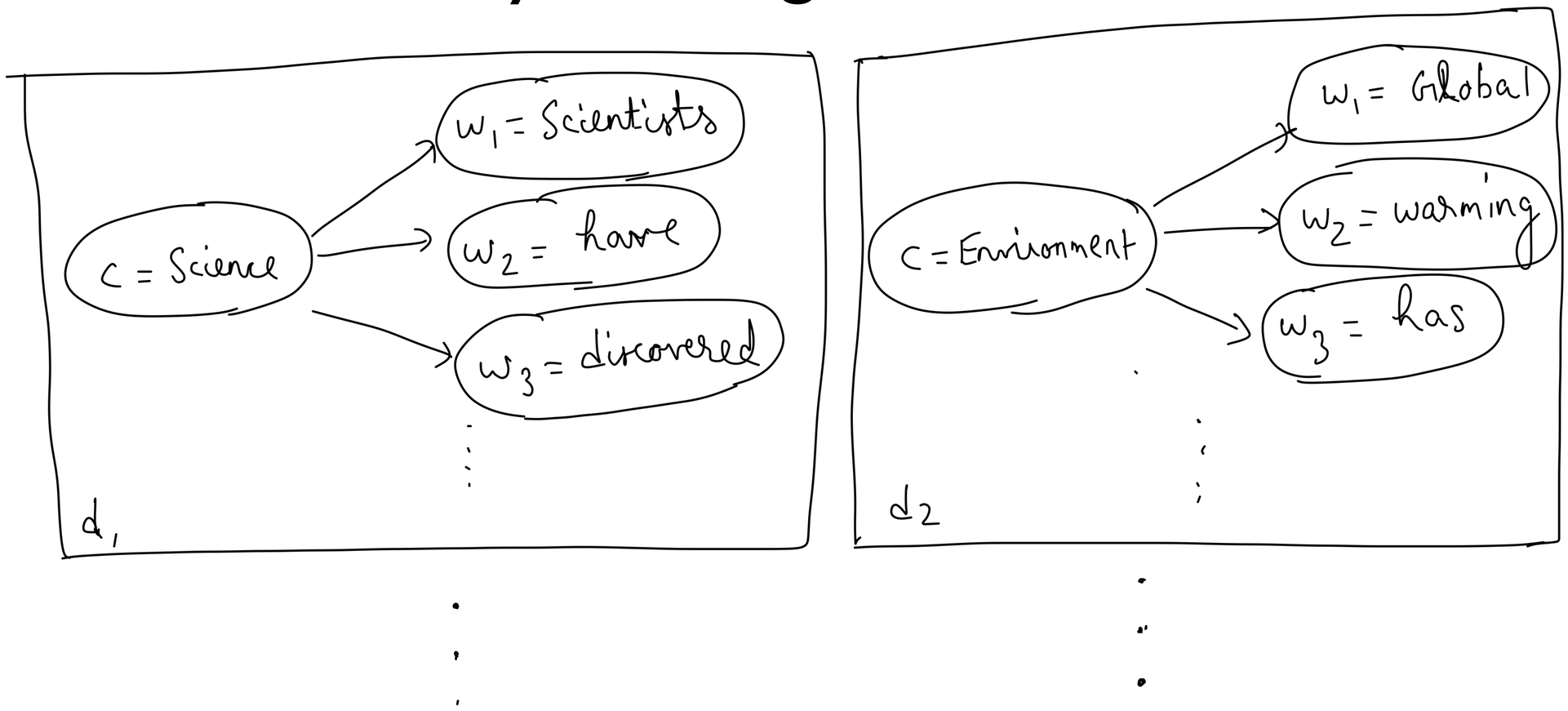
Naive Bayes as a generative model



Naive Bayes as a generative model



Naive Bayes as a generative model



Generate the entire data set one document at a time

Estimating the model

Maximum likelihood estimates:

$$\hat{P}(c_j) = \frac{\text{Count}(\text{class} = c_j)}{\sum_{c} \text{Count}(\text{class} = c)}$$

← Total # of documents

$$\underset{c}{\text{argmax}} \; P(c) \prod_{i=1}^{k} P(w_i \mid c)$$

Data sparsity

- What if count('amazing', *positive*) = *0*?

  ➡ Implies P('amazing' | *positive*) = 0

- Given a review document, d = "…. most amazing movie ever …"

$$c_{MAP} = \underset{c}{\operatorname{argmax}} \; \hat{P}(c) \prod_{i=1}^{K} P(w_i \mid c)$$

$$= \underset{c}{\operatorname{argmax}} \; \hat{P}(c) \cdot 0$$

This sounds familiar…

Solution: Smoothing!

- Laplace smoothing:

$$\hat{P}(w_i \mid c) = \frac{\text{Count}(w_i, c) + \alpha}{\left[\sum_w \text{count}(w, c)\right] + \alpha |V|}$$

↖ Vocabulary size

- Simple, easy to use
- Effective in practice

Overall process

Input: Set of annotated documents $\{(d_i, c_i)\}_{i=1}^{n}$

A. Compute vocabulary **V** of all words

B. Calculate $\hat{P}(c_j) = \dfrac{\text{Count}(c_j)}{\text{n}}$

C. Calculate $\hat{P}(w_i \mid c_j) = \dfrac{\text{Count}(w_i, c_j) + \alpha}{\sum_{w \in V} \left[\text{Count}(w, c_j) + \alpha\right]}$

D. (Prediction) Given document $d = (w_1, w_2, \ldots, w_k)$

$$c_{MAP} = \arg\max_{c} \hat{P}(c) \prod_{i=1}^{K} \hat{P}(w_i \mid c)$$

prior

Naive Bayes as a language model

- Which class assigns the higher probability to s?

Model pos	
0.1	I
0.1	love
0.01	this
0.05	fun
0.1	film

Model neg	
0.2	I
0.001	love
0.01	this
0.005	fun
0.1	film

Sentence s

I	love	this	fun	film
0.1	0.1	0.01	0.05	0.1
0.2	0.001	0.01	0.005	0.1

A) pos B) neg C) both equal

Naive Bayes as a language model

- Which class assigns the higher probability to s?

Model pos	
0.1	I
0.1	love
0.01	this
0.05	fun
0.1	film

Model neg	
0.2	I
0.001	love
0.01	this
0.005	fun
0.1	film

Sentence s

I	love	this	fun	film
0.1	0.1	0.01	0.05	0.1
0.2	0.001	0.01	0.005	0.1

$$P(s|pos) > P(s|neg)$$

Features

Rank	Category	Feature
1	Subject	Number of capitalized words
2	Subject	Sum of all the character lengths of words
3	Subject	Number of words containing letters and numbers
4	Subject	Max of ratio of digit characters to all characters of each word
5	Header	Hour of day when email was sent

(a)

Rank	Category	Feature
1	Subject	Min of the compression ratio for the bz2 compressor
2	Subject	Min of the compression ratio for the zlib compressor
3	Subject	Min of character diversity of each word
4	Subject	Min of the compression ratio for the lzw compressor
5	Subject	Max of the character lengths of words

(b)

Spam URLs Features

Rank	Category	Feature
1	URL	The number of all URLs in an email
2	URL	The number of unique URLs in an email
3	Payload	Number of words containing letters and numbers
4	Payload	Min of the compression ratio for the bz2 compressor
5	Payload	Number of words containing only letters

Rank	Category	Feature
1	Header	Day of week when email was sent
2	Payload	Number of characters
3	Payload	Sum of all the character lengths of words
4	Header	Minute of hour when email was sent
5	Header	Hour of day when email was sent

Top features for spam detection

- In general, Naive Bayes can use any set of features, not just words:
  - URLs, email addresses, Capitalization, …
  - Domain knowledge crucial to performance

Evaluating a classifier

- Precision: % of selected classes that are correct

$$\text{Precision}(+) = \frac{TP}{TP + FP} \qquad \text{Precision}(-) = \frac{TN}{TN + FN}$$

- Recall: % of correct items selected

$$\text{Recall}(+) = \frac{TP}{TP + FN} \qquad \text{Recall}(-) = \frac{TN}{TN + FP}$$

F-Score

- Combined measure using precision and recall
- Harmonic mean of Precision and Recall

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Or more generally,

$$F_\beta = \frac{(1 + \beta^2) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

Advantages of Naive Bayes

- Very fast, low storage requirements
- Robust to irrelevant features

  Irrelevant features cancel each other without affecting results
- Very good in domains with many equally important features

  Decision trees suffer from fragmentation in such cases — especially if little data
- Optimal if the independence assumptions hold

  If assumed independence is correct, this is the ‘Bayes optimal’ classifier
- A good dependable baseline for text classification

  However, other classifiers can give better accuracy

Failings of Naive Bayes (1)

Independence assumptions are too strong

x1	x2	Class: x_1 XOR x_2
1	1	0
0	1	1
1	0	1
0	0	0

- XOR problem: Naive Bayes cannot learn a decision boundary
- Both variables are jointly required to predict class

Failings of Naive Bayes (2)

Class imbalance

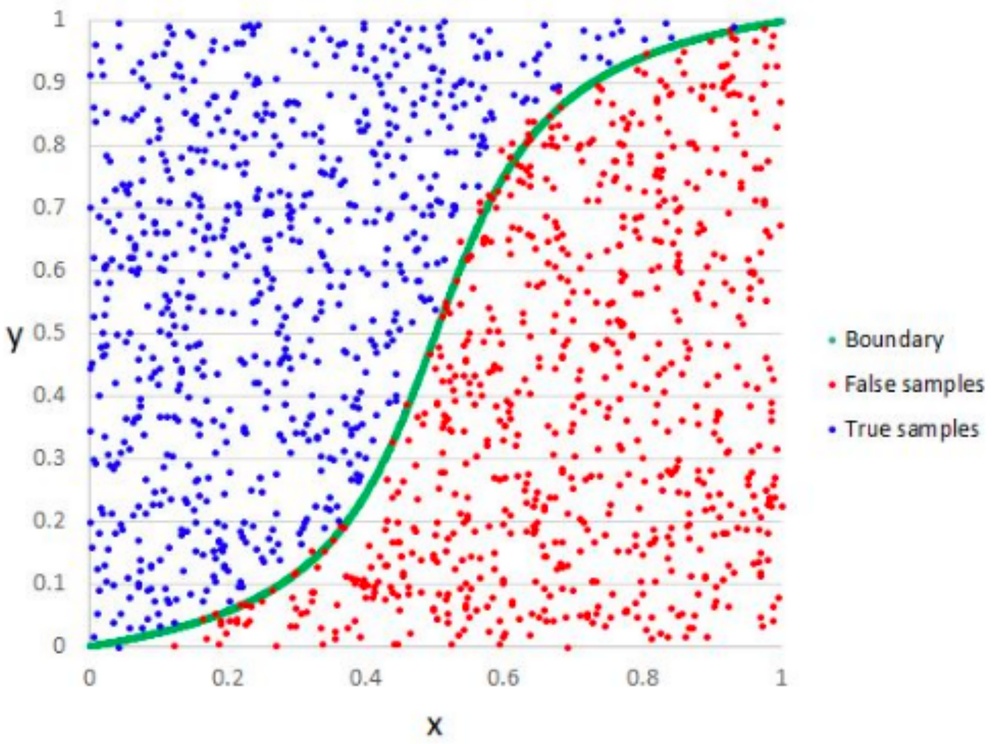
- One or more classes have more instances than others in data
- Data skew causes NB to prefer one class over the other
- Potential solution: Complement Naive Bayes (Rennie et al., 2003)

$$\hat{P}(w_i \mid \tilde{c}_j) = \frac{\sum_{c \neq c_j} \text{Count}(w_i, c)}{\sum_{c \neq c_j} \sum_{w} \text{Count}(w, c)}$$

→ Count # times word occurs in classes other than c

Logistic Regression

Logistic Regression



- Powerful supervised model
- Baseline approach for many NLP tasks
- Connections with neural networks
- Binary (two classes) or multinomial (>2 classes)

Discriminative Model

- Logistic Regression is a *discriminative* model
- Naive Bayes: *generative* model

Discriminative Model

- Logistic Regression: $\hat{c} = \underset{c}{\operatorname{argmax}} \; P(c|d)$
- Naive Bayes: $\hat{c} = \underset{c}{\operatorname{argmax}} \; P(c)\,P(d|c)$

Cat: a domesticated carnivorous mammal with soft fur, a short snout, and retractable claws.

Dog: a domesticated carnivorous mammal with a long snout, nonretractable claws, and a barking, howling, or whining voice.

Overview

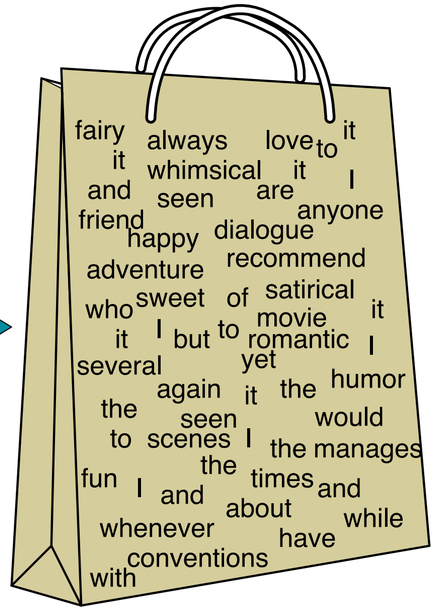
- Inputs:
  1. Classification instance in a **feature representation**
  2. **Classification function** to compute $\hat{y}$ using $P(\hat{y}\,|\,x)$
  3. **Loss function** (for learning)
  4. Optimization **algorithm**
- Train phase: Learn the **parameters** of the model to minimize **loss function**
- Test phase: Apply **parameters** to predict class given a new input x

1. Feature representation

- Input observation: $x^{(i)}$
- Feature vector: $[x_1, x_2, \ldots, x_d] = \mathbf{x}$
- Feature j of ith input : $x_j^{(i)}$

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

$x^{(i)}$



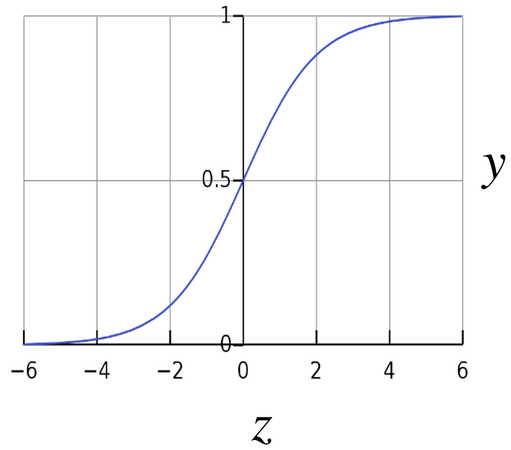
Bag of words

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1

$[x_1, x_2, \ldots, x_d]$

2. Classification function

- *Given*: Input feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]$
- *Output*: $P(y = 1 \mid \mathbf{x})$ and $P(y = 0 \mid \mathbf{x})$
  (binary classification)
- Require a *function*, $F : \mathbb{R}^{\mathrm{d}} \rightarrow [0,1]$
- Sigmoid:



$$f(z) = \frac{1}{1 + e^{-z}} = \frac{e^{z}}{1 + e^{z}}$$

Weights and Biases

- *Which features are important* and *how much*?
- Learn a vector of **weights** and a **bias**
- Weights: Vector of real numbers, $\mathbf{w} = [w_1, w_2, \ldots, w_d]$
- Bias: Scalar intercept, b
- Given input features $\mathbf{x}$, : $z = \mathbf{w} \cdot \mathbf{x} + b$
- Therefore, $f(\mathbf{w} \cdot \mathbf{x} + b) = \dfrac{e^{\mathbf{w} \cdot \mathbf{x} + b}}{1 + e^{\mathbf{w} \cdot \mathbf{x} + b}}$

Putting it together

- Compute probabilities: $P(y = 1 \mid \mathbf{x}) = \dfrac{1}{1 + e^{-z}}$

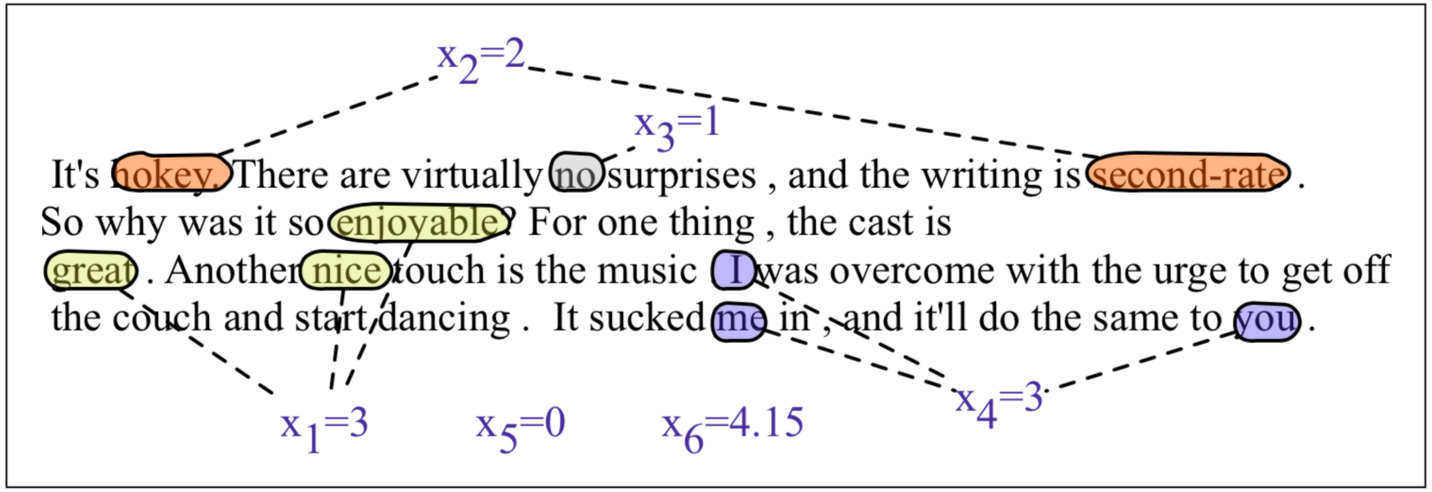
$$P(y = 1) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\begin{aligned} P(y = 0) &= 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b) \\ &= 1 - \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} = \frac{e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} \end{aligned}$$

- Decision boundary:

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1 \mid \mathbf{x}) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Example: Sentiment classification



Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if ``no'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if ``!'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Remember that the values make up the feature vector!

Example: Sentiment classification

Var	Definition	Value
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if ``no'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if ``!'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

- Assume weights $\mathbf{w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ and bias $b = 0.1$

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Designing features

- Most important rule: Data is *key*!
- Linguistic intuition (e.g. part of speech tags, parse trees)
- Complex combinations

$$x_1 = \begin{cases} 1 & \text{if } \text{“}Case(w_i) = \text{Lower”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1 & \text{if } \text{“}w_i \in \text{AcronymDict”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_3 = \begin{cases} 1 & \text{if } \text{“}w_i = \text{St. \& } Case(w_{i-1}) = \text{Cap”} \\ 0 & \text{otherwise} \end{cases}$$

- Feature templates
  - Sparse representations, hash only seen features into index
  - Ex. Trigram(*logistic regression classifier*) = Feature #78
- Advanced: Representation learning (we will see this later!)

Logistic Regression: what's good and what's not

- More freedom in designing features
  - No strong independence assumptions like Naive Bayes
  - More robust to correlated features ("San Francisco" vs "Boston")
    —LR is likely to work better than NB
  - Can even have the same feature twice! (*why?*)
- May not work well on small datasets (compared to Naive Bayes)
- Interpreting learned weights can be challenging

3. Learning

- We have our **classification function -** how to assign weights and bias?
- Goal: prediction $\hat{y}$ as close as possible to actual label y
  - **Distance metric/Loss function** between predicted $\hat{y}$ and true y : $L(\hat{y}, y)$
  - **Optimization algorithm** for updating weights

Loss function

- Assume $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$
- $L(\hat{y}, y) =$ Measure of difference between $\hat{y}$ and y. But what form?
- Maximum likelihood estimation (conditional):
  - Choose w and b such that $\log P(y \mid x)$ is maximized for true labels y paired with input x
  - Similar to language models!
    - where we chose parameters to maximize $\log P(w_t \mid w_{t-n}, \ldots, w_{t-1})$ given a corpus

Cross Entropy loss for a single instance

- Assume a single data point (x, y) and two possible classes to choose from
- **Classifier probability:** $P(y \mid x) = \hat{y}^{\,y}(1-\hat{y})^{1-y}$ (compact notation)
- **Log probability:** $\log P(y \mid x) = \log[\hat{y}^{y}(1-\hat{y})^{1-y}]$

  $= y \log \hat{y} + (1-y)\log(1-\hat{y})$ (maximize this)
- **Loss:** $-\log P(y \mid x) = -\,[y \log \hat{y} + (1-y)\log(1-\hat{y}]$ (minimize this)
  - $y = 1 \implies -\log \hat{y}$, and $y = 0 \implies -\log(1-\hat{y})$

Cross Entropy loss

- For n data points $(x^{(i)}, y^{(i)})$,
- **Classifier probability:** $\Pi_{i=1}^{n} P(y \mid x) = \Pi_{i=1}^{n} \hat{y}^{y}(1-\hat{y})^{1-y}$
- **Loss:** $-\log \prod_{i=1}^{n} P(y \mid x) = -\sum_{i=1}^{n} \log P(y \mid x)$

$$L_{CE} = -\sum_{i=1}^{n} [y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

Example: Computing CE Loss

Var	Definition	Value
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

- Assume weights $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ and bias $b = 0.1$
- If y = 1 (positive sentiment), $L_{CE} = -\log(0.69) = 0.37$
- If y = 0 (negative sentiment), $L_{CE} = -\log(0.31) = 1.17$

Properties of CE Loss

- $L_{CE} = -\sum_{i=1}^{n} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$
- What values can this loss take?

A) 0 to ∞ B) $-\infty$ to ∞ C) $-\infty$ to 0 D) 1 to ∞

Properties of CE Loss

- $L_{CE} = -\sum_{i=1}^{n} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$
- Ranges from 0 (perfect predictions) to ∞
- Lower the value, better the classifier
- *Cross-entropy* between the true distribution $P(y \mid x)$ in the data and predicted distribution $P(\hat{y} \mid x)$

4. Optimization

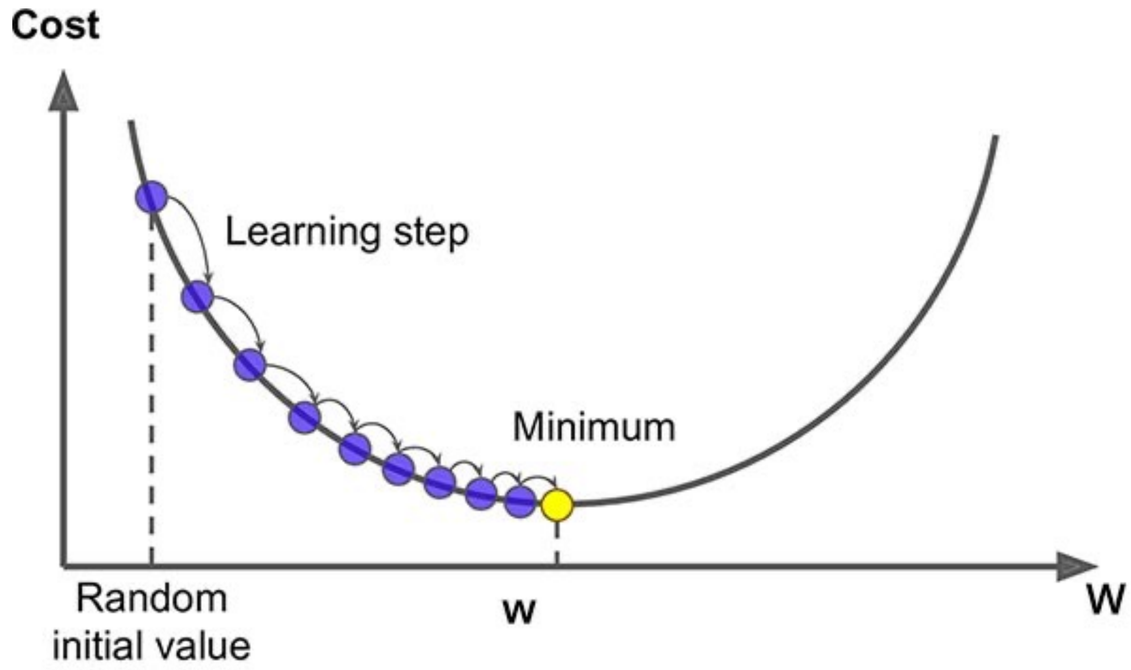
- We have our **classification function** and **loss function** - how do we find the best w and b?

$$\theta = [w; b]$$

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

- Gradient descent:
  - Find direction of steepest slope
  - Move in the opposite direction

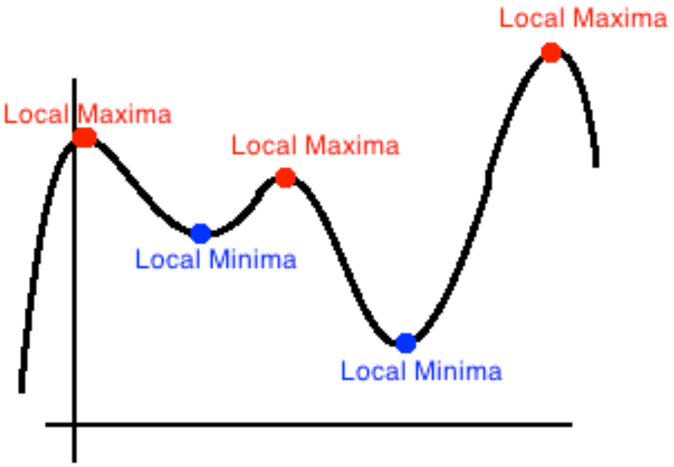
Gradient descent (1-D)



$$\theta^{t+1} = \theta^t - \eta \frac{d}{d\theta} f(x;\theta)$$

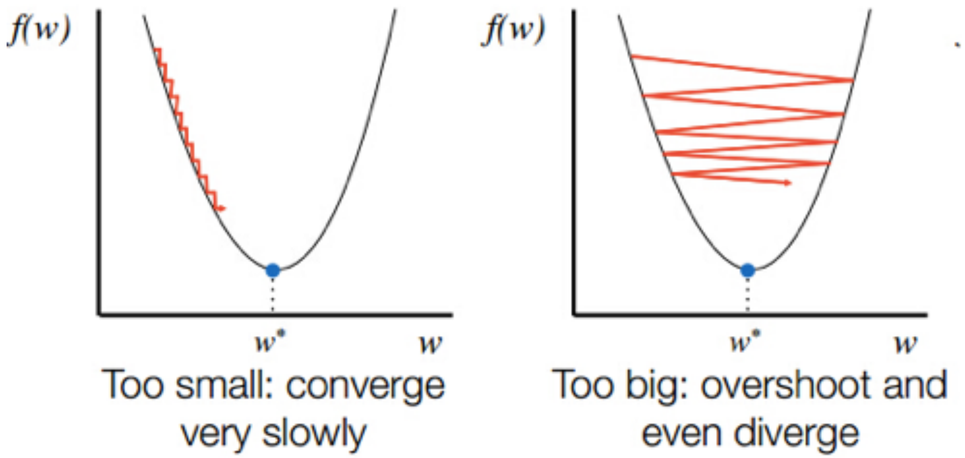
Gradient descent for LR

- Cross entropy loss for logistic regression is **convex** (i.e. has only one global minimum)
  - No local minima to get stuck in
- Deep neural networks are not so easy
  - Non-convex



Learning Rate

- Updates: $\theta^{t+1} = \theta^t - \eta \frac{d}{d\theta} f(x;\theta)$
- Magnitude of movement along gradient
- Higher/faster learning rate = larger updates to parameters



Too small: converge very slowly

Too big: overshoot and even diverge

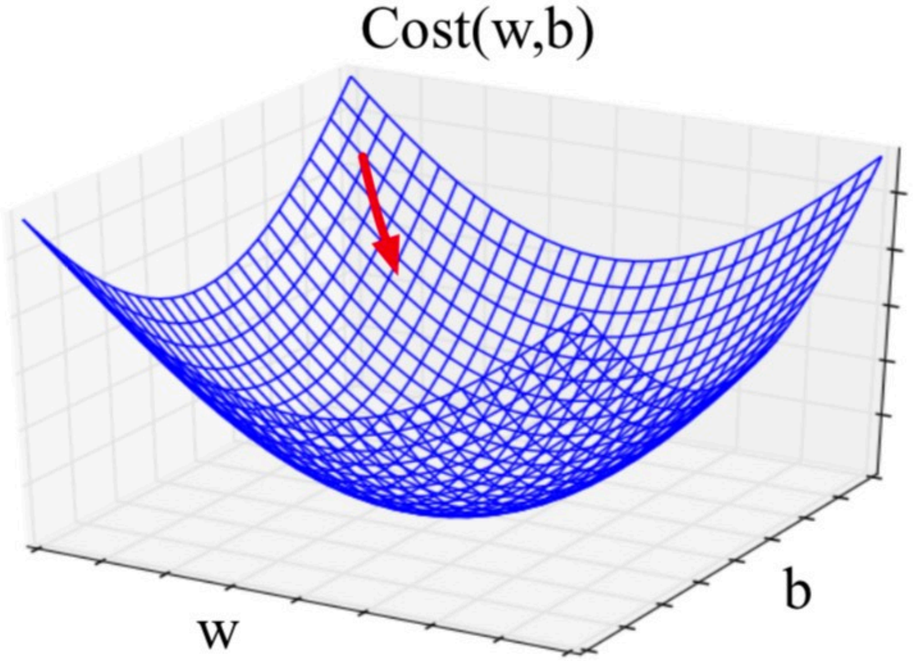
Recap: Logistic regression

- Inputs:
  1. Classification instance in a **feature representation**
  2. **Classification function** to compute $\hat{y}$ using $P(\hat{y}\,|\,x)$
  3. **Loss function** (for learning)
  4. Optimization **algorithm**
- Train phase: Learn the **parameters** of the model to minimize **loss function**
- Test phase: Apply **parameters** to predict class given a new input x

Gradient descent with vector weights

- In LR: weight w is a vector
- Express slope as a partial derivative of loss w.r.t each weight:

$$\nabla_\theta L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$



- Updates: $\theta^{(t+1)} = \theta^t - \eta \nabla L(f(x;\theta), y)$

Gradient for logistic regression

- $L_{CE} = -\sum_{i=1}^{n} [y^{(i)} \log \sigma(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) + (1 - y^{(i)}) \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x}^{(i)} + b))]$
- Gradient, $\frac{dL_{CE}(\mathbf{w}, b)}{dw_j} = \sum_{i=1}^{n} [\sigma(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) - y^{(i)}] x_j^{(i)}$

Diff between true y and prediction

input feature value

- $\frac{dL_{CE}(\mathbf{w}, b)}{db} = \sum_{i=1}^{n} [\sigma(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) - y^{(i)}]$

Stochastic Gradient Descent

- Online optimization
- Compute loss and minimize after **each** training example

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

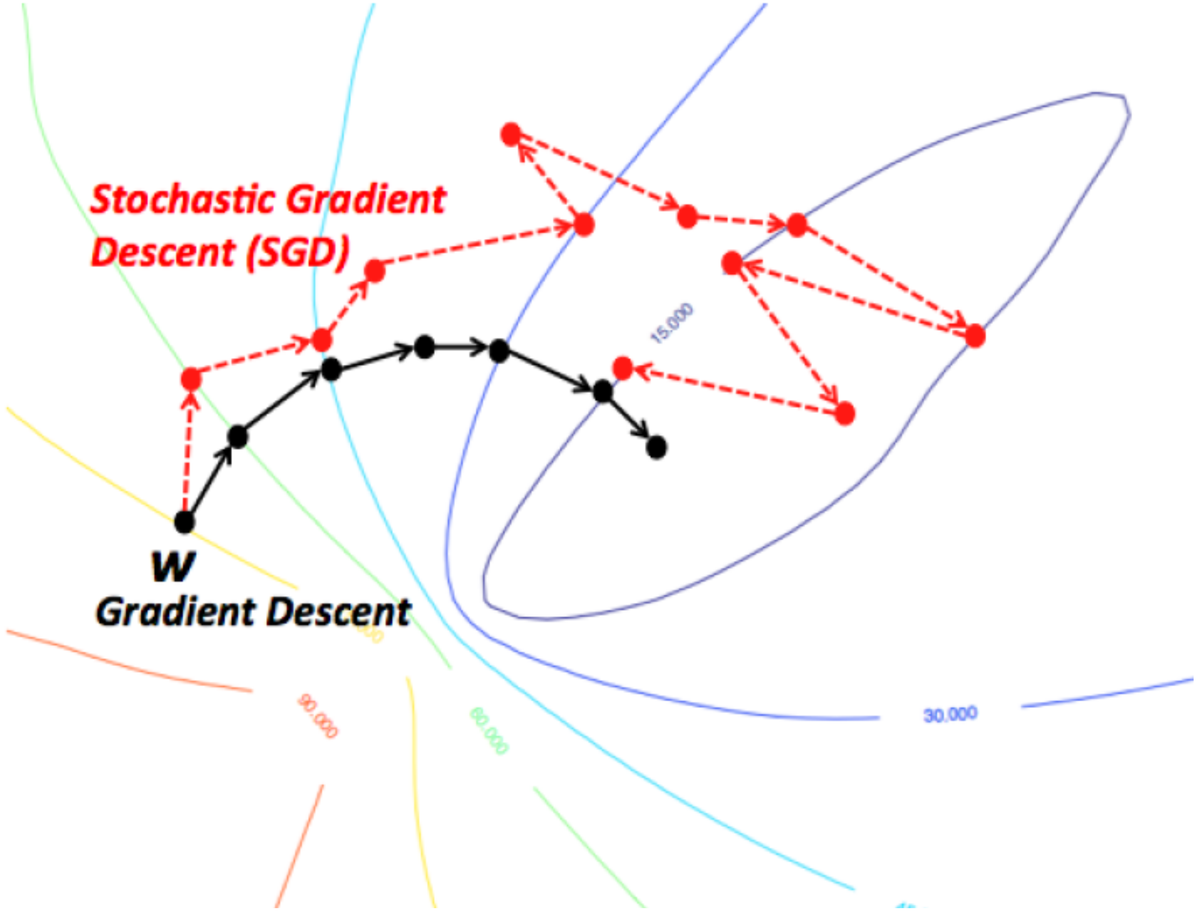
```
    # where: L is the loss function
    #        f is a function parameterized by θ
    #        x is the set of training inputs x^(1), x^(2),..., x^(n)
    #        y is the set of training outputs (labels) y^(1), y^(2),..., y^(n)

θ ← 0
repeat til done    # see caption
   For each training tuple (x^(i), y^(i)) (in random order)
      1. Optional (for reporting):        # How are we doing on this tuple?
         Compute ŷ^(i) = f(x^(i); θ)      # What is our estimated output ŷ?
         Compute the loss L(ŷ^(i), y^(i)) # How far off is ŷ^(i)) from the true output y^(i)?
      2. g ← ∇_θ L(f(x^(i); θ), y^(i))    # How should we move θ to maximize loss?
      3. θ ← θ − η g                      # Go the other way instead
return θ
```

Per Instance Loss

Stochastic Gradient Descent

- Online optimization
- Compute loss and minimize after **each** training example



Regularization

- Training objective: $\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log P(y^{(i)} \mid x^{(i)})$

- This might fit the training set too well! (including noisy features)

- Poor generalization to the unseen test set — **Overfitting**

- **Regularization** helps prevent overfitting

$$\hat{\theta} = \arg\max_{\theta} \left[\sum_{i=1}^{n} \log P(y^{(i)} \mid x^{(i)}) - \alpha R(\theta) \right]$$

Penalize large weights

L2 regularization

- $R(\theta) = ||\theta||^2 = \sum_{j=1}^{d} \theta_j^2$

- Euclidean distance of weight vector θ from origin

- L2 regularized objective:

$$\hat{\theta} = \arg\max_{\theta} \left[\sum_{i=1}^{n} \log P(y^{(i)} | x^{(i)}) - \alpha \sum_{j=1}^{d} \theta_j^2 \right]$$

L1 Regularization

- $R(\theta) = ||\theta||_1 = \sum_{j=1}^{d} |\theta_j|$

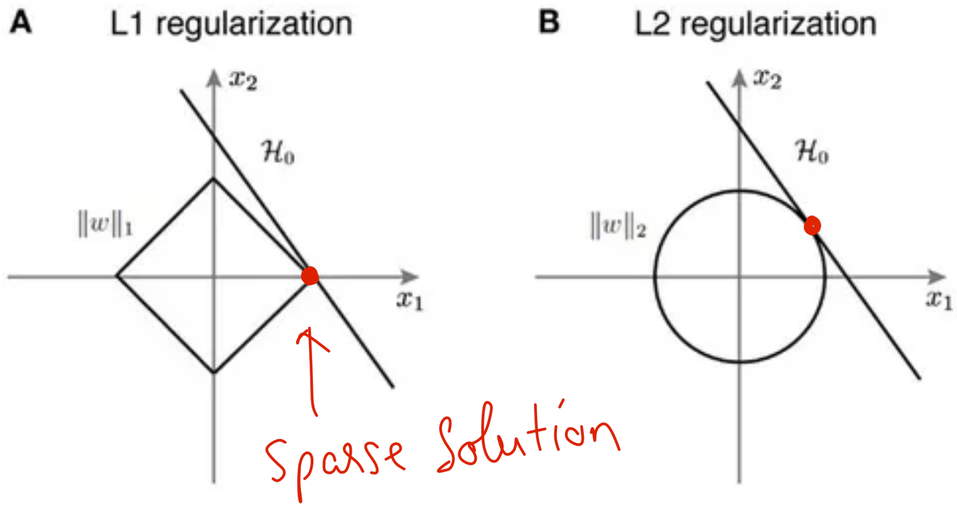
- Manhattan distance of weight vector θ from origin

- L1 regularized objective:

$$\hat{\theta} = \arg\max_{\theta} \left[\sum_{i=1}^{n} \log P(y^{(i)} | x^{(i)}) - \alpha \sum_{j=1}^{d} |\theta_j| \right]$$

L2 vs L1 regularization

- L2 is easier to optimize - simpler derivation
  - L1 is complex since the derivative of $|\theta|$ is not continuous at 0
- L2 leads to many small weights (due to θ^2 term)
  - L1 prefers *sparse* weight vectors with many weights set to 0 (i.e. far fewer features used)



Multinomial Logistic Regression

- What if we have more than 2 classes? (e.g. Part of speech tagging, named entity recognition)
- Need to model $P(y = c \mid x) \quad \forall c \in C$
- Generalize **sigmoid** function to **softmax**

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq k$$

Normalization

Softmax

- Similar to sigmoid, softmax squashes values towards 0 or 1
- If $z = [0,1,2,3,4]$, then
  - softmax$(z) = ([0.0117,0.0317,0.0861,0.2341,0.6364])$
- For multinomial LR,

$$P(y = c \mid x) = \frac{e^{\mathbf{w}_c \cdot \mathbf{x} + b_c}}{\sum_{j=1}^{k} e^{\mathbf{w}_j \cdot \mathbf{x} + b_j}}$$

Features in multinomial LR

- Features need to include both input (x) and class (c)
- There were implicit in binary case

Var	Definition	Wt
$f_1(0,x)$	1 if "!" $\in$ doc 0 otherwise	−4.5
$f_1(+,x)$	1 if "!" $\in$ doc 0 otherwise	2.6
$f_1(-,x)$	1 if "!" $\in$ doc 0 otherwise	1.3

Learning

- Generalize binary loss to multinomial CE loss:

$$L_{CE}(\hat{y}, y) = -\sum_{c=1}^{k} 1\{y = c\} \log P(y = c \mid x)$$

$$= -\sum_{c=1}^{k} 1\{y = c\} \log \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

Binary CE **Loss:**

$$-\log P(y \mid x) = -[y \log \hat{y} + (1 - y)\log(1 - \hat{y})]$$

- Gradient:

$$\frac{dL_{CE}}{dw_c} = -(1\{y = c\} - P(y = c \mid x))x$$

$$= -\left(1\{y = c\} - \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}\right) x$$