Language Technologies Institute

Carnegie Mellon University

Algorithms for NLP

CS 11-711 · Fall 2020

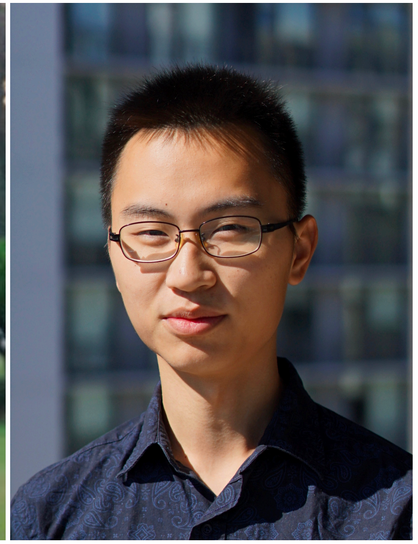
Lecture 2: Linear text classification

Emma Strubell

Let’s try this again…








Emma	Yulia	Bob	Sanket	Han	Jiateng
she/her	she/her	he/him	he/him	he/him	he/him

Outline

- Basic representations of text data for classification
- Four linear classifiers
  - Naïve Bayes
  - Perceptron
  - Large-margin (support vector machine; SVM)
  - Logistic regression

Text classification

Problem definition

- Given a text $\boldsymbol{w} = (w_1, w_2, \ldots, w_T) \in \mathcal{V}^*$
- Choose a label $y \in \mathcal{Y}$
- For example:
  - Sentiment analysis $\mathcal{Y} =$ { positive, negative, neutral }
  - Toxic comment classification $\mathcal{Y} =$ { toxic, non-toxic }
  - Language identification $\mathcal{Y} =$ { Mandarin, English, Spanish, … }

$\boldsymbol{w}$ = The drinks were strong but the fish tacos were bland y = negative

w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9 w_{10}

How to represent text for classification?

One choice of $\mathcal{R}$: bag-of-words

- Sequence length T can be different for every sentence/document
- The **bag-of-words** is a fixed-length vector of word counts:

$\boldsymbol{w}$ = The drinks were strong but the fish tacos were bland

$\boldsymbol{x}$ =

0	…	1	1	…	1	…	0	1	2	1	…	2	…	0
aardvark	…	bland	but	…	fish	…	taco	tacos	the	strong	…	were	…	zyther

- Length of $\boldsymbol{x}$ is equal to the size of the vocabulary, V
- For each $\boldsymbol{x}$ there may be many possible $\boldsymbol{w}$ (representation ignores word order)

Linear classification on bag-of-words

- Let $\psi(\boldsymbol{x}, y)$ score the compatibility of bag-of-words $\boldsymbol{x}$ and label y. Then:

$$\hat{y} = \underset{y}{\text{argmax}}\, \psi(\boldsymbol{x}, y).$$

- In a **linear classifier** this scoring function has the simple form:

$$\psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \sum_{j=1} \theta_j \times f_j(\boldsymbol{x}, y),$$

where $\boldsymbol{\theta}$ is a vector of weights, and $\boldsymbol{f}$ is a **feature function**

Feature functions

- In classification, the feature function is usually a simple combination of $\boldsymbol{x}$ and y, such as:

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

- If we have K labels, this corresponds to column vectors that look like:

$$\boldsymbol{f}(\boldsymbol{x}, y = 1) =$$

x_0	x_1	...	$x_{\lvert V\rvert}$	0	0	0	0	0	0	0	0	0	...	0

T

$$(K-1) \times V$$

Feature functions

- In classification, the feature function is usually a simple combination of $\boldsymbol{x}$ and y, such as:

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

- If we have K labels, this corresponds to column vectors that look like:

$$\boldsymbol{f}(\boldsymbol{x}, y = 1) =$$

x_0	x_1	...	$x_{\lvert V\rvert}$	0	0	0	0	0	0	0	0	0	...	0

T

$$\boldsymbol{f}(\boldsymbol{x}, y = 2) =$$

0	0	0	...	0	x_0	x_1	...	$x_{\lvert V\rvert}$	0	0	0	0	...	0

T

V $\qquad$ $(K-2)\times V$

Feature functions

- In classification, the feature function is usually a simple combination of $\boldsymbol{x}$ and y, such as:

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

- If we have K labels, this corresponds to column vectors that look like:

$\boldsymbol{f}(\boldsymbol{x}, y = 1) =$

x_0	x_1	**...**	$x_{\|V\|}$	0	0	0	0	0	0	0	0	0	…	0

T

$\boldsymbol{f}(\boldsymbol{x}, y = 2) =$

0	0	0	…	0	x_0	x_1	**...**	$x_{\|V\|}$	0	0	0	0	…	0

T

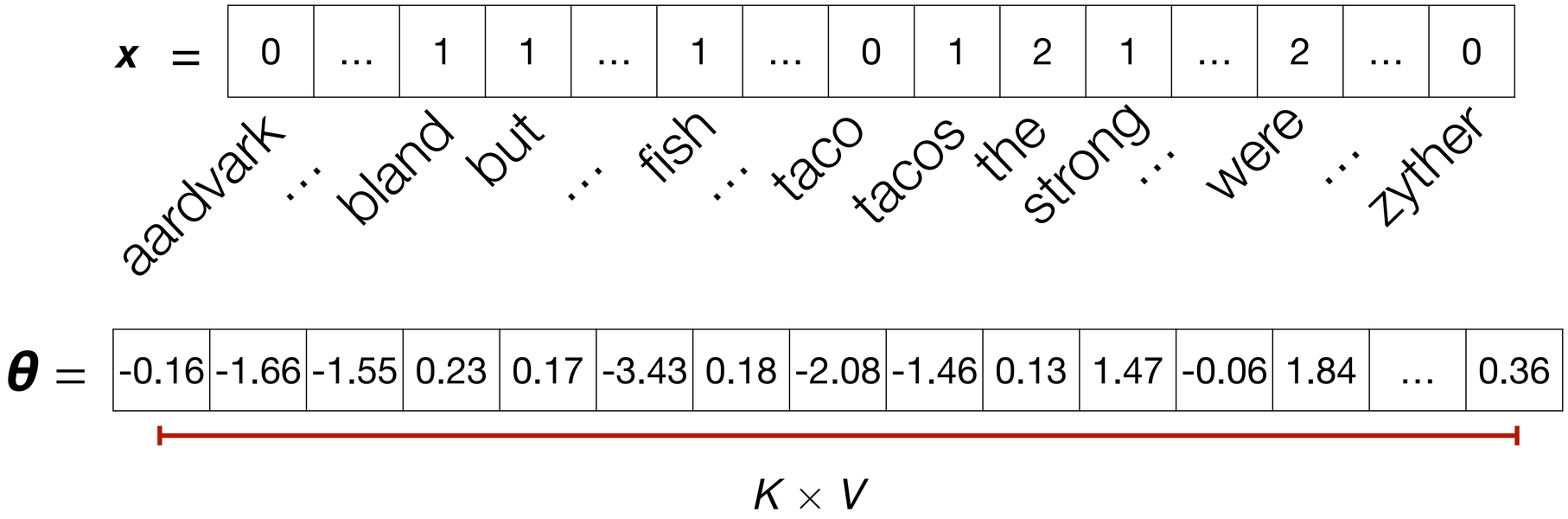
$\boldsymbol{f}(\boldsymbol{x}, y = K) =$

0	0	0	0	0	0	0	0	0	…	0	x_0	x_1	**...**	$x_{\|V\|}$

T

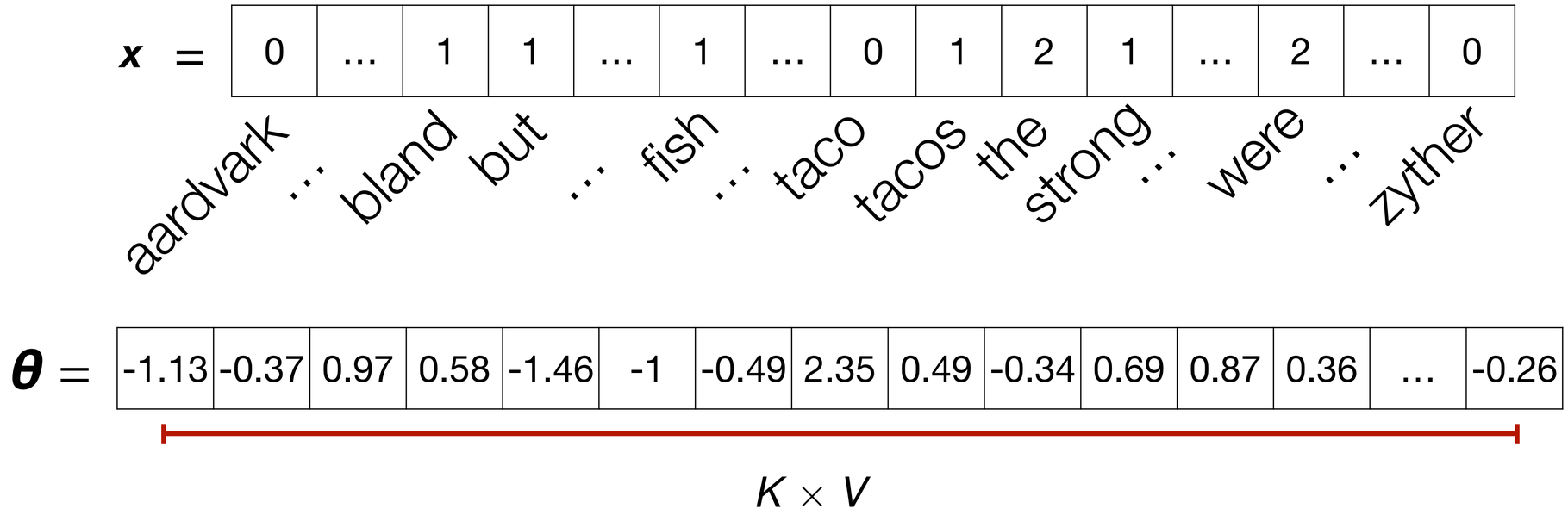
$(K-1) \times V$

Linear classification in Python




```
def compute_score(x, y, weights):
    total = 0
    for feature, count in feature_function(x, y).items():
        total += weights[feature] * count
    return total
```

Linear classification in Python




```
import numpy as np
def compute_score(x, y, weights):
    return np.dot(weights, feature_function(x, y))
```

Ok, but how to obtain θ ?

- The **learning** problem is to find the right weights θ.
- The rest of this lecture will cover four **supervised learning** algorithms:
  - Naïve Bayes
  - Perceptron
  - Large-margin (support vector machine)
  - Logistic regression
- All these methods assume a labeled dataset of N examples: $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$

Probabilistic classification

- Naïve Bayes is a probabilistic classifier. It takes the following strategy:
  - Define a **probability model** $p(\boldsymbol{x}, y)$
  - Estimate the parameters of the probability model by **maximum likelihood**, i.e. by maximizing the likelihood of the dataset
  - Set the scoring function equal to the log-probability:

$$\psi(\boldsymbol{x}, y) = \log \mathrm{p}(\boldsymbol{x}, y) = \log \mathrm{p}(y \mid \boldsymbol{x}) + C$$

where C is constant in y. This ensures that:

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, \mathrm{p}(y \mid \boldsymbol{x})$$

A probability model for text classification

- First, assume each instance ((***x***, *y*) pair) is independent of the others:

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

- Apply the chain rule of probability:

$$\mathrm{p}(\boldsymbol{x}, y) = \mathrm{p}(\boldsymbol{x} \mid y) \times \mathrm{p}(y)$$

- Define the parametric form of each probability:

$$\mathrm{p}(y) = \mathsf{Categorical}(\boldsymbol{\mu}) \qquad \mathrm{p}(\boldsymbol{x} \mid y) = \mathsf{Multinomial}(\boldsymbol{\phi}, T)$$

  - The **multinomial** is a distribution over vectors of counts
  - The **parameters** μ and ϕ are vectors of probabilities

The multinomial distribution

- Suppose the word *bland* has probability ϕ_j.
  What is the probability that this word appears 3 times?

- Each word's probability is exponentiated by its count:

$$\text{Multinomial}(\boldsymbol{x}; \boldsymbol{\phi}) = \prod_{j=1}^{V} \phi_j^{x_j}$$

- The coefficient is the count of the number of possible orderings of $\boldsymbol{x}$. Crucially, it does not depend on the frequency parameter ϕ.

Naïve Bayes text classification

- Naïve Bayes can be formulated in our linear classification framework by setting θ equal to the log parameters:

$\boldsymbol{\theta} =$

$\log \phi_{y1,w1}$	$\log \phi_{y1,w2}$	…	$\log \phi_{y1,wv}$	$\log \mu_{y1}$	$\log \phi_{y2,w1}$	…	$\log \phi_{y2,wv}$	$\log \mu_{y2}$	…	$\log \phi_{yk,wv}$	$\log \mu_{yk}$

$K \times (V + 1)$

$$\psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \log \mathrm{p}(\boldsymbol{x} \mid y) + \log \mathrm{p}(y)$$

where $\boldsymbol{f}(\boldsymbol{x}, y)$ is extended to include an "offset" 1 for each possible label after the word counts.

Estimating Naïve Bayes

- In relative frequency estimation, the parameters are set to empirical frequencies:

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

$$\hat{\mu}_y = \frac{\text{count}(y)}{\sum_{y'} \text{count}(y')}.$$

- This turns out to be identical to the maximum likelihood estimate (yay):

$$\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\mu}} = \underset{\boldsymbol{\phi}, \boldsymbol{\mu}}{\text{argmax}} \prod_{i=1}^{N} \text{p}(\boldsymbol{x}^{(i)}, y^{(i)}) = \underset{\boldsymbol{\phi}, \boldsymbol{\mu}}{\text{argmax}} \sum_{i=1}^{N} \log \text{p}(\boldsymbol{x}^{(i)}, y^{(i)})$$

Smoothing, bias, variance

- To deal with low counts, it can be helpful to smooth probabilities:

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

- Smoothing introduces **bias**, moving the parameters away from their maximum-likelihood estimates.
- But, it corrects **variance**, the extent to which the parameters depend on the idiosyncrasies of a finite dataset.
- The smoothing term α is a **hyperparameter** that must be tuned on a **development set**.

Too naïve?

- Naïve Bayes is so called because:
  - Bayes rule is used to convert the observation probability p(***x*** | *y*) into the label probability p(*y* | ***x***).
  - The multinomial distribution naïvely ignores dependencies between words, and treats each word as equally informative.

- **Discriminative** classifiers avoid this problem by not attempting to model the **generative** probability p(***x***).

The perceptron classifier

- A simple learning rule:
  - Run the current classifier on an instance in the training data, obtaining $\hat{y} = \textsf{argmax}_y \, \psi(\boldsymbol{x}^{(i)}, y)$
  - If the prediction is incorrect:
    1. Increase the weights for the features of the true label $y^{(i)}$
    2. Decrease the weights for the features of the predicted label $\hat{y}$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$$

  - Repeat until all training instances are correctly classified (or you run out of time)
- If the dataset is **linearly separable** — if there is some θ that correctly labels all the training instances — then this method is guaranteed to find it.

Loss functions

- Many classifiers can be viewed as minimizing a **loss function** on the weights. Such a function should have two properties:
  - It should be a good proxy for the accuracy of the classifier.
  - It should be easy to optimize.
- Do you see why 1 — accuracy is not a good loss function?

$$\ell_{\text{0-1}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \begin{cases} 0, & y^{(i)} = \text{argmax}_y \, \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y) \\ 1, & \text{otherwise} \end{cases}$$

Perceptron as gradient descent

- The perceptron can be viewed as optimizing the loss function:

$$\ell_{\text{perceptron}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = -\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \max_{y' \neq y^{(i)}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y')$$

- The gradient of the perceptron loss is part of the perceptron update:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ell_{\text{perceptron}} = -\boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$$

$$\begin{aligned} \boldsymbol{\theta}^{(t+1)} \leftarrow & \boldsymbol{\theta}^{(t)} - \frac{\partial}{\partial \boldsymbol{\theta}} \ell_{\text{perceptron}} \\ = & \boldsymbol{\theta}^{(t)} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y}) \end{aligned}$$

Gradient descent!

Perceptron vs. Naïve Bayes

- Both Naïve Bayes and perceptron loss functions are convex, making them relatively easy to optimize. However, NB can be optimized in closed form while perceptron requires iterating over the dataset multiple times.
- NB can suffer *infinite* loss on a single example, since the logarithm of 0 probability is -inf; some examples will be over-emphasized, others will be under-emphasized
- NB assumes the observed features are conditionally independent given the label; performance depends on the extent to which this holds.
- Perceptron treats all correct answers equally, even if θ only gives the correct answer by a very small margin, the loss is still 0.

Large margin learning

- For better generalization, the correct label should outscore all other labels by a large **margin**:

$$\gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \max_{y' \neq y^{(i)}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y')$$

- The margin can be incorporated into a **margin loss**:

$$\ell_{\textsc{Margin}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \begin{cases} 0, & \gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) \geq 1, \\ 1 - \gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}), & \text{otherwise} \end{cases}$$

$$= \max\left(0, 1 - \gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)})\right)$$

Large margin learning

- Margin loss can be minimized using a learning rule similar to perceptron. First, let's generalize the notion of classification error using a **cost function**:

$$c(y^{(i)}, y) = \begin{cases} 1, & y^{(i)} \neq \hat{y} \\ 0, & \text{otherwise}, \end{cases}$$

???

- Using the cost function, we define the **online support vector machine** (SVM) classification rule:

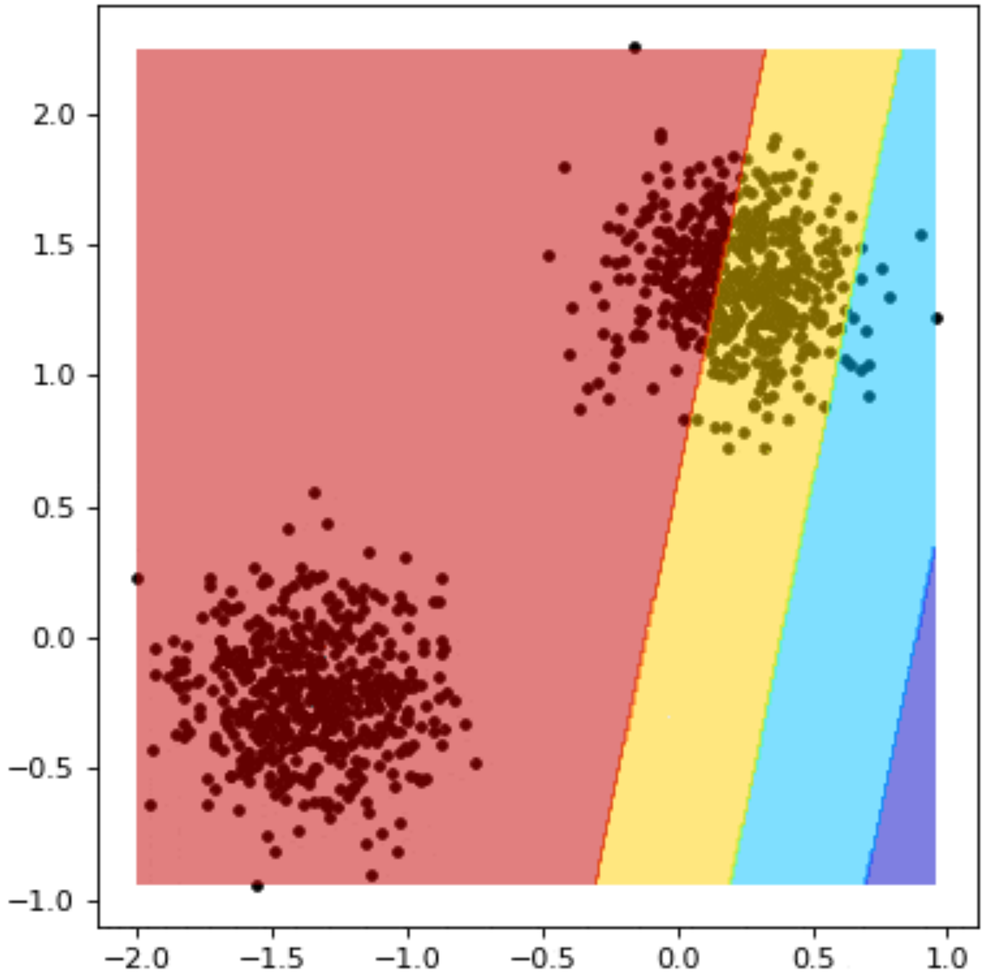
cost-augmented decoding

$$\hat{y} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y) + c(y^{(i)}, y)$$

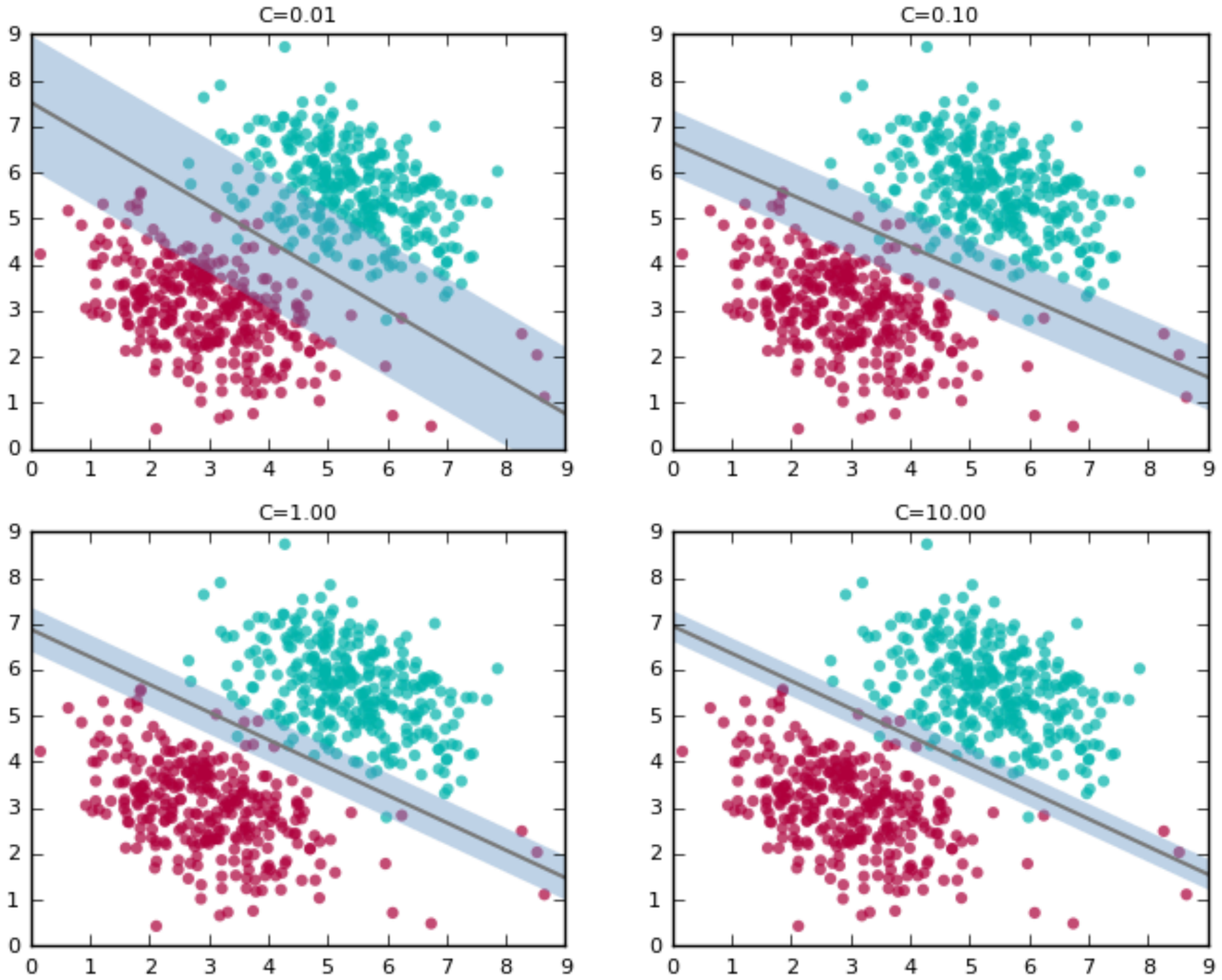
$$\boldsymbol{\theta}^{(t)} \leftarrow (1 - \lambda)\boldsymbol{\theta}^{(t-1)} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$$

regularization

Large margin learning



Large margin learning



Logistic regression

- Perceptron and large margin classification are **discriminative**: they learn to discriminate correct and incorrect labels.
- Naïve Bayes is **probabilistic**: it assigns calibrated confidence scores to its predictions.
- Logistic regression is both discriminative and probabilistic. It directly computes the conditional probability of the label:

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

  - Exponentiation ensures that the probabilities are non-negative.
  - Normalization ensures that the probabilities sum to one.

Learning logistic regression

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

- Two equivalent views of logistic regression learning:
  - **Maximization** of the conditional log-likelihood:

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

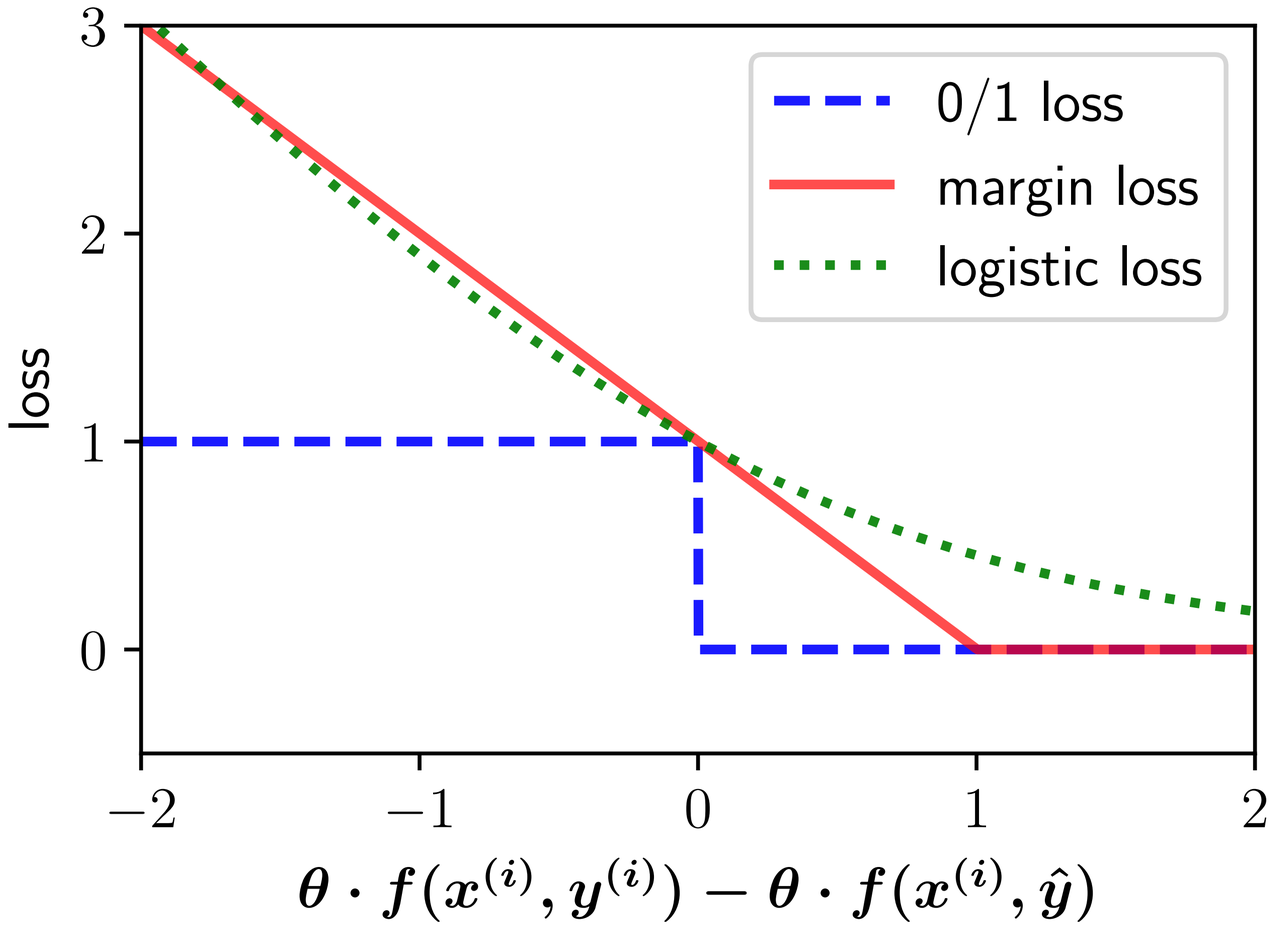
$$= \sum_{i=1}^{N} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \log \sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y')\right)$$

  - **Minimization** of the logistic loss:

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

(Compare to perceptron loss!)

Loss functions



High-dimensional classification

- Possible problems:
  - What if the number of features in $\boldsymbol{f}(\boldsymbol{x}, y)$ is larger than the number of training instances?
  - What happens in logistic regression if a feature appears only with one label? What will its weight be?
- These problems relate to the **variance** of the classifier — its sensitivity to idiosyncratic features of the training data.

Regularization

- Learning can often be made more robust by **regularization**: penalizing large weights. E.g.:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell_{\text{LOGREG}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) + \lambda ||\boldsymbol{\theta}||_2^2$$

  - $||\boldsymbol{\theta}||_2^2 = \sum_j \theta_j^2$
  - The scalar λ controls the strength of regularization (a hyperparameter).
  - The **support vector machine** classifier combines regularization with the large margin loss.

Regularization

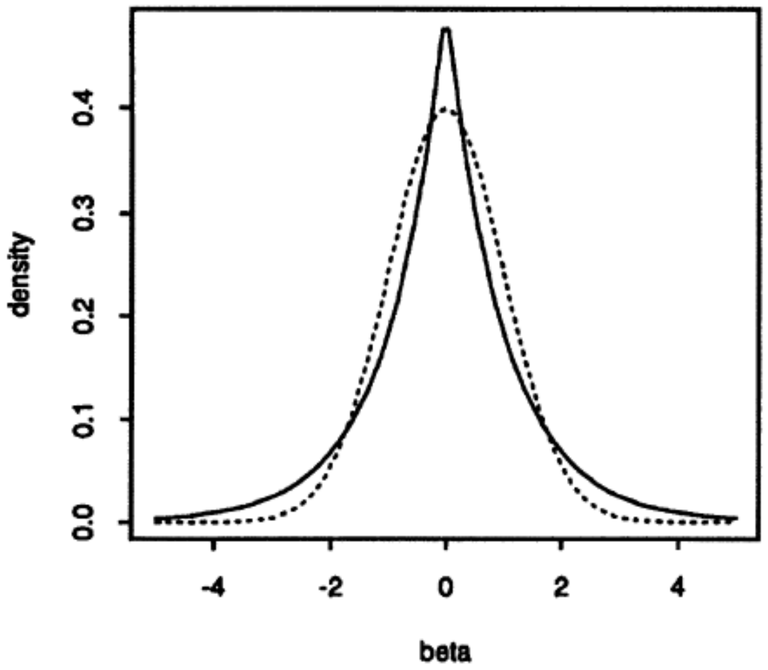


Fig. 7. Double-exponential density (———) and normal density (- - - - -): the former is the implicit prior used by the lasso; the latter by ridge regression

Gradient descent

- Logistic regression, perceptron and large margin classification all learn by minimizing a **loss function**. A general strategy for minimization is **gradient descent**:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{i=1}^{N} \ell(\boldsymbol{\theta}^{(t)}; \boldsymbol{x}^{(i)}, y^{(i)})$$

  - Where $\eta \in \mathbb{R}^+$ is the **learning rate**.

Stochastic gradient descent

- Computing the gradient over all instances (**batched**) is expensive.
- **Stochastic** gradient descent (SGD) approximates the gradient by its value on a single instance:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \sum_{i=1}^{N} \ell(\boldsymbol{\theta}^{(t)}; \boldsymbol{x}^{(i)}, y^{(i)}) \approx C \times \frac{\partial}{\partial \boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{(t)}; \boldsymbol{x}^{(i)}, y^{(i)})$$

  where ($\boldsymbol{x}^{(i)}$, $y^{(i)}$) is sampled at random from the training set.

- **Minibatch** gradient descent approximates the gradient by its value on a small number of instances. This is well suited to modern high-throughput hardware (e.g. GPUs and TPUs) and is commonly used in deep learning.

Linear classification: summary

	Pros	Cons
Naïve Bayes	simple, probabilistic, fast	not very accurate
Perceptron	simple, accurate	not probabilistic, may overfit
Large margin	error-driven learning, can be regularized	not probabilistic
Logistic regression	error-driven learning, regularized	more difficult to implement

Announcements

- Please fill out the survey posted to Piazza so that we can get to know you better! **Due next Friday (9/11).**
- No recitation this Friday 9/4. Please use office hours or post to Piazza if you have questions!
- Working on pushing the recitation back to start at 3pm to avoid current conflict w/ Colloquium.