NLP Programming Tutorial 6 - Advanced Discriminative Learning

Graham Neubig
Nara Institute of Science and Technology (NAIST)

Review: Classifiers and the Perceptron

Prediction Problems

Given x, predict y

Example we will use:

- Given an introductory sentence from Wikipedia
- Predict whether the article is about a person

Given		Predict
Gonso was a Sanron sect priest (754-827) in the late Nara and early Heian periods.	→	Yes!
Shichikuzan Chigogataki Fudomyoo is a historical site located at Magura, Maizuru City, Kyoto Prefecture.	→	No!

- This is binary classification

Mathematical Formulation

$$\begin{aligned} y &= \operatorname{sign}(\mathbf{w} \cdot \boldsymbol{\varphi}(x)) \\ &= \operatorname{sign}\left(\sum_{i=1}^{I} w_i \cdot \varphi_i(x)\right) \end{aligned}$$

- **x**: the input
- **φ(x)**: vector of feature functions $\{\varphi_1(x), \varphi_2(x), \ldots, \varphi_I(x)\}$
- **w**: the weight vector $\{w_1, w_2, \ldots, w_I\}$
- **y**: the prediction, +1 if “yes”, -1 if “no”
  - (sign(v) is +1 if v >= 0, -1 otherwise)

Online Learning

```
create map w
for I iterations
    for each labeled pair x, y in the data
        phi = create_features(x)
        y' = predict_one(w, phi)
        if y' != y
            update_weights(w, phi, y)
```

- In other words
    - Try to classify each training example
    - Every time we make a mistake, update the weights
- Many different online learning algorithms
    - The most simple is the perceptron

Perceptron Weight Update

$$\mathbf{w} \leftarrow \mathbf{w} + y\,\varphi(x)$$

- In other words:
  - If y=1, increase the weights for features in φ(x)
    - Features for positive examples get a higher weight
  - If y=-1, decrease the weights for features in φ(x)
    - Features for negative examples get a lower weight

→ Every time we update, our predictions get better!

```
update_weights(w, phi, y)
    for name, value in phi:
        w[name] += value * y
```

Stochastic Gradient Descent and Logistic Regression

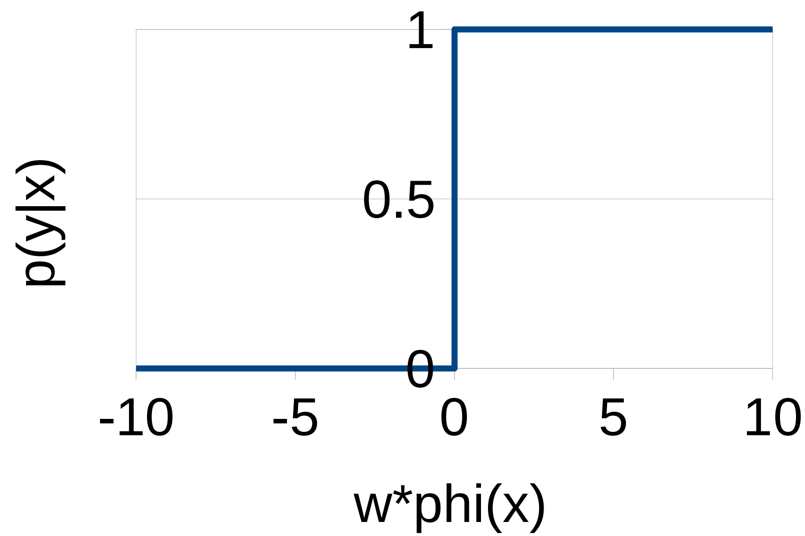
Perceptron and Probabilities

- Sometimes we want the probability $P(y|x)$
  - Estimating confidence in predictions
  - Combining with other systems
- However, perceptron only gives us a prediction

$$y = \text{sign}(w \cdot \varphi(x))$$

In other words:

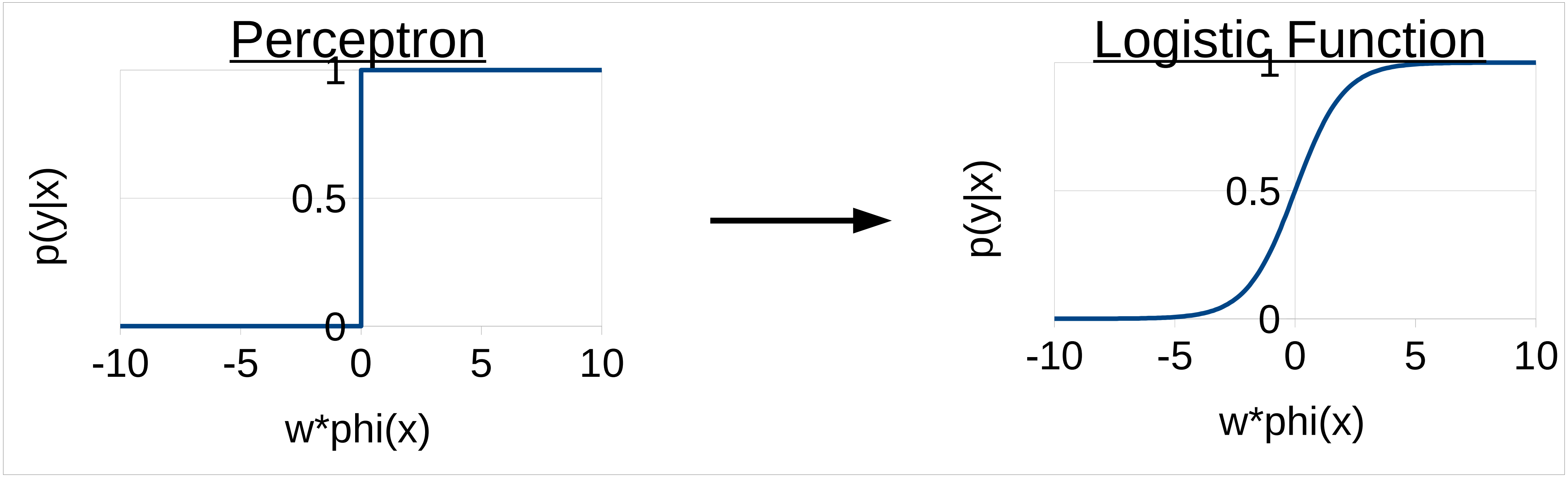
$$P(y=1|x) = 1 \text{ if } w \cdot \varphi(x) \geq 0$$
$$P(y=1|x) = 0 \text{ if } w \cdot \varphi(x) < 0$$



The Logistic Function

- The logistic function is a “softened” version of the function used in the perceptron

$$P(y=1|x)=\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$



- Can account for uncertainty
- Differentiable

Logistic Regression

- Train based on conditional likelihood
- Find the parameters **w** that maximize the conditional likelihood of all answers y_i given the example x_i

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \prod_i P(y_i \mid x_i; \boldsymbol{w})$$

- How do we solve this?

Stochastic Gradient Descent

- Online training algorithm for probabilistic models (including logistic regression)

```
create map w
for I iterations
    for each labeled pair x, y in the data
        w += α * dP(y|x)/dw
```

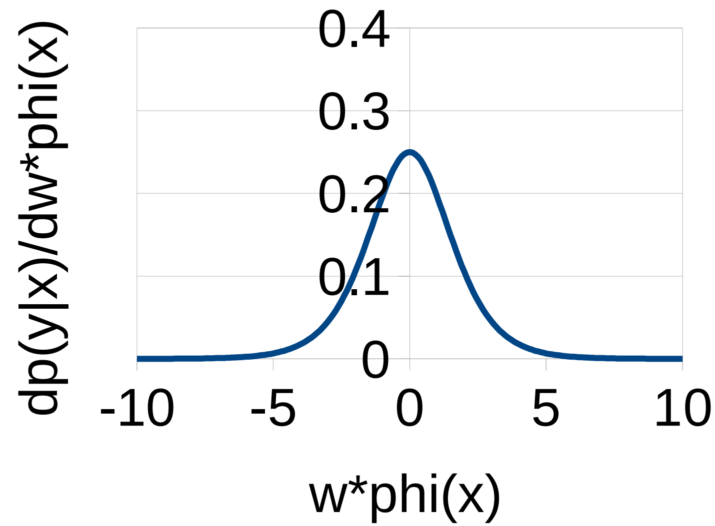
- In other words
  - For every training example, calculate the gradient (the direction that will increase the probability of y)
  - Move in that direction, multiplied by learning rate α

Gradient of the Logistic Function

- Take the derivative of the probability

$$\frac{d}{dw}P(y=1|x) = \frac{d}{dw}\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$

$$= \varphi(x)\frac{e^{w\cdot\varphi(x)}}{(1+e^{w\cdot\varphi(x)})^2}$$



$$\frac{d}{dw}P(y=-1|x) = \frac{d}{dw}\left(1-\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}\right)$$

$$= -\varphi(x)\frac{e^{w\cdot\varphi(x)}}{(1+e^{w\cdot\varphi(x)})^2}$$

Example: Initial Update

- Set α=1, initialize **w**=**0**

x = A site , located in Maizuru , Kyoto y = -1

$$\mathbf{w}\cdot\varphi(x)=0 \qquad \frac{d}{d\,w}P(y=-1|x) = -\frac{e^0}{(1+e^0)^2}\varphi(x)$$

$$= -0.25\,\varphi(x)$$

$$\mathbf{w} \leftarrow \mathbf{w} + -0.25\,\varphi(x)$$

$w_{\text{unigram “Maizuru”}}$ = -0.25
$w_{\text{unigram “,”}}$ = -0.5
$w_{\text{unigram “in”}}$ = -0.25
$w_{\text{unigram “Kyoto”}}$ = -0.25
$w_{\text{unigram “A”}}$ = -0.25
$w_{\text{unigram “site”}}$ = -0.25
$w_{\text{unigram “located”}}$ = -0.25

Example: Second Update

x = Shoken , monk born in Kyoto y = 1

-0.5 -0.25 -0.25

$$\boldsymbol{w}\cdot\varphi(x)=-1 \qquad \frac{d}{d\,\boldsymbol{w}}P(y=1|x) = \frac{e^{1}}{(1+e^{1})^{2}}\varphi(x)$$

$$= 0.196\,\varphi(x)$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + 0.196\,\varphi(x)$$

$w_{\text{unigram “Maizuru”}}$	= -0.25	$w_{\text{unigram “A”}}$	= -0.25	$w_{\text{unigram “Shoken”}}$	= 0.196
$w_{\text{unigram “,”}}$	= -0.304	$w_{\text{unigram “site”}}$	= -0.25	$w_{\text{unigram “monk”}}$	= 0.196
$w_{\text{unigram “in”}}$	= -0.054	$w_{\text{unigram “located”}}$	= -0.25	$w_{\text{unigram “born”}}$	= 0.196
$w_{\text{unigram “Kyoto”}}$	= -0.054				

SGD Learning Rate?

- How to set the learning rate α?
- Usually decay over time:

$$\alpha = \frac{1}{C + t}$$

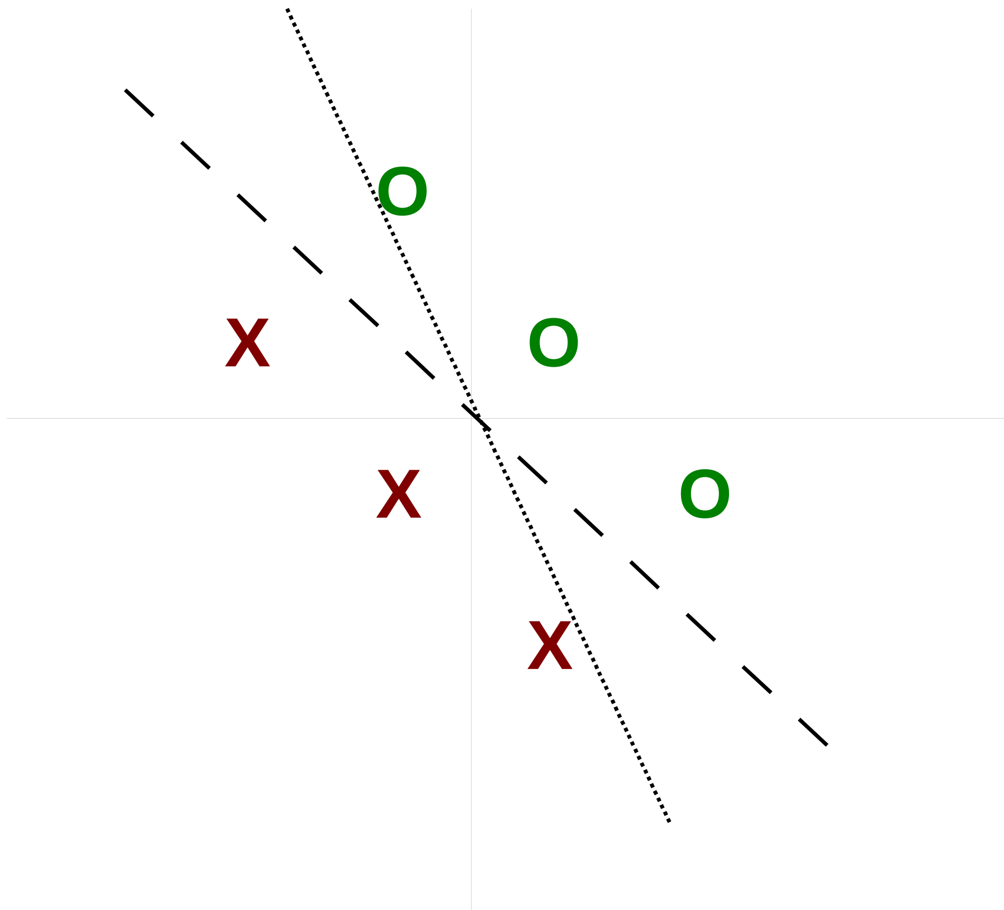
parameter (C) number of samples (t)

- Or, use held-out data, and reduce the learning rate when the likelihood rises

Classification Margins

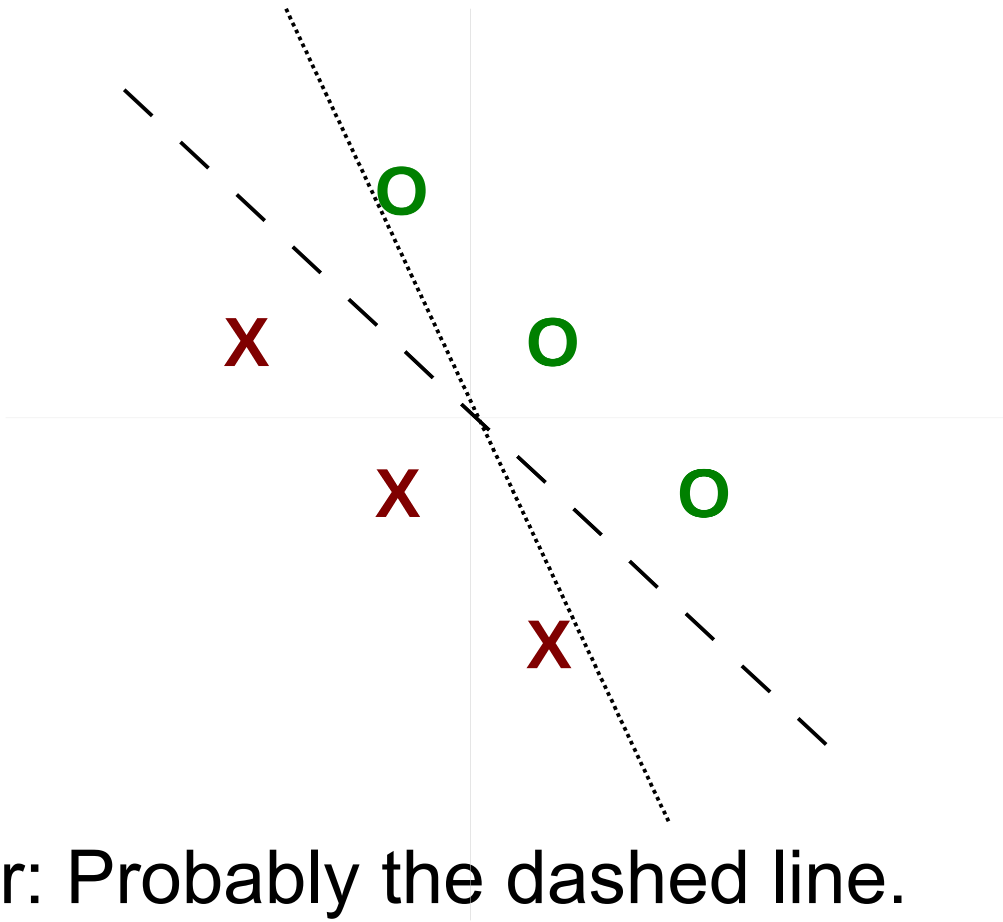
Choosing between Equally Accurate Classifiers

- Which classifier is better? Dotted or Dashed?



Choosing between Equally Accurate Classifiers

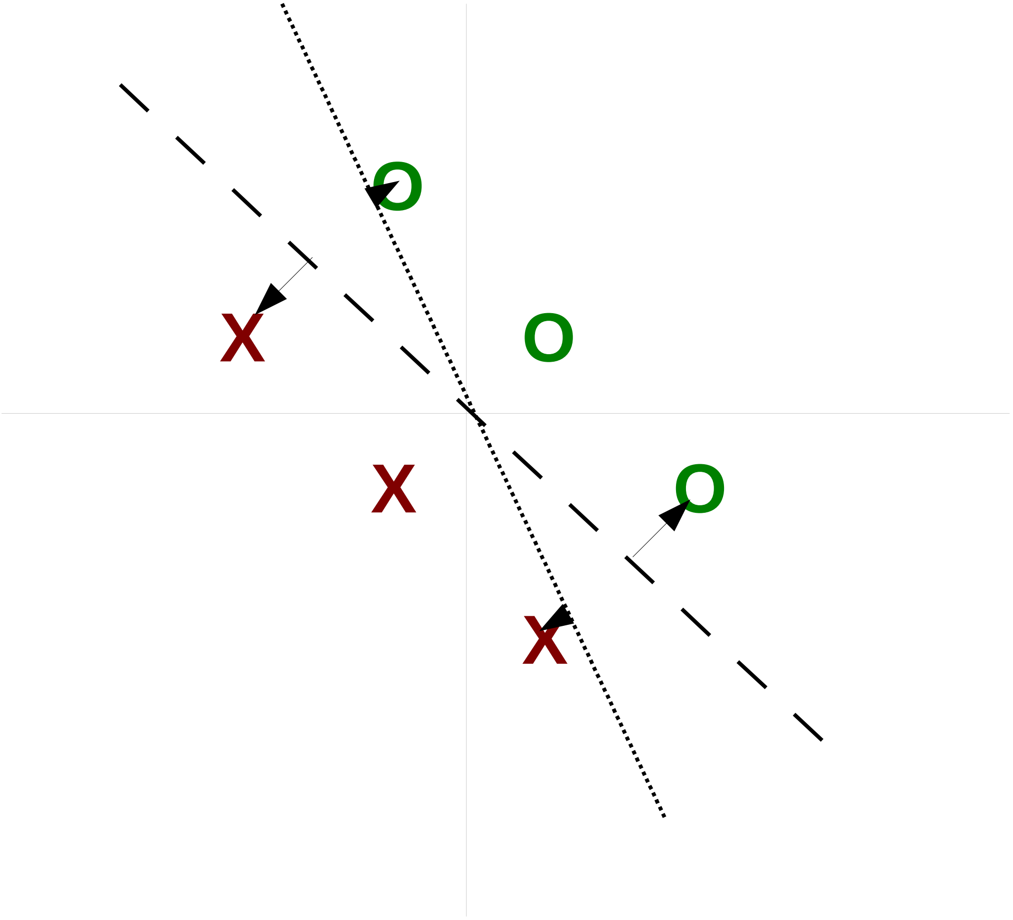
- Which classifier is better? Dotted or Dashed?



- Answer: Probably the dashed line.
- Why?: It has a larger margin.

What is a Margin?

- The distance between the classification plane and the nearest example:



Support Vector Machines

- Most famous margin-based classifier
  - Hard Margin: Explicitly maximize the margin
  - Soft Margin: Allow for some mistakes
- Usually use batch learning
  - Batch learning: slightly higher accuracy, more stable
  - Online learning: simpler, less memory, faster convergence
- Learn more about SVMs: http://disi.unitn.it/moschitti/material/Interspeech2010-Tutorial.Moschitti.pdf
- Batch learning libraries: LIBSVM, LIBLINEAR, SVMLite

Online Learning with a Margin

- Penalize not only mistakes, but also correct answers under a margin

```
create map w
for I iterations
    for each labeled pair x, y in the data
        phi = create_features(x)
        val = w * phi * y
★       if val <= margin
            update_weights(w, phi, y)
```

(A correct classifier will always make w * phi * y > 0)
If margin = 0, this is the perceptron algorithm

Regularization

Cannot Distinguish Between Large and Small Classifiers

- For these examples:

-1	he saw a bird in the park
+1	he saw a robbery in the park

- Which classifier is better?

Classifier 1	Classifier 2
he +3	bird -1
saw -5	robbery +1
a +0.5	
bird -1	
robbery +1	
in +5	
the -3	
park -2	

Cannot Distinguish Between Large and Small Classifiers

- For these examples:

-1	he saw a bird in the park
+1	he saw a robbery in the park

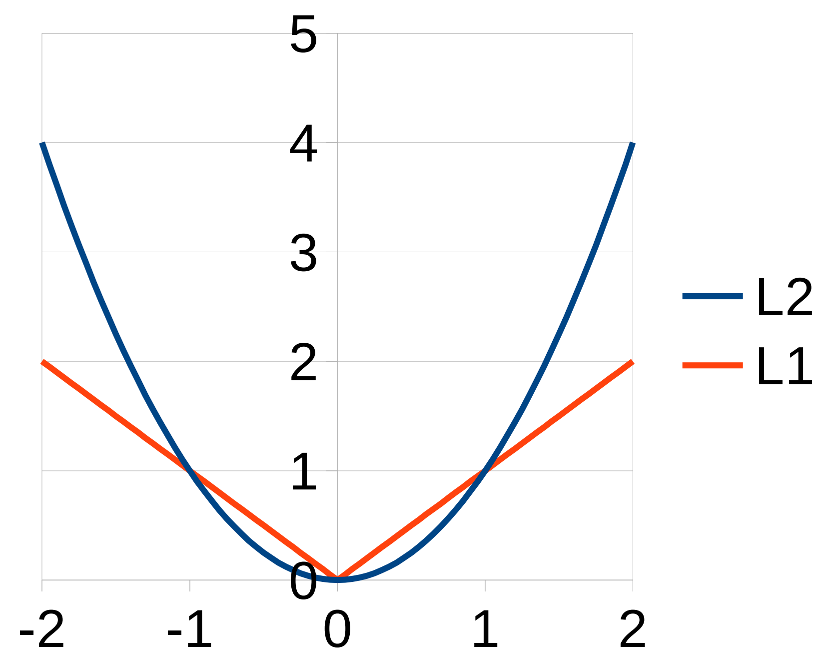
- Which classifier is better?

Classifier 1	Classifier 2
he +3	bird -1
saw -5	robbery +1
a +0.5	
bird -1	
robbery +1	
in +5	
the -3	
park -2	

Probably classifier 2!
It doesn't use
irrelevant information.

Regularization

- A penalty on adding extra weights
- L2 regularization:
  - Big penalty on large weights, small penalty on small weights
  - High accuracy
- L1 regularization:
  - Uniform increase whether large or small
  - Will cause many weights to become zero → small model



L1 Regularization in Online Learning

- After update, reduce the weight by a constant c

```
update_weights(w, phi, y, c)
★    for name, value in w:
★        if abs(value) < c:
★            w[name] = 0
★        else:
★            w[name] -= sign(value) * c
     for name, value in phi:
         w[name] += value * y
```

If abs. value < c,
set weight to zero

If value > 0,
decrease by c
If value < 0,
increase by c

Example

- Every turn, we Regularize, Update, Regularize, Update

Regularization: c=0.1
Updates: {1, 0} on 1st and 5th turns
{0, -1} on 3rd turn

	R_1	U_1	R_2	U_2	R_3	U_3
Change:	{0, 0}	{1, 0}	{-0.1, 0}	{0, 0}	{-0.1, 0}	{0, -1}
w:	{0, 0}	{1, 0}	{0.9, 0}	{0.9, 0}	{0.8, 0}	{0.8, -1}

	R_4	U_4	R_5	U_5	R_6	U_6
Change:	{-0.1, 0.1}	{0, 0}	{-0.1, 0.1}	{1, 0}	{-0.1, 0.1}	{0, 0}
w:	{0.7, -0.9}	{0.7, -0.9}	{0.6, -0.8}	{1.6, -0.8}	{1.5, -0.7}	{1.5, -0.7}

Efficiency Problems

- Typical number of features:
  - Each sentence (phi): 10~1000
  - Overall (w): 1,000,000~100,000,000

```
update_weights(w, phi, y, c)
    for name, value in w:
        if abs(value) <= c:
            w[name] = 0
        else:
            w[name] -= sign(value) * c
    for name, value in phi:
        w[name] += value * y
```

This loop is VERY SLOW!

Efficiency Trick

- Regularize only when the value is used!

```
getw(w, name, c, iter, last)
    if iter != last[name]:            # regularize several times
        c_size = c * (iter - last[name])
        if abs(w[name]) <= c_size:
            w[name] = 0
        else:
            w[name] -= sign(w[name]) * c_size
        last[name] = iter
    return w[name]
```

- This is called “lazy evaluation”, used in many applications

Choosing the Regularization Constant

- The regularization constant c has a large effect
- Large value
  - small model
  - lower score on training set
  - less overfitting
- Small value
  - large model
  - higher score on training set
  - more overfitting
- Choose best regularization value on development set
  - e.g. 0.0001, 0.001, 0.01, 0.1, 1.0

Exercise

Exercise

- Write program:
  - train-svm/train-lr: Create an svm or LR model with L2 regularization constant 0.001
- Train a model on data-en/titles-en-train.labeled
- Predict the labels of data-en/titles-en-test.word
- Grade your answers and compare them with the perceptron
  - script/grade-prediction.py data-en/titles-en-test.labeled your_answer
- Extra challenge:
  - Try many different regularization constants
  - Implement the efficiency trick

Thank You!