COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Supervised Classification with the Perceptron

CMSC 470

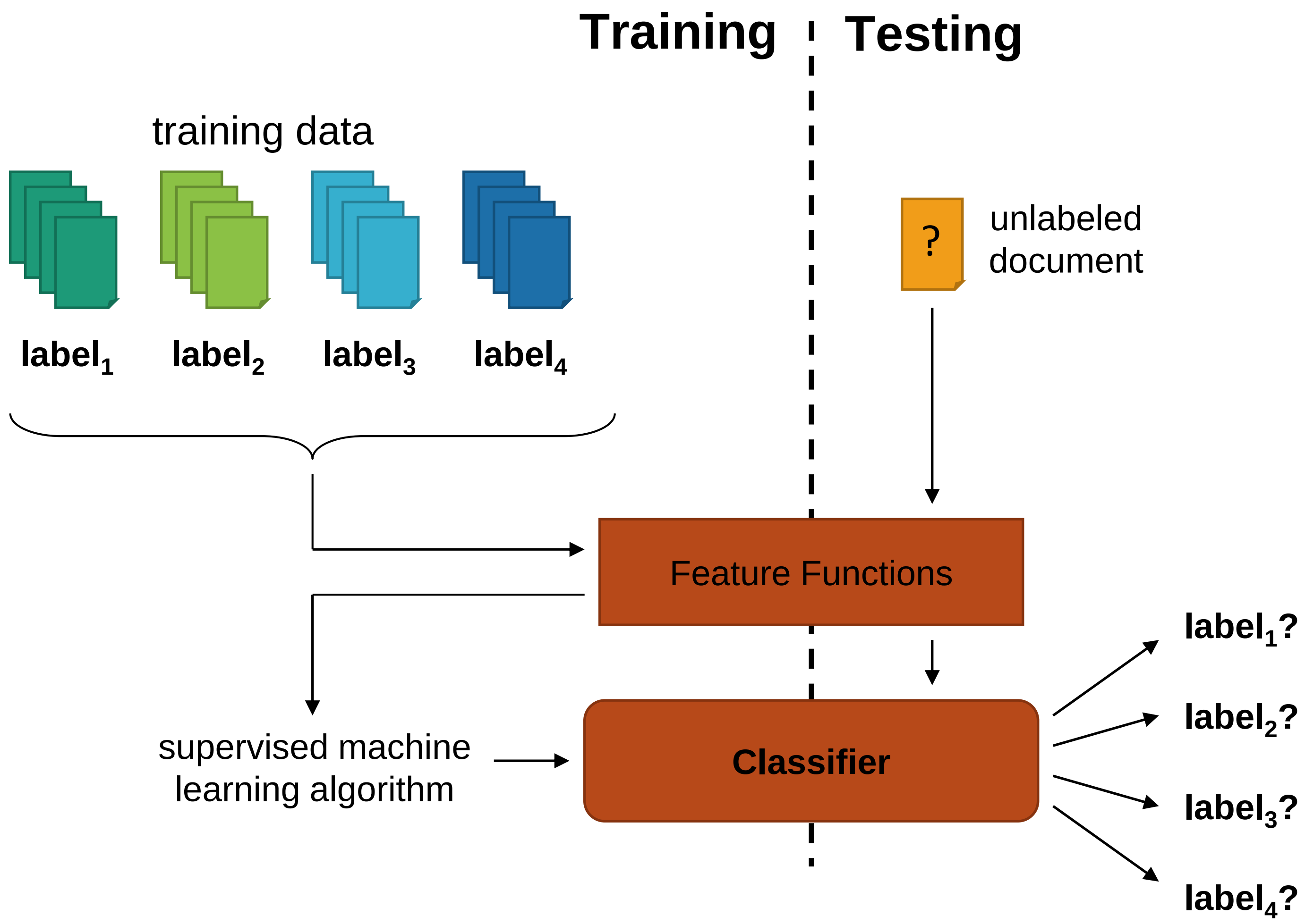
Marine Carpuat

Slides credit: Hal Daume III & Piyush Rai

Last time

- Word senses distinguish different meanings of same word
- Sense inventories
- Annotation issues and annotator agreement (Kappa)
- Definition of Word Sense Disambiguation Task
- An unsupervised approach: Lesk algorithm
- Supervised classification:
  - Train vs. test data
  - The most frequent class baseline
- Evaluation metrics: accuracy, precision, recall

WSD as **Supervised** Classification



Evaluation Metrics for Classification

How are annotated examples used in supervised learning?

- Supervised learning = requires examples annotated with correct prediction

- Used in 2 ways:
  - To find good values for the model (hyper)parameters (training data)
  - To evaluate how good the resulting classifier is (test data)

- How do we know how good a classifier is?
  - Compare classifier predictions with human annotation
  - On held out test examples
  - Evaluation metrics: accuracy, precision, recall

Quantifying Errors in a Classification Task: The 2-by-2 contingency table (per class)

	correct	not correct
selected	tp	fp
not selected	fn	tn

Quantifying Errors in a Classification Task: Precision and Recall

	correct	not correct
selected	tp	fp
not selected	fn	tn

Precision: % of selected items that are correct
Recall: % of correct items that are selected

Q: When are Precision/Recall more informative than accuracy?

A combined measure: F

- A combined measure that assesses the P/R tradeoff is F measure (weighted harmonic mean):

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \qquad \text{With } \beta^2 = \frac{1}{\alpha} - 1$$

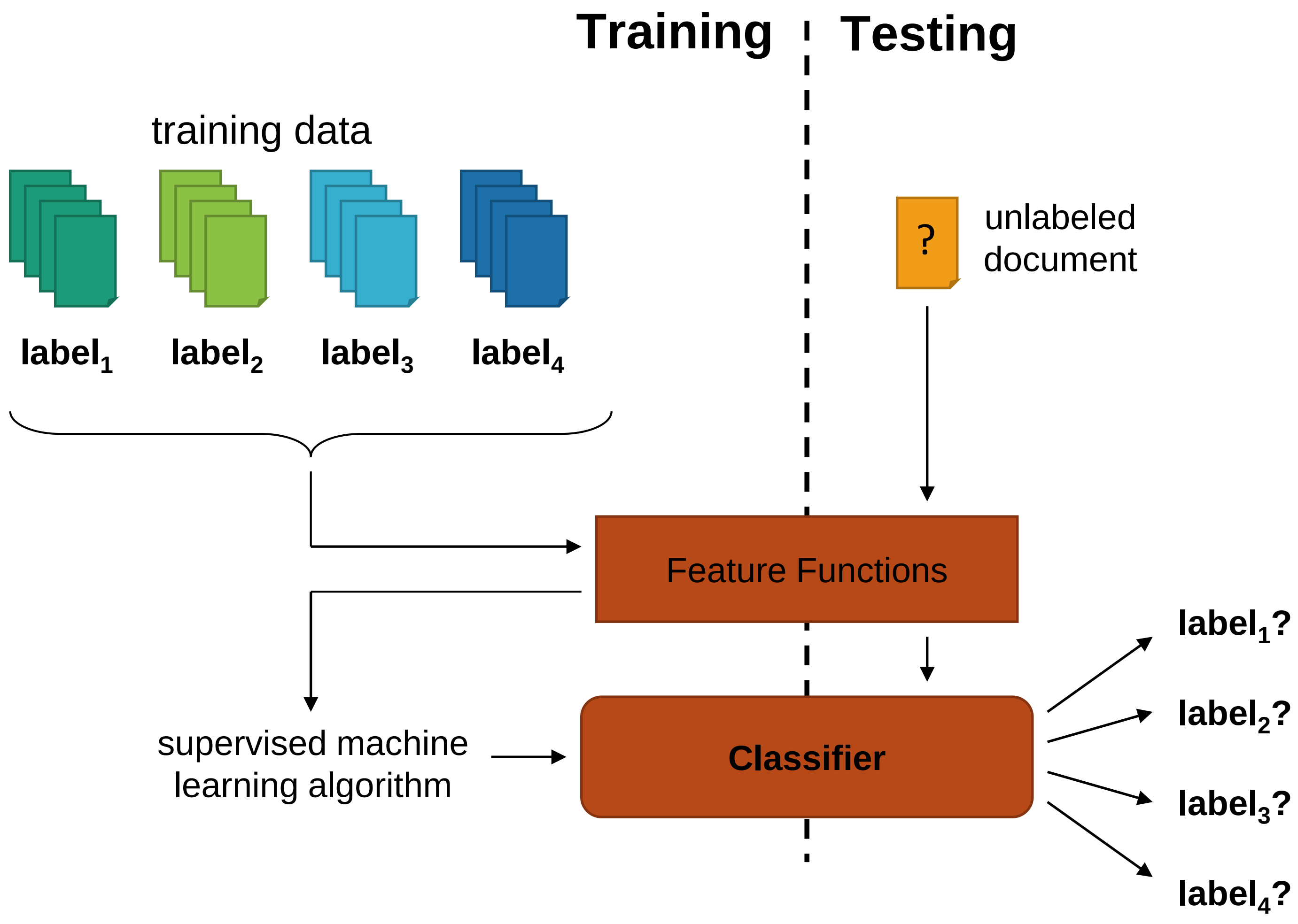
- People usually use balanced F1 measure
  - i.e., with $\beta = 1$ (that is, $\alpha = ½$):

    $F = 2PR/(P+R)$

The Perceptron

A simple Supervised Classifier

WSD as **Supervised** Classification



Formalizing classification

Task definition

- *Given inputs*:
  - an example x

    often x is a D-dimensional vector of binary or real values
  - a fixed set of classes Y

    $Y = \{y_1, y_2, \ldots, y_J\}$

    e.g. word senses from WordNet
- *Output*: a predicted class $y \in Y$

Classifier definition

A function $f: x \rightarrow f(x) = y$

Many different types of functions/classifiers can be defined

- We'll talk about perceptron, logistic regression, neural networks.

Example: Word Sense Disambiguation for "bass"

WordNet Sense	Spanish Translation	Roget Category	Target Word in Context
$bass^4$	lubina	FISH/INSECT	...fish as Pacific salmon and striped **bass** and...
$bass^4$	lubina	FISH/INSECT	...produce filets of smoked **bass** or sturgeon...
$bass^7$	bajo	MUSIC	...exciting jazz **bass** player since Ray Brown...
$bass^7$	bajo	MUSIC	...play **bass** because he doesn't have to solo...

- Y = {-1,+1} since there are 2 senses in our inventory
- Many different definitions of x are possible
  - E.g., vector of word frequencies for words that co-occur in a window of +/- k words around "bass"
  - Instead of frequency, we could use binary values, or tf.idf, or PPMI, etc.
  - Instead of window, we could use the entire sentence
  - Instead of/in addition to words, we could use POS tags
  - …

Perception Test Algorithm for Binary Classification: Predict class -1 or +1 for example x

f(x) = sign(w.x + b)

Algorithm 6 PerceptronTest($w_0, w_1, \ldots, w_D, b, \hat{\mathbf{x}}$)

$a \leftarrow \sum_{d=1}^{D} w_d \, \hat{x}_d + b$ // compute activation for the test example

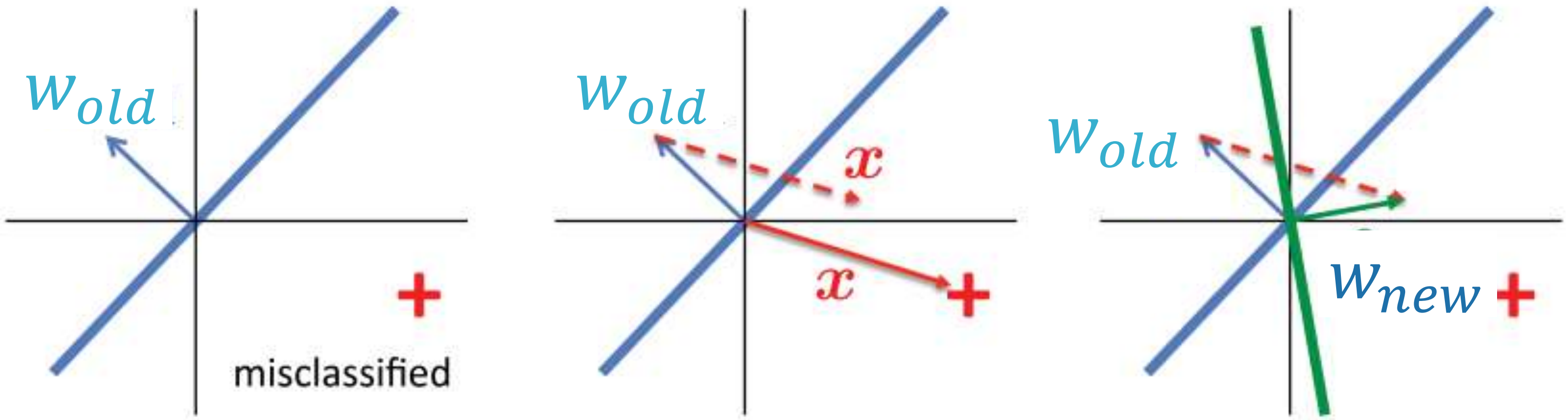
return SIGN(a)

Perceptron Training Algorithm:
Find good values for (w,b) given training data D

Algorithm 5 PERCEPTRONTRAIN($\mathbf{D}$, *MaxIter*)

$w_d \leftarrow 0$, for all $d = 1 \ldots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \ldots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d\, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + yx_d$, for all $d = 1 \ldots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \ldots, w_D, b$

The Perceptron update rule: geometric interpretation



Machine Learning Vocabulary

x is often called the feature vector

- its elements are defined (by us, the model designers) to capture properties or features of the input that are expected to correlate with predictions

w and b are the parameters of the classifier

- they are needed to fully define the classification function f(x) = y
- their values are found by the training algorithm using training data D

MaxIter is a hyperparameter

- controls when training stops
- MaxIter impacts the nature of function f indirectly

All of the above affect the performance of the final classifier!

Standard Perceptron: predict based on final parameters

Algorithm 5 PERCEPTRONTRAIN(**D**, *MaxIter*)

$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + y x_d$, for all $d = 1 \dots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \dots, w_D, b$

Predict based on final + intermediate parameters

- The voted perceptron

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(\text{k})}\text{sign}\left(\boldsymbol{w}^{(\text{k})} \cdot \hat{\boldsymbol{x}} + b^{(\text{k})}\right)\right)$$

- The averaged perceptron

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(\text{k})}\left(\boldsymbol{w}^{(\text{k})} \cdot \hat{\boldsymbol{x}} + b^{(\text{k})}\right)\right)$$

- Require keeping track of “survival time” of weight vectors

$$c^{(1)}, \ldots, c^{(\text{K})}$$

How would you modify this algorithm for voted perceptron?

Algorithm 5 PerceptronTrain($\mathbf{D}$, *MaxIter*)

$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d\, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + yx_d$, for all $d = 1 \dots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \dots, w_D, b$

How would you modify this algorithm for averaged perceptron?

Algorithm 5 PERCEPTRONTRAIN($\mathbf{D}$, $MaxIter$)

$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + yx_d$, for all $d = 1 \dots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \dots, w_D, b$

Averaged perceptron decision rule

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(\text{k})} \left(\boldsymbol{w}^{(\text{k})} \cdot \hat{\boldsymbol{x}} + b^{(\text{k})}\right)\right)$$

can be rewritten as

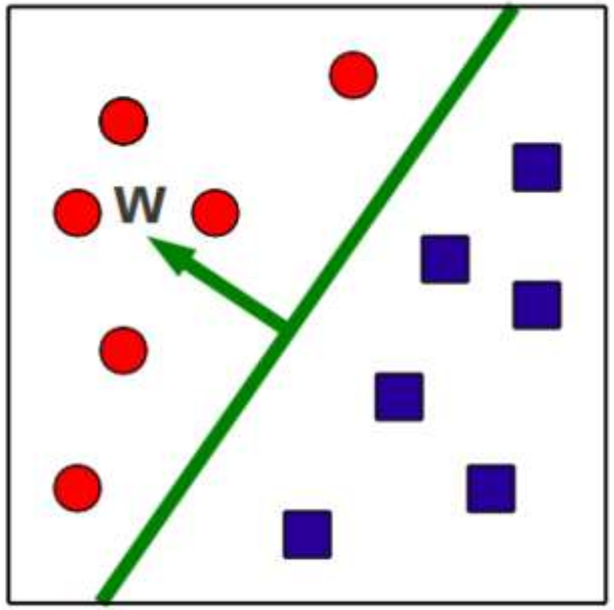
$$\hat{y} = \text{sign}\left(\left(\sum_{k=1}^{K} c^{(\text{k})} \boldsymbol{w}^{(\text{k})}\right) \cdot \hat{\boldsymbol{x}} + \sum_{k=1}^{K} c^{(\text{k})} b^{(\text{k})}\right)$$

An Efficient Algorithm for Averaged Perceptron Training

Algorithm 7 AveragedPerceptronTrain($\mathbf{D}$, *MaxIter*)

$\boldsymbol{w} \leftarrow \langle 0, 0, \dots 0 \rangle$, $b \leftarrow 0$ // initialize weights and bias
$\boldsymbol{u} \leftarrow \langle 0, 0, \dots 0 \rangle$, $\beta \leftarrow 0$ // initialize cached weights and bias
$c \leftarrow 1$ // initialize example counter to one
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    if $y(\boldsymbol{w} \cdot \boldsymbol{x} + b) \leq 0$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\,\boldsymbol{x}$ // update weights
      $b \leftarrow b + y$ // update bias
      $\boldsymbol{u} \leftarrow \boldsymbol{u} + y\,c\,\boldsymbol{x}$ // update cached weights
      $\beta \leftarrow \beta + y\,c$ // update cached bias
    end if
    $c \leftarrow c + 1$ // increment counter regardless of update
  end for
end for
return $\boldsymbol{w} - \frac{1}{c}\boldsymbol{u}, b - \frac{1}{c}\beta$ // return averaged weights and bias

Perceptron for binary classification



- Classifier = a hyperplane that separates positive from negative examples

$$\hat{y} = sign(w.x + b)$$

- Perceptron training
  - Finds such a hyperplane
  - If training examples are separable

Convergence of Perceptron

Theorem (Block & Novikoff, 1962)

If the training data $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ is **linearly separable** with margin γ by a unit norm hyperplane w_* ($||w_*|| = 1$) with $b = 0$,

Then **perceptron training converges after $\boldsymbol{\frac{R^2}{\gamma^2}}$ errors** during training

(assuming ($||x|| < R$) for all x).

More Machine Learning vocabulary: overfitting/underfitting/generalization

Training error is not sufficient

- We care about **generalization** to new examples

- A classifier can classify training data perfectly, yet classify new examples incorrectly
  - Because training examples are only a sample of data distribution
    - a feature might correlate with class by coincidence
  - Because training examples could be noisy
    - e.g., accident in labeling

Overfitting

- Consider a model θ and its:
  - Error rate over training data $error_{train}(\theta)$
  - True error rate over all data $error_{true}(\theta)$

- We say h overfits the training data if

$$error_{train}(\theta) < error_{true}(\theta)$$

Evaluating on test data

- Problem: we don't know $error_{true}(\theta)$!

- Solution:
  - we set aside a test set
    - some examples that will be used for evaluation
  - we don't look at them during training!
  - after learning a classifier θ, we calculate

$$error_{test}(\theta)$$

Overfitting

- Another way of putting it

- A classifier θ is said to overfit the training data, if there are other parameters θ', such that
  - θ has a smaller error than θ' on the training data
  - but θ has larger error on the test data than θ'.

Underfitting/Overfitting

- Underfitting
  - Learning algorithm had the opportunity to learn more from training data, but didn’t
- Overfitting
  - Learning algorithm paid too much attention to idiosyncracies of the training data; the resulting classifier doesn’t generalize

Back to the Perceptron

- Practical strategies to improve generalization for the perceptron
  - Voting/Averaging
  - Randomize order of training data
  - Use a development test set to find good hyperparameter values
    - E.g., early stopping is a good strategy to avoid overfitting

The Perceptron
What you should know

- What is the underlying function used to make predictions
- Perceptron test algorithm
- Perceptron training algorithm
- How to improve perceptron training with the averaged perceptron
- Fundamental Machine Learning Concepts:
  - train vs. test data; parameter; hyperparameter; generalization; overfitting; underfitting.
- How to define features