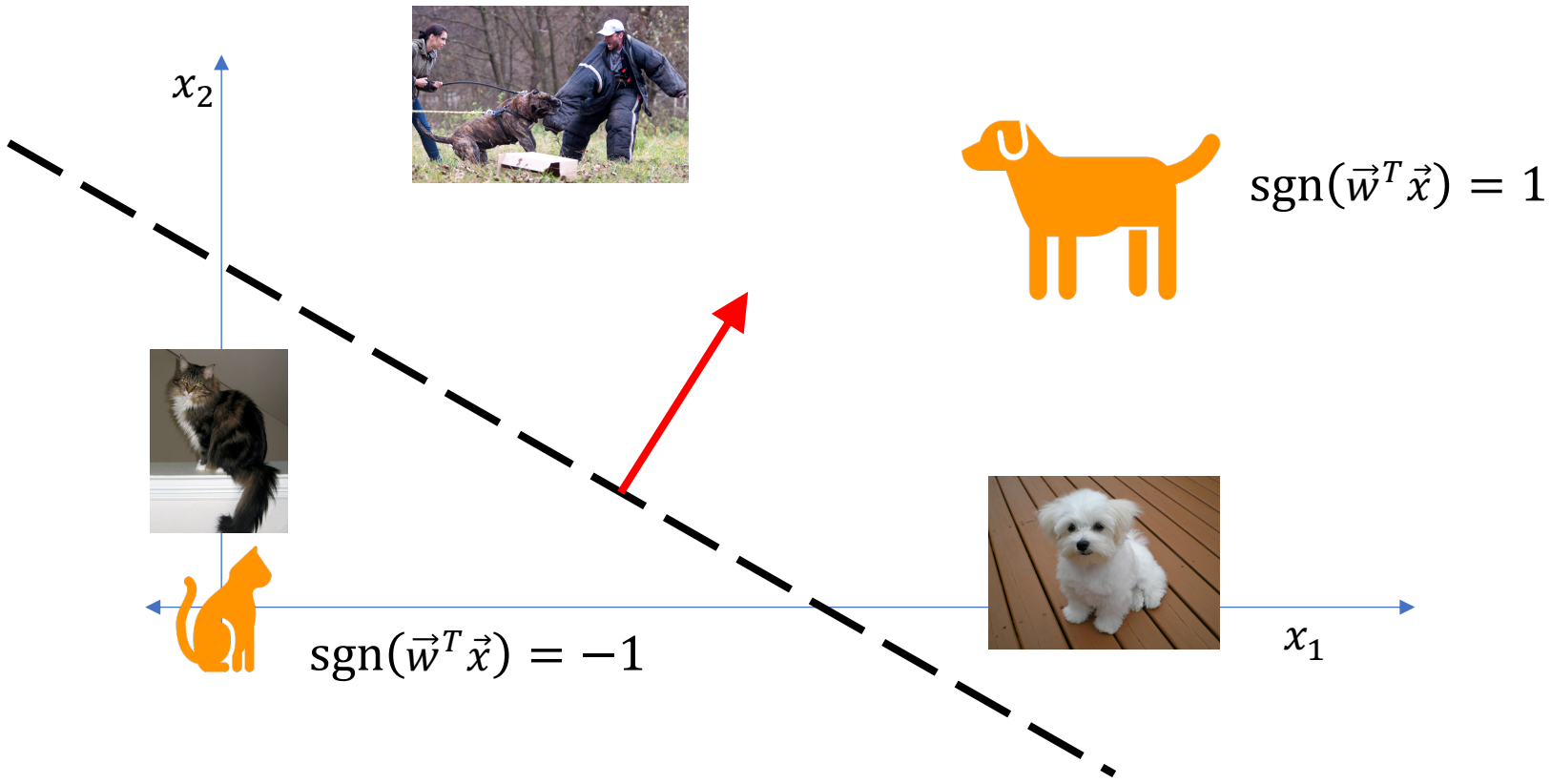
CS440/ECE448 Lecture 25: Classifiers

Mark Hasegawa-Johnson, 4/2020



Classifiers

- Many types of classifiers
  - Naive Bayes, Bayesian Network
  - K-Nearest-Neighbors
  - Linear Classifiers: Perceptron, Logistic Regression
- How good is your classifier?
  - Train, Dev, and Test corpora
  - Confusion Matrix; Precision and Recall
- Review: Perceptron and Logistic Regression
  - Signed and unsigned classification functions

Classifiers

- Many types of classifiers
  - Naive Bayes, Bayesian Network
  - K-Nearest-Neighbors
  - Linear Classifiers: Perceptron, Logistic Regression
- How good is your classifier?
  - Train, Dev, and Test corpora
  - Confusion Matrix; Precision and Recall
- Review: Perceptron and Logistic Regression
  - Signed and unsigned classification functions

Definition of “Classifier”

A classifier is a function:

- Input = a set of features (observations about some object)
- Output = a class label

Example: Naïve Bayes

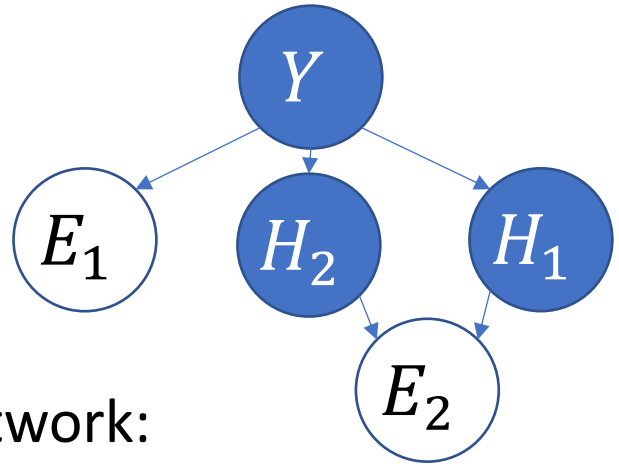
- Class label = Y, drawn from some set of labels
- Features = $E_1, \ldots, E_D$
- Classification function: output the label, Y, which maximizes $P(Y, E_1, \ldots, E_D)$ under the following "naïve Bayes" assumption:

$$P(Y, E_1, \ldots, E_D) = P(Y) \prod_{d=1}^{D} P(E_d|Y)$$

Example: Bayesian Network

- Class Label = one of the variables in the network, Y
- Features = the set of variables that are observed, $E_1, \ldots, E_D$
- Classification function: output the label, Y, which maximizes $P(Y, E_1, \ldots, E_D)$ under the assumption that this distribution is given by the structure of the network.

An Example Bayesian Network
with some observed variables E_1 and E_2,
a hidden class variable , Y,
and some other hidden variables, H_1 and H_2:



The model specified by this Bayesian network:

$$P(Y, E_1, E_2) = \sum_{H_1} \sum_{H_2} P(Y)P(E_1|Y)P(H_1|Y)P(H_2|Y)P(E_2|H_1, H_2)$$

Nearest Neighbors Classifier

- Given n different training examples, x_i, for $1 \leq i \leq n$ (x_i might be a vector, or a visible object, or a word, or whatever). Each one has a corresponding class label, y_i =correct_label(x_i).

- Input to the classifier: a test token x whose correct label is unknown.
- Classification function:
  1. Find the training token x_i that is most similar to the test token.
  2. Find out the corresponding class label, y_i =correct_label(x_i).
  3. Output y_i as the best guess for the label of test token x.

Example of Nearest-Neighbor Classification

Test Token: Maltese

This is the most similar training token…

Therefore the Maltese is classified as a dog.

Training Tokens:

By DK1k - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=77847428

K-Nearest Neighbors (KNN) Classifier

The nearest-neighbors classifier sometimes fails if one of the training tokens is unusual. In that case, a test token that is similar to the weird training token might get misclassified. Solution: K-Nearest Neighbors.

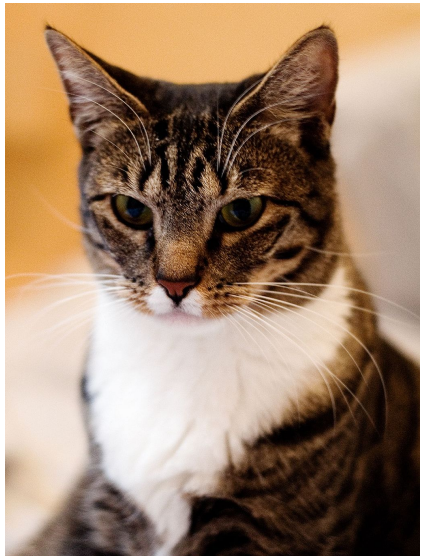
By Mandruss - This file was derived from: Maine Coon female 2.jpg, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=79389146

K-Nearest Neighbors (KNN) Classifier

By DK1k - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=77847428

By Mandruss - This file was derived from: Maine Coon female 2.jpg, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=79389146

By Mike Powell from United States - Stephanie 2, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=6114385



By Dustin Warrington - Flickr, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=2645977

K-Nearest Neighbors Classification Function

1. Find the K training tokens, x_i, that are most similar to the test token (K is a number chosen in advance by the system designer, e.g., $K = 3$).
2. Find out the corresponding class labels, y_i =correct_label(x_i).
3. Vote! Find the class label that is most frequent among the K-nearest neighbors, and output that as the label of the test token.

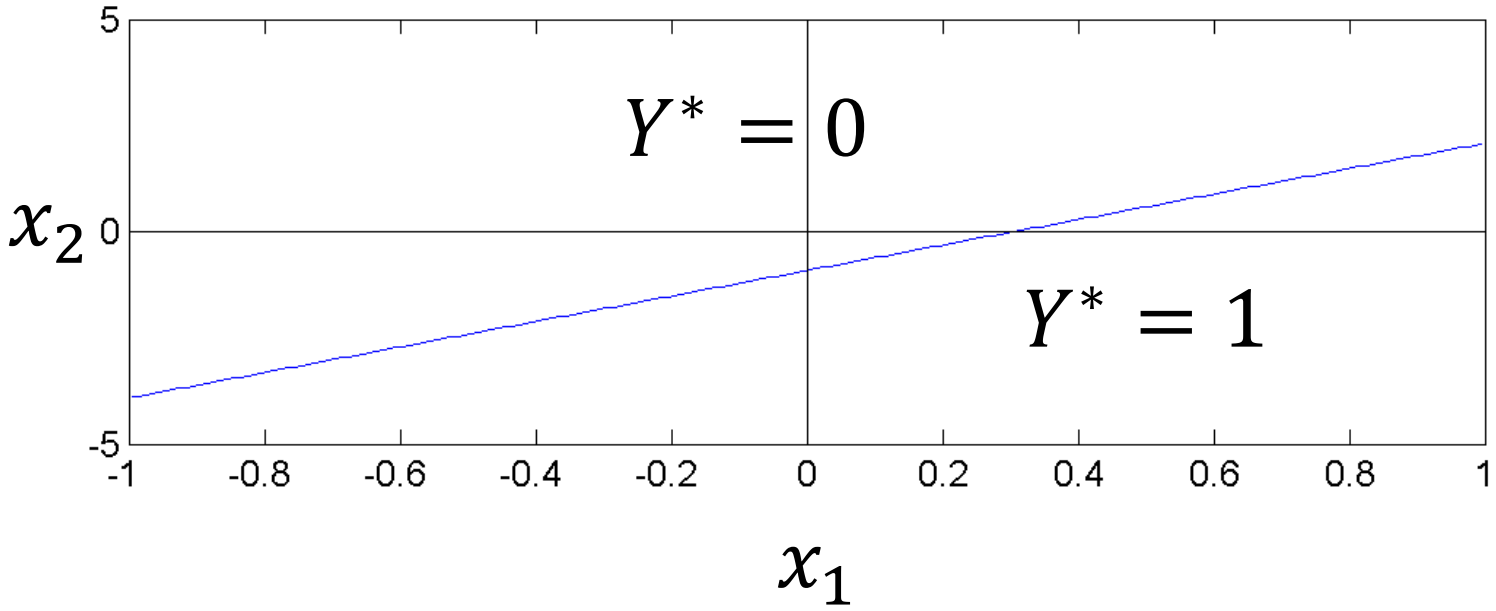
Linear Classifiers

Consider the classifier

$$Y^* = 1 \quad \text{if:} \quad b + \sum_{j=1}^{D} w_j x_j > 0$$

$$Y^* = 0 \quad \text{if:} \quad b + \sum_{j=1}^{D} w_j x_j < 0$$

This is called a "linear classifier" because the boundary between the two classes is a line. Here is an example of such a classifier, with its boundary plotted as a line in the two-dimensional space x_1 by x_2:

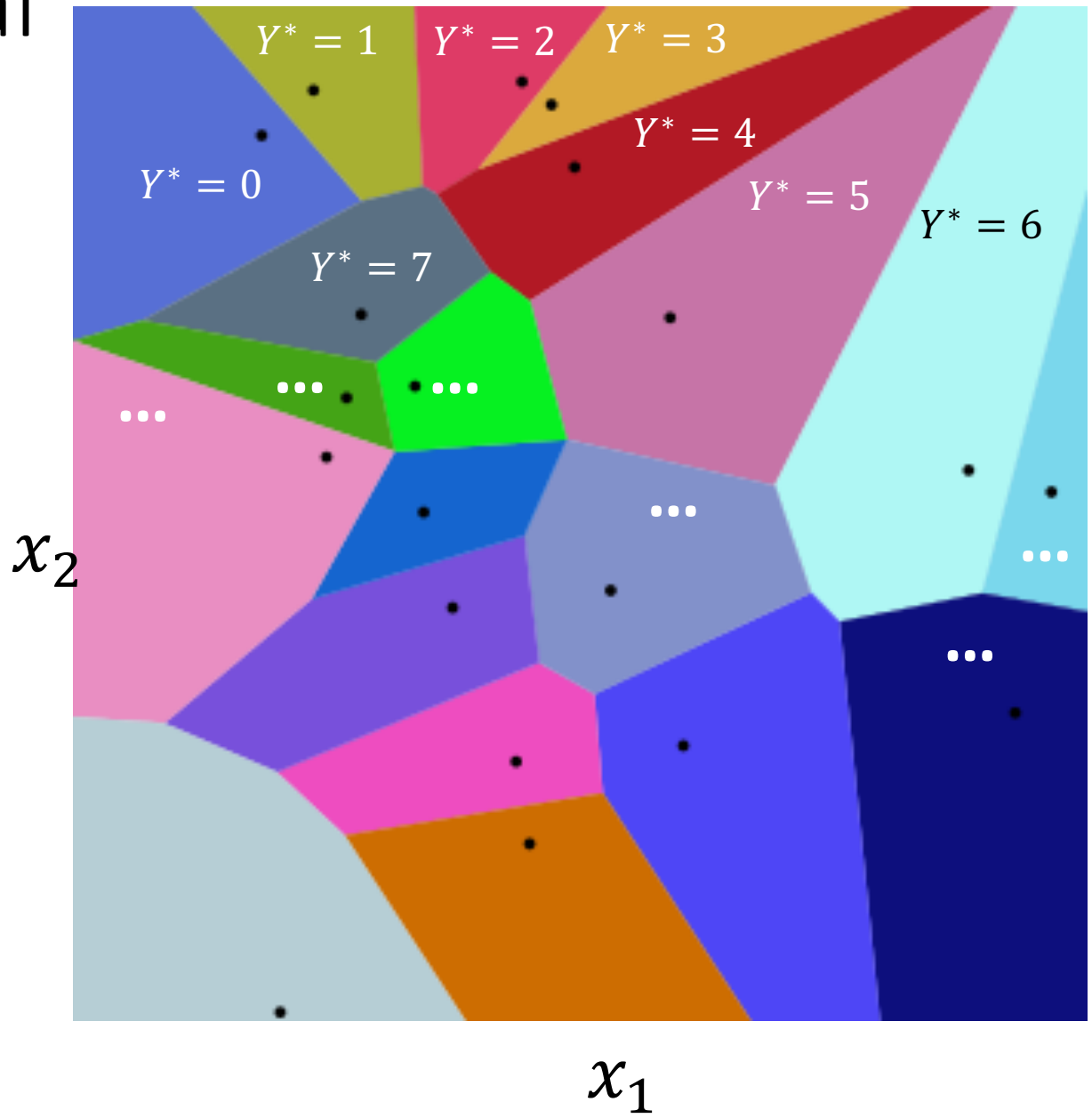


Linear Classifiers in General

Consider the classifier

$$Y^* = \arg\max_c \left(b_c + \sum_j w_{cj} x_j \right)$$

- This is called a "multi-class linear classifier."
- The regions $Y^* = 0,\ Y^* = 1,\ Y^* = 2$ etc. are called "Voronoi regions."
- They are regions with piece-wise linear boundaries. Here is an example from Wikipedia of Voronoi regions plotted in the two-dimensional space x_1 by x_2:



Classifiers

- Many types of classifiers
  - Naive Bayes, Bayesian Network
  - K-Nearest-Neighbors
  - Linear Classifiers: Perceptron, Logistic Regression
- How good is your classifier?
  - Train, Dev, and Test corpora
  - Confusion Matrix; Precision and Recall
- Review: Perceptron and Logistic Regression
  - Signed and unsigned classification functions

Accuracy

When we train a classifier, the metric that we usually report is "accuracy."

$$Accuracy = \frac{\#\ tokens\ correctly\ classified}{\#\ tokens\ total}$$

Classifiers

- Many types of classifiers
  - Naive Bayes, Bayesian Network
  - K-Nearest-Neighbors
  - Linear Classifiers: Perceptron, Logistic Regression
- How good is your classifier?
  - Train, Dev, and Test corpora
  - Confusion Matrix; Precision and Recall
- Review: Perceptron and Logistic Regression
  - Signed and unsigned classification functions

Accuracy on which corpus?

Consider the following experiment: among all of your friends' pets, there are 4 dogs and 4 cats.

1. Measure several attributes of each animal: weight, height, domesticity, color, number of letters in its name…
2. You discover that, among your friends' pets, all dogs have 1-syllable names, while the names of all cats have 2+ syllables.
3. Your classifier: an animal is a cat if its name has 2+ syllables.
4. Your accuracy: 100%

Is it correct to say that this classifier has 100%? Is it useful to say so?

Training vs. Test Corpora

Training Corpus = a set of data that you use in order to optimize the parameters of your classifier (for example, optimize which features you measure, what are the weights of those features, what are the thresholds, and so on).

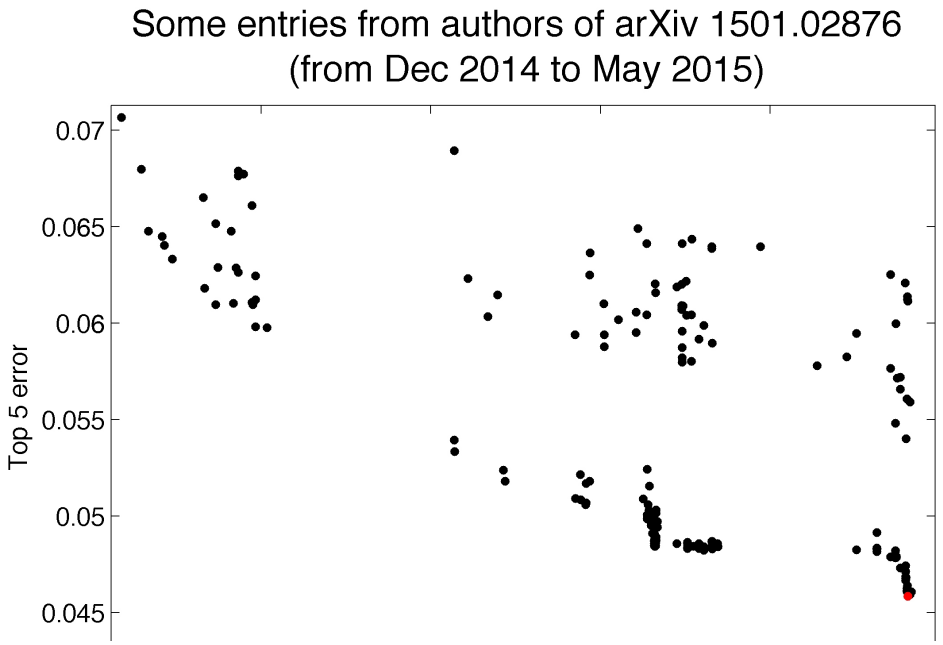
Test Corpus = a set of data that is non-overlapping with the training set (none of the test samples is also in the training dataset) that you can use to measure the accuracy.

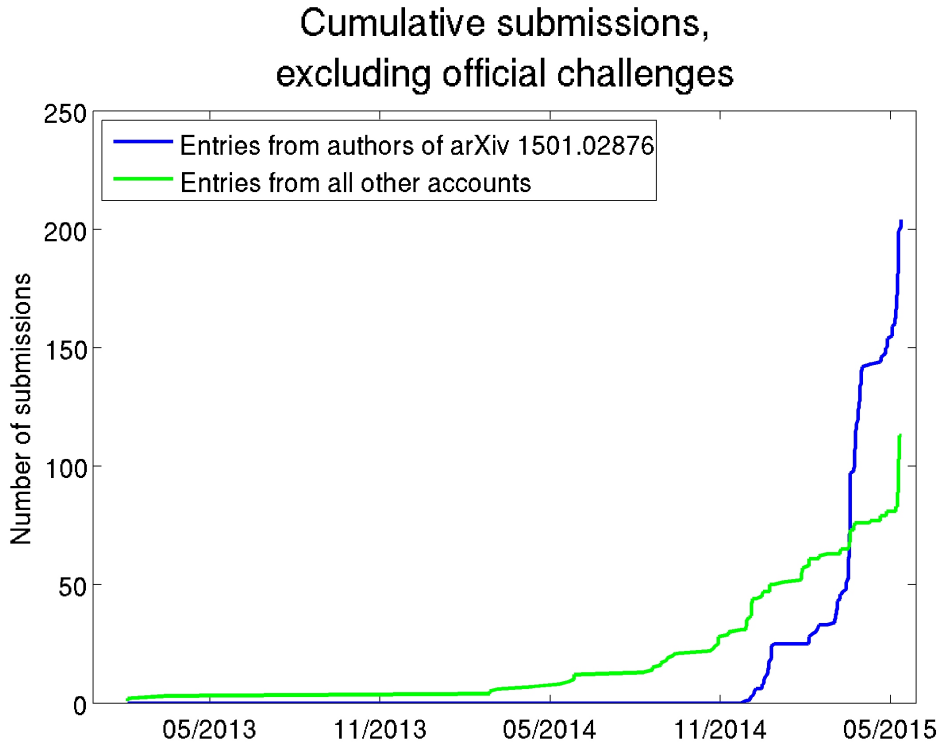
- Measuring the training corpus accuracy is useful for debugging: if your training algorithm is working, then training corpus accuracy should always go up.
- Measuring the test corpus accuracy is the only way to estimate how your classifier will work on new data (data that you've never yet seen).

Accuracy on which corpus?

This actually happened:

- Large Scale Visual Recognition Challenge 2015: Each competing institution was allowed to test up to 2 different fully-trained classifiers per week.
- One institution used 30 different e-mail addresses so that they could test a lot more classifiers (200, total). One of their systems achieved <46% error rate – the competition's best, at that time.
- That institution was forbidden from participating in the ImageNet competitions for the following 12 months.





Training vs. Development vs. Evaluation Corpora

Training Corpus = a set of data that you use in order to optimize the parameters of your classifier (for example, optimize which features you measure, what are the weights of those features, what are the thresholds, and so on).

Development Test (DevTest or Validation) Corpus = a dataset, separate from the training dataset, on which you test 200 different fully-trained classifiers (trained, e.g., using different training algorithms, or different features), in order to see which one works best.

Evaluation Test Corpus = a dataset that is used only to test the ONE classifier that does best on DevTest. From this corpus, you learn how well your classifier will actually perform in the real world.

Classifiers

- Many types of classifiers
  - Naive Bayes, Bayesian Network
  - K-Nearest-Neighbors
  - Linear Classifiers: Perceptron, Logistic Regression
- How good is your classifier?
  - Train, Dev, and Test corpora
  - Confusion Matrix; Precision and Recall
- Review: Perceptron and Logistic Regression
  - Signed and unsigned classification functions

Accuracy

When we train a classifier, the metric that we usually report is "accuracy."

$$Accuracy = \frac{\#\ tokens\ correctly\ classified}{\#\ tokens\ total}$$

The problem with accuracy

- Here are the first several words of the ISLEdict:
  - AAA (proper noun)
  - Aaberg (proper noun)
  - Aachen (proper noun)
  - Aaron (proper noun)
  - aaronic (adjective)
- How can we classify part of speech? How about this rule:

If it’s a word, then it is a proper noun.

The problem of unbalanced class distributions

- In most real-world problems, there is one class label that is much more frequent than all others.
  - Words: most words are nouns
  - Animals: most animals are insects
  - Disease: most people are healthy
- It is therefore easy to get a very high accuracy. All you need to do is write a program that completely ignores its input, and always guesses the majority class. The accuracy of this classifier is called the "chance accuracy."
- It is sometimes very hard to beat the chance accuracy. If chance=90%, and your classifier gets 89% accuracy, is that good, or bad?

The solution: Confusion Matrix

- In case of an unbalanced class distribution, it's not useful to summarize the performance of your classifier with just one number (Accuracy). You need more information.
- Confusion Matrix = a matrix whose $(m, n)^{\text{th}}$ element is the number of tokens of the m^{th} class that were labeled, by the classifier, as belonging to the n^{th} class.
- (The term might have been invented by George Miller & Patricia Nicely, 1955, "Analysis of Some Perceptual Confusions of English Consonants")

title: Consonant Confusions in CV utterances, for V=/a/, for S/N = +12db and
Phones involved: 16, namely p t k f T (th) s S (sh) b d g v D (dh) z Z (zh) m

	p	t	k	f	T	s	S	b	d	g	v	D	z	Z	m	n		Total
p	228	7	7	1	0	0	1	0	0	0	0	0	0	0	0	0	p	244
t	0	236	8	0	0	0	0	0	0	0	0	0	0	0	0	0	t	244
k	26	5	213	0	0	0	0	0	0	0	0	0	0	0	0	0	k	244
f	6	1	1	194	35	0	0	3	0	0	1	3	0	0	0	0	f	244
T	0	2	2	96	146	2	0	2	1	0	1	8	0	0	0	0	T	260
s	0	2	0	1	31	204	1	1	9	4	0	7	0	0	0	0	s	260
S	0	0	0	0	0	1	243	0	0	0	0	0	0	0	0	0	S	244
b	0	0	0	13	12	0	0	207	2	3	19	8	0	0	0	0	b	264
d	0	0	0	0	0	0	0	0	240	9	0	0	0	3	0	0	d	252
g	0	0	0	0	0	0	0	1	41	199	0	0	2	1	0	0	g	244
v	0	0	0	3	3	0	0	20	0	2	182	47	2	0	0	1	v	260
D	0	0	0	0	7	0	0	10	3	22	49	170	19	0	0	0	D	280
z	0	0	0	0	1	0	0	3	8	24	2	22	145	3	0	0	z	208
Z	0	0	0	0	0	0	1	0	2	0	0	0	13	264	0	0	Z	280
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	213	11	m	224
n	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	248	n	248

Plaintext versions of the Miller & Nicely matrices, posted by Dinoj Surendran, http://people.cs.uchicago.edu/~dinoj/research/nicely.html

False Positives & False Negatives

- Recall and precision are measured for each class separately.
- For each class, you create a 2x2 confusion matrix, with four entries:
  - TP (True Positives) = tokens that correctly belong to the target class, and were labeled as such by the classifier
  - FN (False Negatives) = tokens that correctly belong to the target class, but were not so labeled by the classifier
  - FP (False Positives) = tokens that don't belong to the target class, but the classifier thought they do belong
  - TN (True Negative) = tokens that don't belong to the target class, and were correctly rejected by the classifier

Classified As:

Correct Label:	Other	Target Class
Other	TN	FP
Target Class	FN	TP

Recall & Precision

Recall tells you, of the number that should have been detected, how many were correctly detected:

$$Recall = \frac{TP}{TP + FN}$$

Precision tells you, of the number actually detected, how many were correct:

$$Precision = \frac{TP}{TP + FP}$$

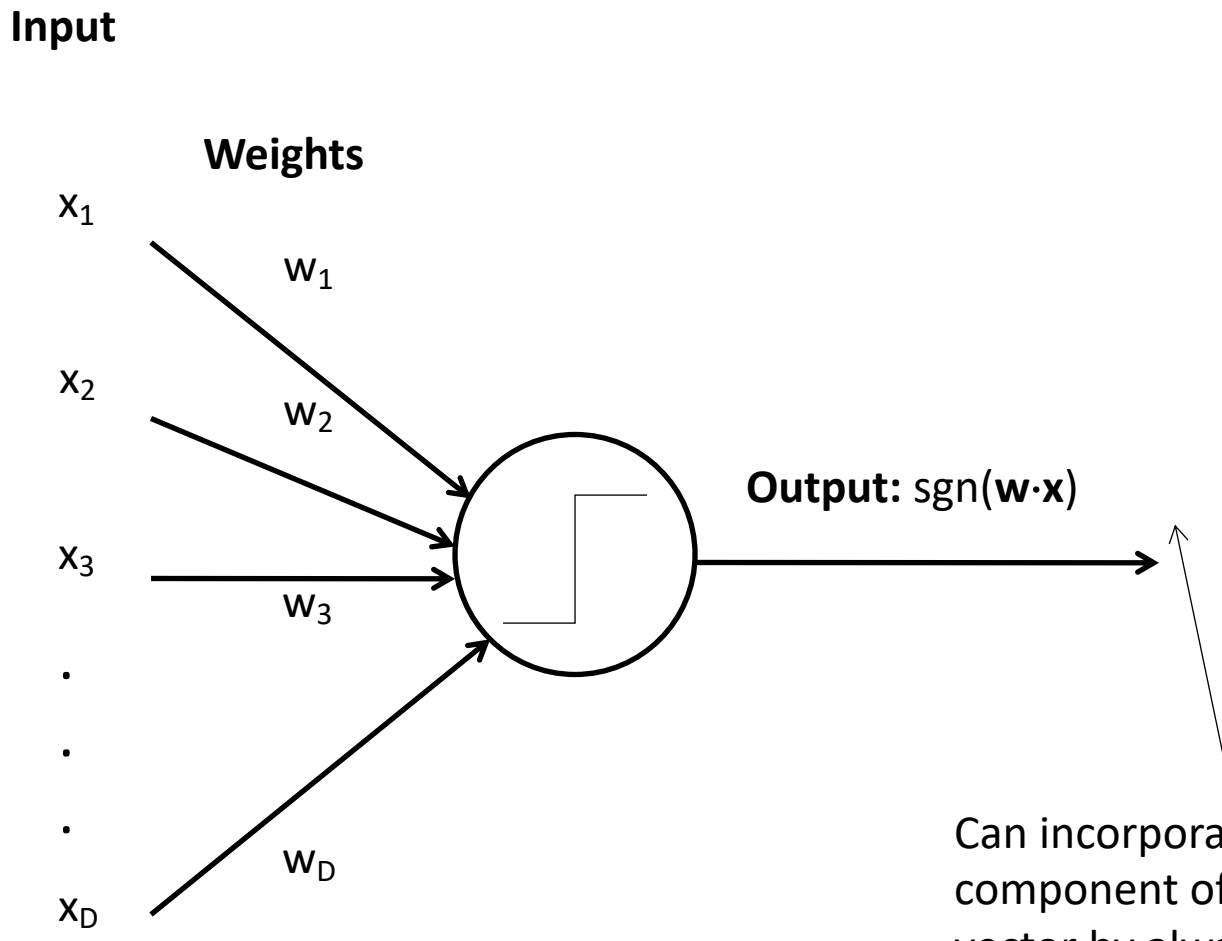
Classified As:

Correct Label:	Other	Target Class
Other	TN	FP
Target Class	FN	TP

Classifiers

- Many types of classifiers
  - Naive Bayes, Bayesian Network
  - K-Nearest-Neighbors
  - Linear Classifiers: Perceptron, Logistic Regression
- How good is your classifier?
  - Train, Dev, and Test corpora
  - Confusion Matrix; Precision and Recall
- Review: Perceptron and Logistic Regression
  - Signed and unsigned classification functions

Perceptron



Perceptron classification function:

$y^* = \text{sgn}(w_1x_1 + w_2x_2 + \ldots + w_Dx_D + b)$
$= \text{sgn}(\vec{w}^T\vec{x})$

Where $\vec{w} = [w_1, \ldots, w_D, b]^T$
and $\vec{x} = [x_1, \ldots, x_D, 1]^T$

Perceptron

For each training instance $\vec{x}$ with ground truth label $y \in \{-1,1\}$:

- Classify with current weights: $y^* = \text{sgn}(\vec{w}^T \vec{x})$
- Update weights:
  - if $y = y^*$ then do nothing
  - If $y \neq y^*$ then $\vec{w} = \vec{w} + \eta y \vec{x}$

- η (eta) is the "learning rate." If the training data are linearly separable, you can just set it to $\eta = 1$. If not (or if you don't know), you should set it to $\eta = 1/n$, where n is the number of training tokens you've seen.

Differentiable Perceptron

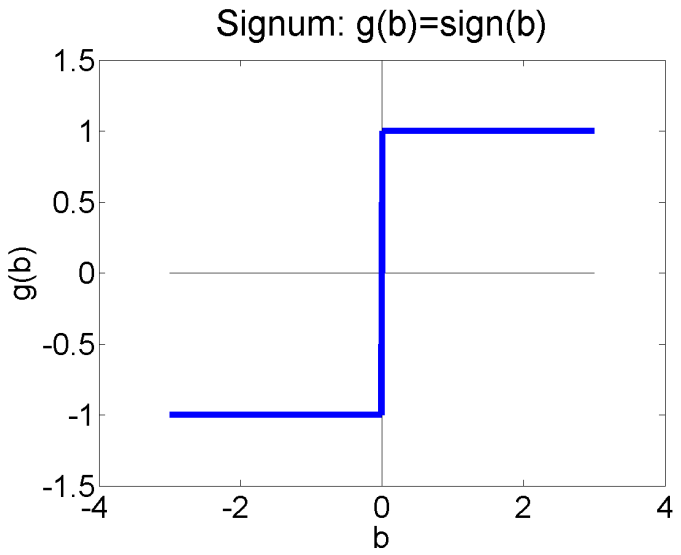
- Also known as a “one-layer feedforward neural network,” also known as “logistic regression.” Has been re-invented many times by many different people.
- Basic idea: replace the non-differentiable decision function

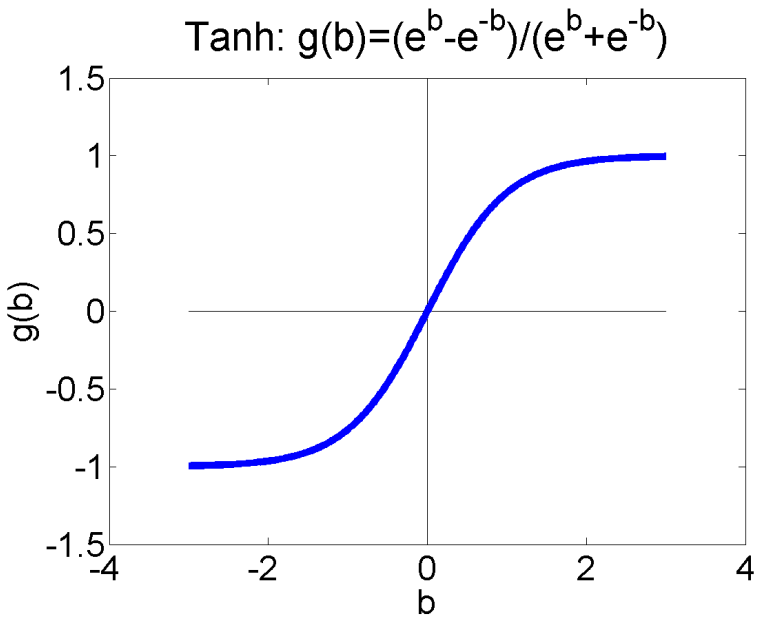
$$y^* = \operatorname{sgn}(\vec{w}^T \vec{x})$$

with a differentiable decision function:

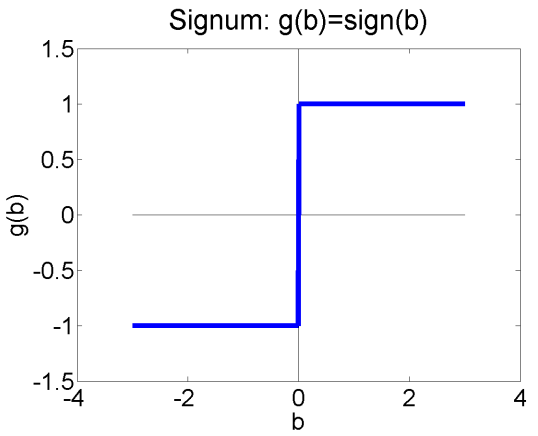
$$y^* = \tanh(\vec{w}^T \vec{x})$$

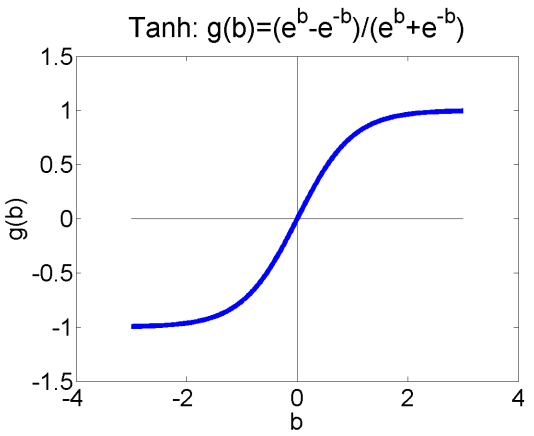
$$= \frac{1 - e^{-2\vec{w}^T \vec{x}}}{1 + e^{-2\vec{w}^T \vec{x}}}$$

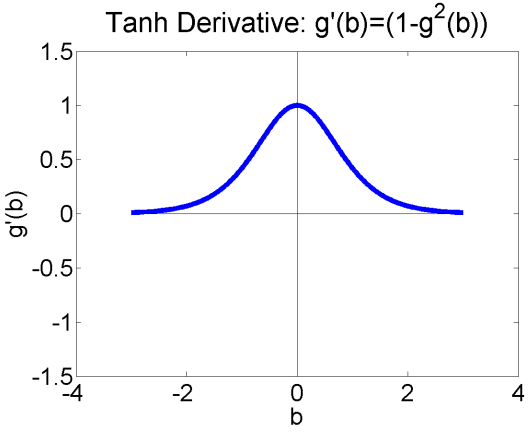




Differentiating tanh







Instead of $y_i^* = \text{sgn}(\beta)$

We use $y_i^* = \tanh(\beta)$.

(I'm using the abbreviation $\beta = \vec{w}^T \vec{x}$.)

That's pronounced "tanch," it means

"hyperbolic tangent," and it looks like this:

$$y_i^* = \tanh(\beta) = \frac{e^{\beta} - e^{-\beta}}{e^{\beta} + e^{-\beta}} = \frac{1 - e^{-2\beta}}{1 + e^{-2\beta}}$$

Its derivative is

$$\frac{dy_i^*}{d\beta} = \tanh'(\beta) = \frac{d}{d\beta}\left(\frac{1 - e^{-2\beta}}{1 + e^{-2\beta}}\right) = \frac{4e^{-2\beta}}{(1 + e^{-2\beta})^2} = 1 - \tanh^2(\beta) = 1 - y_i^{*2}$$

Differentiating the error

Remember that, if the true label is $y_i \in \{-1,1\}$ and the classifier outputs $y_i^* \in \{-1,1\}$, then we can write the training corpus error as:

$$L(\vec{w}) = \frac{1}{4}\sum_{i=1}^{n}(y_i - y_i^*)^2$$

Its derivative is:

$$\nabla_{\vec{w}} L = -\frac{1}{2}\sum_{i=1}^{n}(y_i - y_i^*)\,\nabla_{\vec{w}} y_i^* = -\frac{1}{2}\sum_{i=1}^{n}(y_i - y_i^*)\left(1 - y_i^{*2}\right)\nabla_{\vec{w}}(\vec{w}^T\vec{x}_i)$$

$$= -\sum_{i=1}^{n}\left(\frac{y_i - y_i^*}{2}\right)\left(1 - y_i^{*2}\right)\vec{x}_i$$

Comparing logistic regression vs. the perceptron

Logistic regression:

$$\vec{w} = \vec{w} - \eta \nabla_{\vec{w}} L = \vec{w} + \eta \sum_{i=1}^{n} \left(\frac{y_i - y_i^*}{2} \right) \left(1 - y_i^{*2}\right) \vec{x}_i$$

- If $y_i = y_i^*$ then do nothing.
- If $y_i \neq y_i^*$ then set $\vec{w} = \vec{w} + \eta \left(\frac{y_i - y_i^*}{2} \right) \left(1 - y_i^{*2}\right) \vec{x}_i$

Perceptron:

- If $y_i = y_i^*$ then do nothing.
- If $y_i \neq y_i^*$ then set $\vec{w} = \vec{w} + \eta y_i \vec{x}_i$

Classifiers

- Many types of classifiers
  - Naive Bayes, Bayesian Network
  - K-Nearest-Neighbors
  - Linear Classifiers: Perceptron, Logistic Regression
- How good is your classifier?
  - Train, Dev, and Test corpora
  - Confusion Matrix; Precision and Recall
- Review: Perceptron and Logistic Regression
  - Signed and unsigned classification functions

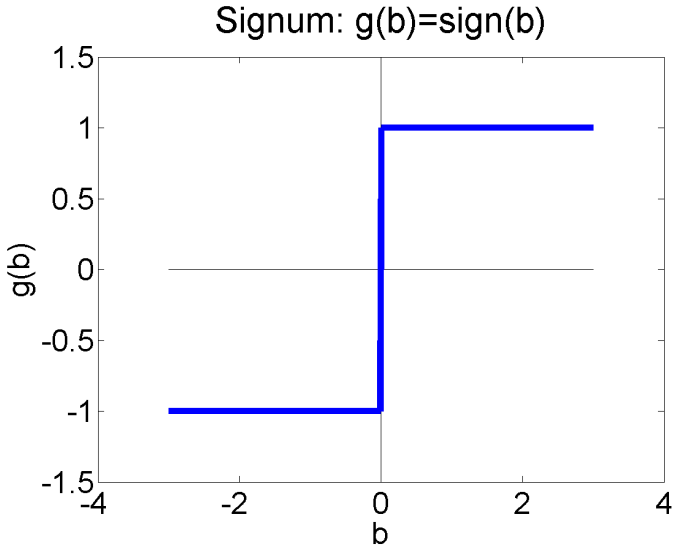
Signed vs. Unsigned Classification Functions

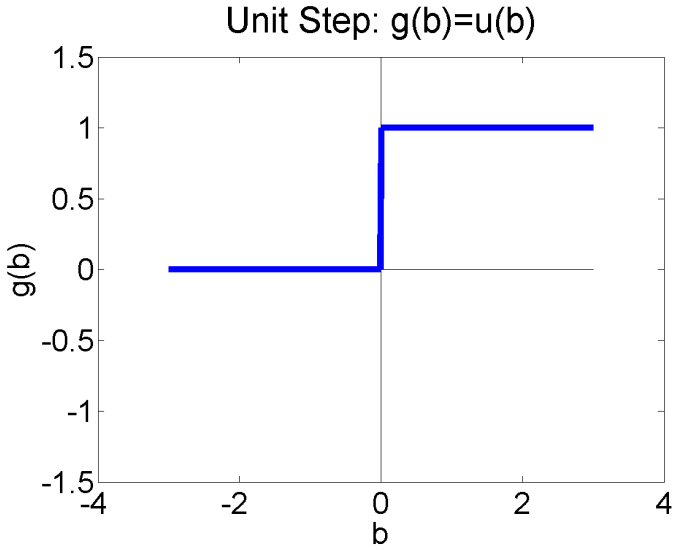
Signed Binary Classifier: Signum

Sometimes, we don't mind if the classifier outputs $y_i^* \in \{-1,1\}$. This can be written as $y_i^* = \mathrm{sgn}(\beta)$.

Unsigned Binary Classifier: Unit Step

Other times, we really want the classifier to output $y_i^* \in \{0,1\}$. This can be written as $y_i^* = \mathrm{u}(\beta)$, where $\mathrm{u}(\beta)$ is called the "unit step function."

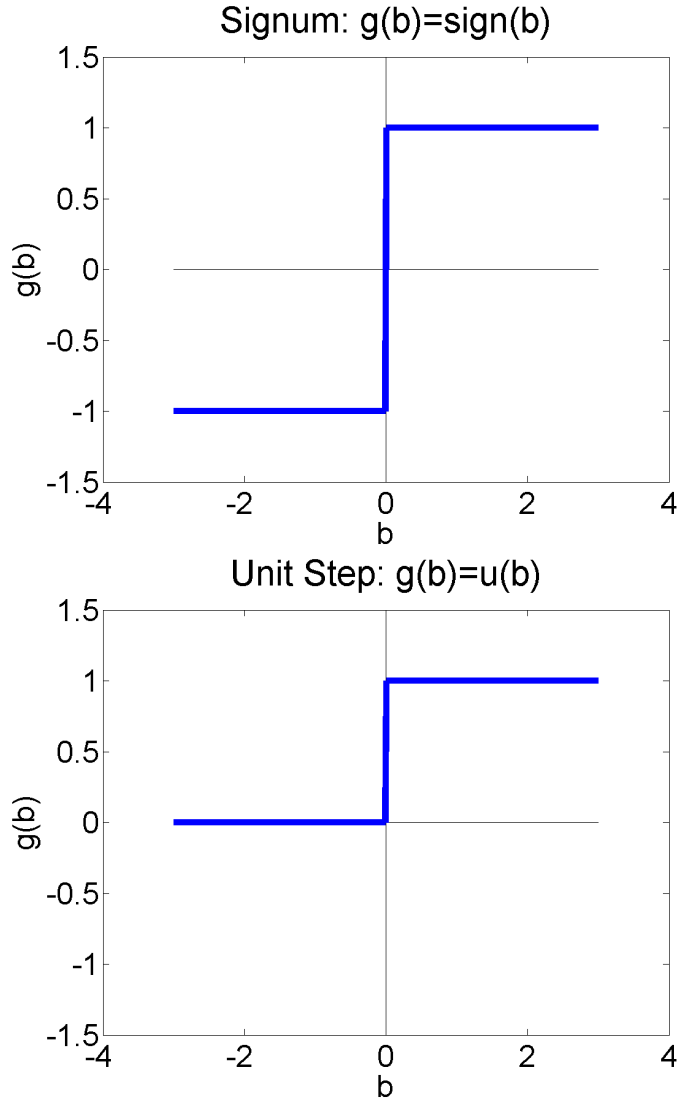




The relationship between sgn(β) and u(β)

Notice that

$$\mathrm{u}(\beta) = \frac{1}{2} + \frac{1}{2}\mathrm{sgn}(\beta)$$



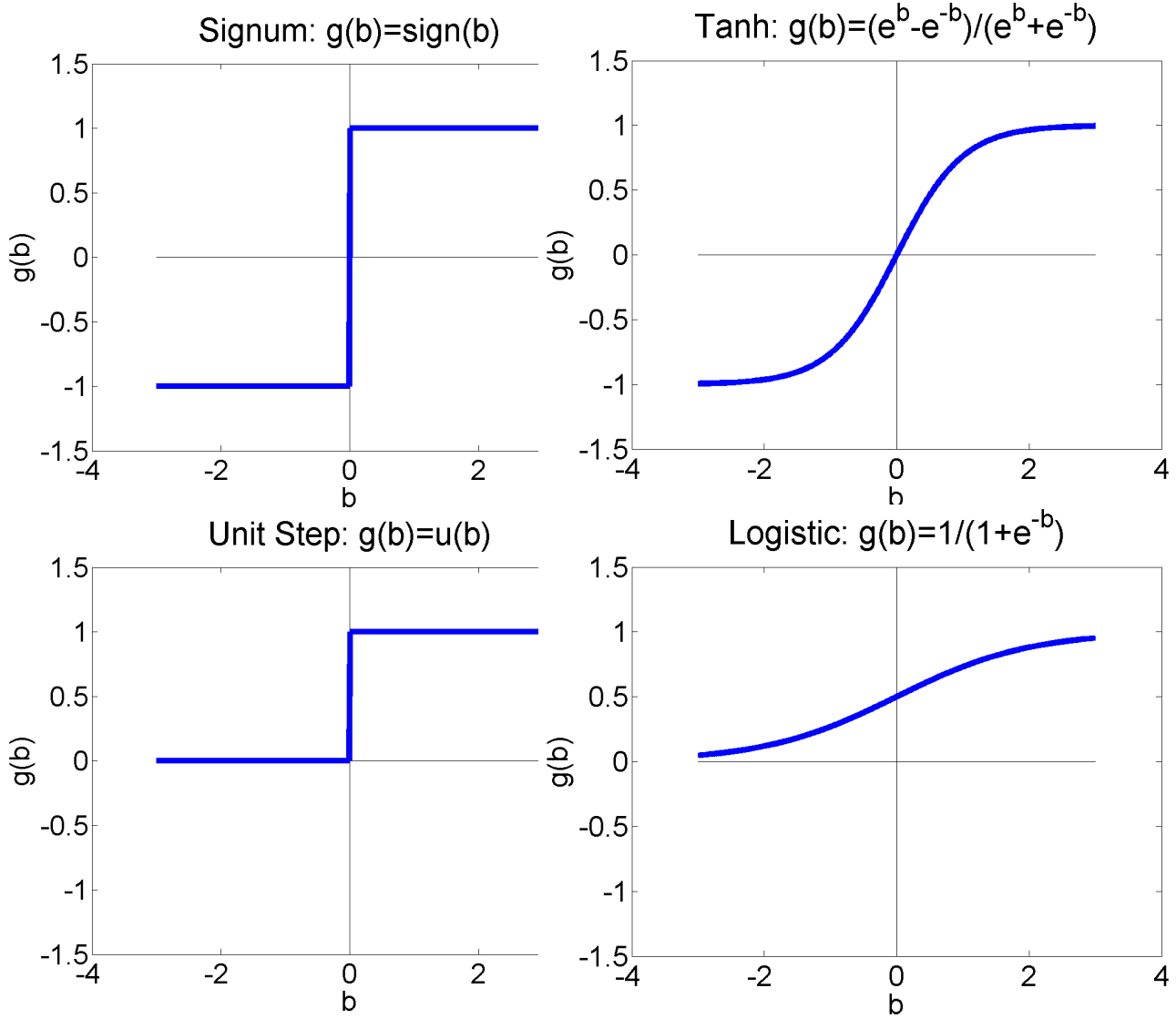
Signed vs. Unsigned Logistic Regression

Signed Logistic Regression: tanh

$$\tanh(\beta) = \frac{1 - e^{-2\beta}}{1 + e^{-2\beta}}$$

Unsigned Logistic Regression: The "logistic function" or "sigmoid function"

$$\sigma(\beta) = \frac{1}{1 + e^{-\beta}}$$



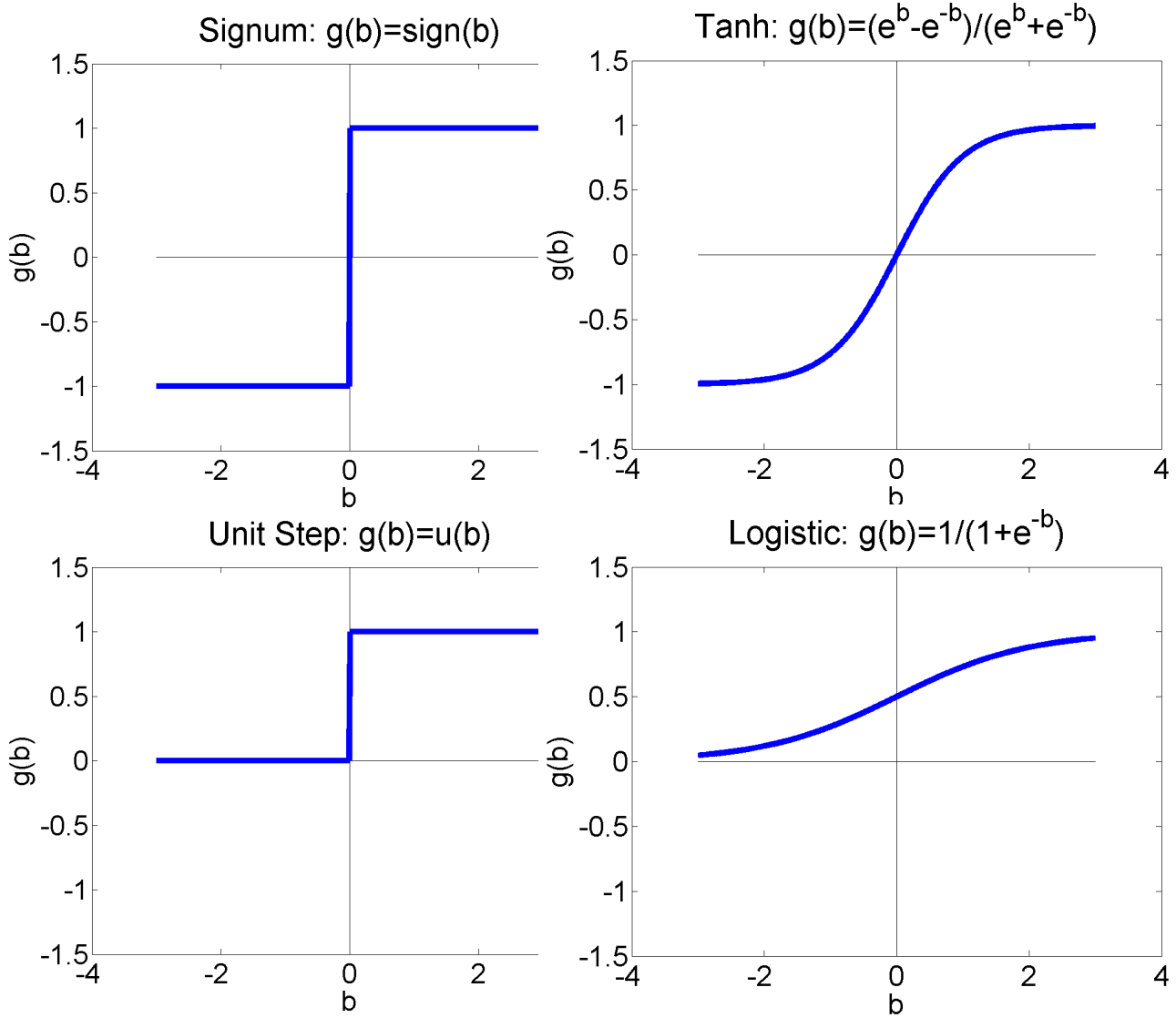
The relationship between $\tanh(\beta)$ and $\sigma(\beta)$

$$\tanh(\beta) = \frac{1 - e^{-2\beta}}{1 + e^{-2b}}$$

$$\sigma(\beta) = \frac{1}{1 + e^{-\beta}}$$

Notice that

$$\sigma(\beta) = \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{\beta}{2}\right)$$



A question for you to solve…

If

$$\sigma(\beta) = \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{\beta}{2}\right)$$

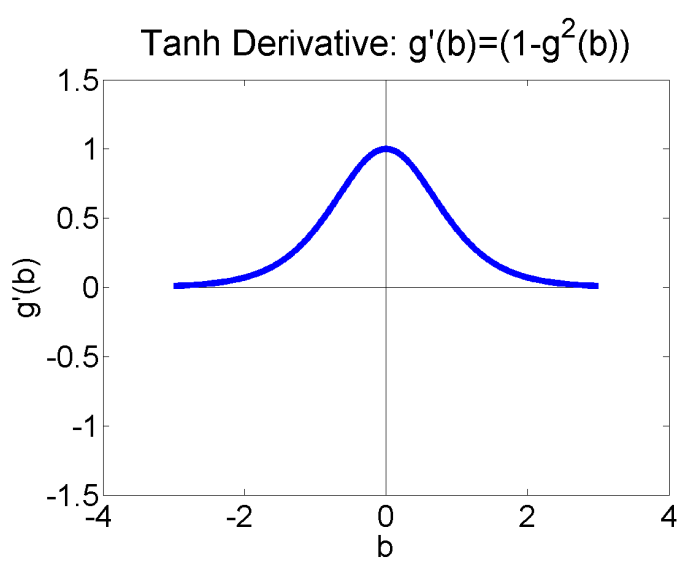
and

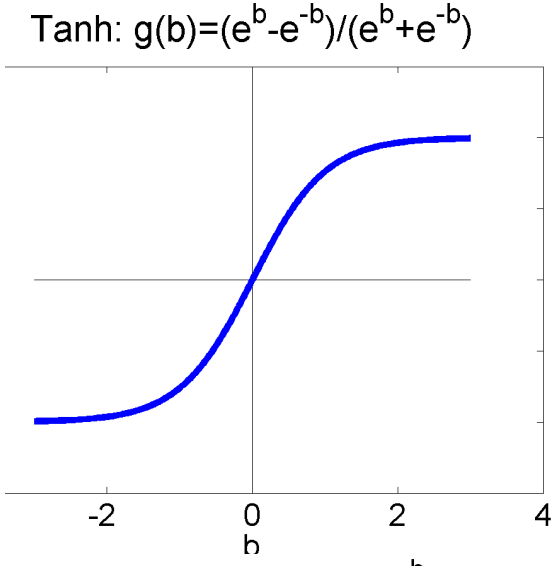
$$\tanh'(\beta) = 1 - \tanh^2(\beta)$$

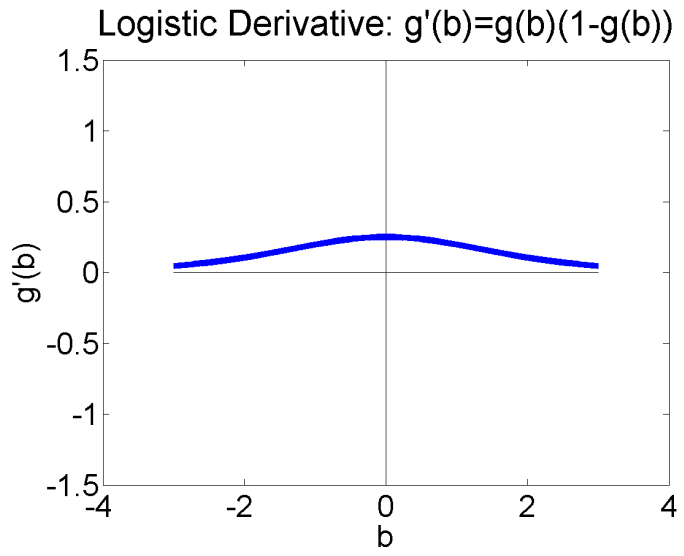
and

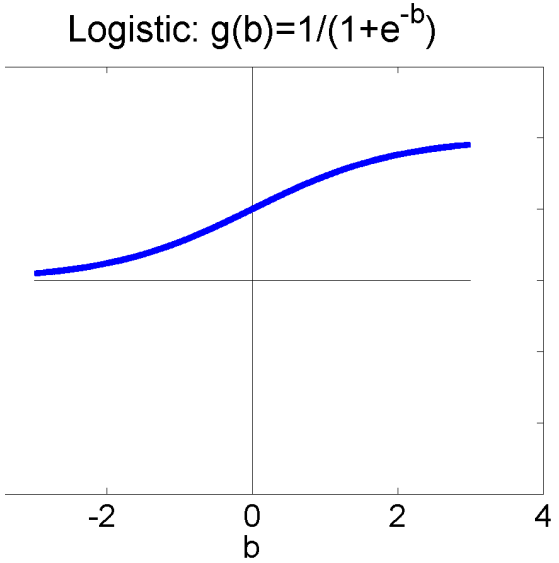
$$\tanh^2(\beta) = (2\,\sigma(2\beta) - 1)^2$$

…then what is $\sigma'(\beta)$?









Conclusions

- **Naïve Bayes** is a type of classifier. Other Bayesian Networks can be used as classifiers.
- **Nearest-Neighbors**: find the nearest training token and use its label. KNN: find the K nearest training tokens, and vote.
- **Training** Corpus trains the parameters; **DevTest** Corpus chooses the best of your classifiers; **Evaluation Test** Corpus tells you how well your classifier will do in the real world.
- **Confusion Matrix, Precision and Recall** tell you how your classifier will perform if you have an unbalanced class distribution.
- **Unsigned Logistic Regression**: $\sigma(b) = \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{b}{2}\right)$