\[
\text{sgn}(\mathbf{w}^T \mathbf{x}) = \begin{cases} 
1 & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\
-1 & \text{if } \mathbf{w}^T \mathbf{x} < 0 
\end{cases}
\]
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 = \( E_1, ..., E_D \)
• Classification function: output the label, \( Y \), which maximizes \( P(Y, E_1, ..., E_D) \) under the following “naïve Bayes” assumption:

\[
P(Y, E_1, ..., E_D) = P(Y) \prod_{d=1}^{D} P(E_d | Y)
\]
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, $y_i = \text{correct\_label}(x_i)$.

• 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, $y_i = \text{correct\_label}(x_i)$.
  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:
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.
K-Nearest Neighbors (KNN) Classifier

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, $y_i = \text{correct_label}(x_i)$.

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.
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.”

\[
\text{Accuracy} = \frac{\text{# tokens correctly classified}}{\text{# tokens total}}
\]
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.”

\[
\text{Accuracy} = \frac{\# \text{ tokens correctly classified}}{\# \text{ tokens total}}
\]
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 \((m, n)\)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”)

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

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

<table>
<thead>
<tr>
<th>Correct Label:</th>
<th>Classified As:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other</td>
<td>Other</td>
</tr>
<tr>
<td></td>
<td>TN</td>
</tr>
<tr>
<td>Target Class</td>
<td>FN</td>
</tr>
</tbody>
</table>
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:

\[
Precision = \frac{TP}{TP + FP}
\]

<table>
<thead>
<tr>
<th>Classified As:</th>
<th>Other</th>
<th>Target Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other</td>
<td>TN</td>
<td>FP</td>
</tr>
<tr>
<td>Target Class</td>
<td>FN</td>
<td>TP</td>
</tr>
</tbody>
</table>
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 classification function:

\[ y^* = \text{sgn}(w_1 x_1 + w_2 x_2 + \ldots + w_D x_D + b) = \text{sgn}(\vec{w}^T \vec{x}) \]

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

Can incorporate bias as component of the weight vector by always including a feature with value set to 1
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 $\vec{w} = \vec{w} + \eta y \vec{x}$

- $\eta$ (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

\[ y^* = \text{sgn}(\mathbf{w}^T \mathbf{x}) \]

with a differentiable decision function:

\[ y^* = \tanh(\mathbf{w}^T \mathbf{x}) \]

\[ = \frac{1 - e^{-2\mathbf{w}^T \mathbf{x}}}{1 + e^{-2\mathbf{w}^T \mathbf{x}}} \]
Differentiating tanh

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

We use \( y_i^* = \text{tanh}(\beta) \).

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

That’s pronounced “tanch,” it means “hyperbolic tangent,” and it looks like this:

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

Its derivative is

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

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

$$= - \sum_{i=1}^{n} \left( \frac{y_i - y_i^*}{2} \right) (1 - y_i^{*2}) \vec{x}_i$$
Comparing logistic regression vs. the perceptron

**Logistic regression:**

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

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

**Perceptron:**

- If \( y_i = y_i^* \) then do nothing.
- If \( y_i \neq y_i^* \) then set \( \overrightarrow{w} = \overrightarrow{w} + \eta y_i \hat{x}_i \)
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 \( y_i^* = \text{sgn}(\beta) \).

Unsigned Binary Classifier: Unit Step
Other times, we really want the classifier to output \( y_i^* \in \{ 0, 1 \} \). This can be written as \( y_i^* = u(\beta) \), where \( u(\beta) \) is called the “unit step function.”
The relationship between $\text{sgn}(\beta)$ and $u(\beta)$

Notice that

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

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

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

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-Neighors**: 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**: \( \sigma(b) = \frac{1}{2} + \frac{1}{2} \tanh \left( \frac{b}{2} \right) \)