CS 886
Applied Machine Learning
Linear Methods for Classification

Dan Lizotte

University of Waterloo

18 Sept 2012
Linear models in general [HTF Ch. 2.8.3]

- By linear models, we mean that the hypothesis function \( h_w(x) \) is a *linear function of the parameters* \( w \).
- In other words, predictions are a *linear combination of feature values*.
- In general (this more ML-like notation)

\[
h_w(x) = \sum_{k=0}^{p} w_k \phi_k(x) = w^T \phi(x)
\]

where \( \phi_k \) are called *basis functions* (or features!)

- As usual, we will assume that \( \phi_0(x) = 1, \forall x \), to create a bias term.
- To recover degree-\( d \) polynomial regression in one variable, set

\[
\phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2, \ldots, \phi_d(x) = x^d
\]

- The hypothesis can alternatively be written as:

\[
h_w(x) = \Phi w
\]

where \( \Phi \) is a matrix with one row per instance; row \( j \) contains \( \phi(x_j) \).

- Basis functions are *fixed* for a given analysis.
Example basis functions: Polynomials

\[ \phi_k(x) = x^k \]

“Global” functions: a small change in \( x \) may cause large change in the output of many basis functions. Leads to numerical instability, but there are much better-behaved polynomial bases, e.g. Legendre.
Example basis functions: Gaussians

\[ \phi_k(x) = \exp \left( \frac{-(x - \mu_k)^2}{2\sigma^2} \right) \]

- \( \mu_k \) controls the position along the x-axis, \( \sigma \) controls the width
- \( \mu_k, \sigma \) fixed for now (later we discuss adjusting them)
- "Local" functions: if \( \sigma \) is relatively small, a small change in \( x \) only causes a change in the output of a few basis functions (the ones with means close to \( x \))
**Example basis functions: Sigmoidal**

\[
\phi_k(x) = \sigma \left( \frac{x - \mu_k}{s} \right)
\]

where \(\sigma(a) = \frac{1}{1 + \exp(-a)}\)

- \(\mu_k\) controls the position along the x-axis, \(s\) controls the slope
- "Local" functions: a small change in \(x\) only causes a change in the output of a few basis (most others will stay close to 0 or 1)
Linear Methods for Classification

- Classification tasks
- Error functions for classification
- Logistic Regression
- Support Vector Machines
Recall: Classification problems

- Given a data set \( \langle x_i, y_i \rangle \), where \( y_i \) are discrete, find a hypothesis that can predict new \( y \) given new \( x \)
- If \( y_i \in \{0, 1\} \), (or \( \{-1, 1\} \)) this is *binary classification* (special case)
- If \( y_i \) can take more than two values, the problem is called *multi-class classification*
- Multi-class versions of most binary classification algorithms can be developed in a fairly straightforward way
Classifier learning algorithms

- What is a good error function for classification?
- What hypothesis classes can we use?
- What algorithms are useful for searching those hypotheses classes?
Example: Given “nucleus size" predict cancer recurrence
Example: Solution by linear regression

- Univariate real input: nucleus size
- Output coding: non-recurrence = 0, recurrence = 1
- **Sum squared error** minimized by the red line
Linear regression for classification

- The predictor shows an increasing trend towards recurrence with larger nucleus size, as expected.
- Output *cannot be directly interpreted* as a class prediction.
- Thresholding output (e.g., at 0.5) could be used to predict 0 or 1. (In this case, prediction would be 0 except for extremely large nucleus size.)
- Interpret as probability? Not bounded to \([0, 1]\), not consistent even for well-separated data
Probabilistic view

• Suppose we have two possible classes: $y \in \{0, 1\}$.
• The symbols “0” and “1” are unimportant. Could have been \{a, b\}, \{up, down\}, whatever. We’ll use $y \in \{0, 1\}$ though.
• Rather than try to predict the class label directly, ask: What is the probability that a given input $x$ to has class $y = 1$?
• Bayes Rule:

$$P(y = 1|x) = \frac{P(x, y = 1)}{P(x)} = \frac{P(x|y = 1)P(y = 1)}{P(x|y = 1)P(y = 1) + P(x|y = 0)P(y = 0)}$$
Probabilistic models for binary classification

- Can also write:

\[
P(y = 1|\mathbf{x}) = \sigma \left( \log \frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})} \right) = \sigma \left( \log \frac{P(\mathbf{x}|y = 1)P(y = 1)}{P(\mathbf{x}|y = 0)P(y = 0)} \right)
\]

where \( \sigma(a) = \frac{1}{1+\exp(-a)} \), the sigmoid or logistic function.

- **Discriminative Learning**
  - Model (i.e. learn) \( \log \frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})} \) (log odds ratio) as a linear function of \( \mathbf{x} \)
  - Only models how to discriminate (i.e. separate) examples of the two classes. Does not model distribution of \( \mathbf{x} \).

- **Generative Learning**
  - Model (i.e. learn) \( P(y = 1), P(y = 0), P(\mathbf{x}|y = 1), P(\mathbf{x}|y = 0) \), then use rightmost formula above
  - can actually use the model to generate (i.e. fantasize) data
Logistic regression [HTF Ch. 4.4]

- Represent the hypothesis as a logistic function of a linear combination of inputs:
  \[
  h(x) = \sigma(w^T x)
  \]
- Interpret \( h(x) \) as \( P(y = 1|x) \), interpret \( w^T x \) as the log-odds ratio.
- How do we choose \( w \)?
- In the probabilistic framework, observing \( \langle x_i, 1 \rangle \) (\( \langle x_i, 0 \rangle \)) does not mean \( h(x_i) \) should be 1 (0).
- **Maximize the probability of having observed the \( y_i \), given the \( x_i \).**
Max Conditional Likelihood/Min Cross-Entropy

- **Maximize the probability of having observed the** $y_i$, **given the** $x_i$.
- **Assumption 1:** Examples are drawn i.i.d. distribution. Probability of observing all $y$s is product

$$
P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n | X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \prod_{i=1}^{n} P(Y_i = y_i | X_i = x_i)
$$

- **Assumption 2:**

$$
P(y = 1|x) = h_w(x) = \sigma(w^T x) = 1/(1 + \exp(-w^T x))
$$
$$
P(y = 0|x) = (1 - \sigma(w^T x)) = \exp(-w^T x)/(1 + \exp(-w^T x))
$$

- More stable to maximize log probability. Therefore

$$
\log \prod_{i=1}^{n} P(Y_i = y_i | X_i = x_i) = \sum_{i=1}^{n} [y_i \log(h_w(x_i)) + (1 - y_i) \log(1 - h_w(x_i))]
$$
Max Conditional Likelihood/Min Cross-Entropy

- **Maximize the probability of having observed the** $y_i$, **given the** $x_i$.
- More stable to maximize log probability. Therefore

$$\log \prod_{i=1}^{n} P(Y_i = y_i | X_i = x_i) = \begin{cases} 
\log h_w(x_i) & \text{if } y_i = 1 \\
\log(1 - h_w(x_i)) & \text{if } y_i = 0 
\end{cases}$$

$$= \sum_{i=1}^{n} [y_i \log(h_w(x_i)) + (1 - y_i) \log(1 - h_w(x_i)))]$$

- Suggests an error

$$J(h_w) = - \sum_{i=1}^{n} [y_i \log(h_w(x_i)) + (1 - y_i) \log(1 - h_w(x_i))]$$

- This is the *cross entropy*. Number of bits to transmit the labels if both parties know $h_w$ and the $x_i$. 

Cross-Entropy Error Surface

\[ J(h) = - \sum_{i=1}^{n} \left[ y_i \log \sigma(w^T x_i) + (1 - y_i) \log(1 - \sigma(w^T x_i)) \right] \]

Nice convex error surface\(^1\), but \textit{cannot solve in closed form}

\(^1\)Not always well-behaved though...
Back to the breast cancer problem

\[ h_w(x) = \frac{1}{1 + \exp(-w^T x)} \], \ w = (-3.4671, 0.1296)^T
## Supervised Learning Methods: “Objective-driven”

<table>
<thead>
<tr>
<th>Mthd.</th>
<th>Form</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLS</td>
<td>( h_w(x) = x^T w )</td>
<td>( \text{MSE: } \sum_{i=1}^{n} (h_w(x_i) - y_i)^2 )</td>
</tr>
<tr>
<td></td>
<td>Models ( E[y</td>
<td>x] )...</td>
</tr>
<tr>
<td>LR</td>
<td>( h_w(x) = \frac{1}{1+e^{-x^T w}} )</td>
<td>( -\sum_{i=1}^{n} y_i \log h(x_i) + (1 - y_i) \log(1 - h(x_i)) )</td>
</tr>
<tr>
<td></td>
<td>Models ( P(y</td>
<td>x) )...</td>
</tr>
</tbody>
</table>
Generative learning

- Learn a model of the joint distribution $P(x, y)$
- This distribution can be used both to \textit{generate} data \textbf{and} to classify instances
- Note $P(y|x) = P(x, y)/P(x) = P(x, y)/\sum_y P(x, y)$
- Suppose $x_i$ are binary, $y$ is binary. How can we represent $P(x, y)$?
- Just write down the probability of each possible $\langle x, y \rangle$
- How many are there? $2^{p+1}$
- Even writing all these down is a challenge; good estimation would require much more than $2^{p+1}$ examples in our training set.
Naïve Bayes assumption [HTF Ch. 6.6.3]

- Assume the $x_i$ are conditionally independent given $y$.
- In other words, assume that:
  \[
P(x_j = 1|y, x_k, \ldots) = P(x_j = 1|y), \forall j, k \neq j
  \]
- Then, for any input vector $\mathbf{x}$, we have:
  \[
P(\mathbf{x}|y) = P(x_1, x_2, \ldots, x_p|y)
  = P(x_1|y)P(x_2|y, x_1) \cdots P(x_p|y, x_1, \ldots x_{p-1})
  = P(x_1|y)P(x_2|y) \cdots P(x_p|y) \text{ by NB assumption}
  \]
  \[
P(\mathbf{x}, y) = P(x_1|y)P(x_2|y) \cdots P(x_p|y)P(y) \text{ by NB assumption}
  \]
- For binary features, instead of $O(2^p)$ numbers to describe a model, we only need $2p + 1$
Aside: Bayesian networks

• If not all conditional independence relations are true, we can introduce dependencies. This increases the number of parameters beyond $2p + 1$

• This more general type of model, annotated with conditional distributions, is called a *Bayesian Network*. One kind of *graphical model*.

• Beyond the scope of this course; new book by Kevin Murphy: http://www.cs.ubc.ca/~murphyk/MLbook/index.html
Learning the parameters of a Naïve Bayes classifier

- Parameters of the model are $\theta_1 = P(y = 1)$, $\theta_{j,1} = P(x_j = 1|y = 1)$, $\theta_{j,0} = P(x_j = 1|y = 0)$, for $j \in \{1, \ldots, p\}$.
- Find the parameters that maximize the joint log likelihood of the training data.
- The likelihood in this case is:

$$
\mathcal{L}(\theta_1, \theta_{j,1}, \theta_{j,0}, j \in \{1..p\}) = \prod_{i=1}^{n} P(y_i, x_i) = \prod_{i=1}^{n} \left[ P(y_i) \prod_{j=1}^{p} P(x_{i,j}|y_i) \right]
$$

- First, use the log trick:

$$
\log \mathcal{L}(\theta_1, \theta_{j,1}, \theta_{j,0}) = \sum_{i=1}^{n} \left( \log P(y_i) + \sum_{j=1}^{p} \log P(x_{i,j}|y_i) \right)
$$

- Observe that each term in the sum depends on the values of $y_i, x_i$ that appear in the $i$th instance.
Max Likelihood Parameter Estimation for Naïve Bayes

\[
\log L(\theta_1, \theta_{j,1}, \theta_{j,0}) = \sum_{i=1}^{n} \left[ y_i \log \theta_1 + (1 - y_i) \log(1 - \theta_1) \right]
+ \sum_{j=1}^{p} y_i \left( x_{i,j} \log \theta_{i,1} + (1 - x_{i,j}) \log(1 - \theta_{i,1}) \right)
+ \sum_{j=1}^{p} (1 - y_i) \left( x_{i,j} \log \theta_{i,0} + (1 - x_{i,j}) \log(1 - \theta_{i,0}) \right)
\]

To estimate \(\theta_1\), we take the derivative of \(\log L\) wrt \(\theta_1\) and set it to 0:

\[
\frac{\partial L}{\partial \theta_1} = \sum_{i=1}^{n} \left( \frac{y_i}{\theta_1} + \frac{1 - y_i}{1 - \theta_1} (-1) \right) = 0
\]
Maximum likelihood parameters estimation for Naïve Bayes

By solving for $\theta_1$, we get:

$$\theta_1 = \frac{1}{n} \sum_{j=1}^{n} y_i = \frac{\text{number of examples of class 1}}{\text{total number of examples}}$$

Using a similar derivation, we get:

$$\theta_{j,1} = \frac{\text{number of instances for which } x_{i,j} = 1 \text{ and } y_i = 1}{\text{number of instances for which } y_i = 1}$$

$$\theta_{j,0} = \frac{\text{number of instances for which } x_{i,j} = 1 \text{ and } y_i = 0}{\text{number of instances for which } y_i = 0}$$
Logistic Regression vs. Naïve Bayes

- Model $P(y|x)$
- Assume $x_j \in \mathbb{R}$, and assume
  \[ P(y|x) = \sigma(x^Tw) \]
- Train: Maximize $\sum_{i=1}^{n} P(y_i|x_i)$
- Training requires numerical optimization

- Model $P(y, x)$
- Assume, $x_j \in \mathbb{Z}$ and assume
  \[ P(x_j|y, x_k, \ldots) = P(x_j|y) \]
- Train: Maximize $\sum_{i=1}^{n} P(y_i, x_i)$
- Training requires counting events

*For NB, $x_j \in \mathbb{R}$ possible with assumptions about form of $P(x_j|y)$ (e.g. Normal)
Decision boundary [HTF Ch. 2.2.3]

- How complicated is a classifier?
- One way to think about it is in terms of its decision boundary, i.e., the line it defines for separating examples
- **Linear classifiers** draw a hyperplane between examples of the different classes. **Non-linear classifiers** draw more complicated surfaces between the different classes.
- For a probabilistic classifier with a cutoff\(^2\) of 0.5, the decision boundary is the curve on which:

\[
\frac{P(y = 1|x)}{P(y = 0|x)} = 1, \text{ i.e., where } \log \frac{P(y = 1|x)}{P(y = 0|x)} = 0
\]

- For logistic regression, the boundary is \(w^T x = 0\), i.e. linear

\(^2\)Is this always the right cutoff?
Naïve Bayes for binary features

- The parameters of the model are \( \theta_{i,1} = P(x_i = 1|y = 1) \), \( \theta_{i,0} = P(x_i = 1|y = 0) \), and \( \theta_1 = P(y = 1) \).

- What is the decision boundary?

\[
\frac{P(y = 1|x)}{P(y = 0|x)} = \frac{P(y = 1) \prod_{i=1}^{n} P(x_i|y = 1)}{P(y = 0) \prod_{i=1}^{n} P(x_i|y = 0)}
\]

- Using the log trick, we get:

\[
\log \frac{P(y = 1|x)}{P(y = 0|x)} = \log \frac{P(y = 1)}{P(y = 0)} + \sum_{i=1}^{n} \log \frac{P(x_i|y = 1)}{P(x_i|y = 0)}
\]

- Note that in the equation above, the \( x_i \) would be 1 or 0.
Decision boundary of Naïve Bayes with binary features

Let:

\[ w_0 = \log \frac{P(y = 1)}{P(y = 0)} \]

\[ w_{i,1} = \log \frac{P(x_i = 1|y = 1)}{P(x_i = 1|y = 0)} \]

\[ w_{i,0} = \log \frac{P(x_i = 0|y = 1)}{P(x_i = 0|y = 0)} \]

We can re-write the decision boundary as:

\[
\log \frac{P(y = 1|x)}{P(y = 0|x)} = w_0 + \sum_{i=1}^{n} (w_{i,1}x_i + w_{i,0}(1 - x_i))
\]

\[ = w_0 + \sum_{i=1}^{n} w_{i,0} + \sum_{i=1}^{n} (w_{i,1} - w_{i,0})x_i \]

This is a \textit{linear decision boundary}!
## Supervised Learning Methods: “Objective-driven”

<table>
<thead>
<tr>
<th>Mthd.</th>
<th>Form</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLS</td>
<td>$h_w(x) = x^T w$</td>
<td>(\text{MSE: } \sum_{i=1}^{n} (h_w(x_i) - y_i)^2)</td>
</tr>
<tr>
<td></td>
<td>Models (E[y</td>
<td>x])...</td>
</tr>
<tr>
<td>LR</td>
<td>(h_w(x) = \frac{1}{1+e^{-x^T w}})</td>
<td>(-\sum_{i=1}^{n} y_i \log h(x_i) + (1 - y_i) \log (1 - h(x_i)))</td>
</tr>
<tr>
<td></td>
<td>Models (P(y</td>
<td>x))...</td>
</tr>
<tr>
<td>NB</td>
<td>[P(Y=1)P(x</td>
<td>Y=1) \sum_{i=0,1} P(x</td>
</tr>
<tr>
<td></td>
<td>Models (P(y, x))...</td>
<td>...by modeling (P(y)) and (P(x</td>
</tr>
</tbody>
</table>

- Both LR and NB define a linear decision boundary. How are they different? **Training objective.**
## Supervised Learning Methods: “Objective-driven”

<table>
<thead>
<tr>
<th>Mthd.</th>
<th>Form</th>
<th>Objective</th>
</tr>
</thead>
</table>
| **OLS** | \( h_w(x) = x^T w \) | MSE: \( \sum_{i=1}^{n} (h_w(x_i) - y_i)^2 \)  
Models \( E[y|x] \)...using a linear function |
| **LR** | \( h_w(x) = \frac{1}{1+e^{-x^T w}} \) | \(-\sum_{i=1}^{n} y_i \log h(x_i) + (1 - y_i) \log(1 - h(x_i)) \)  
Models \( P(y|x) \)...using sigmoid of a linear function |
| **NB** | \( \frac{P(Y=1)P(x|Y=1)}{\sum_{i=0,1} P(x|Y=i)P(Y=i)} \) | \(-\sum_{i=1}^{n} \left( \log P(y_i) + \sum_{j=1}^{p} \log P(x_{i,j}|y_i) \right) \)  
Models \( P(y, x) \)...by modeling \( P(y) \) and \( P(x|y) \) |
| **SVM** | \( h_w(x) = \text{sgn}(x^T w) \) | “Distance of closest pt. to boundary (maximize)”  
Models boundary |
Large Margin Classifiers: Linear Support Vector Machines

• Linear classifiers that focus on learning the decision boundary rather than the conditional distribution $P(y|x)$
  • Perceptrons
    • Definition
    • Perceptron learning rule
    • Convergence
  • Margin & max margin classifiers
  • (Linear) support vector machines
    • Formulation as optimization problem
Perceptrons [HTF Ch. 4.5]

- Consider a binary classification problem with data \( \{x_i, y_i\}_{i=1}^n \), \( y_i \in \{-1, +1\} \). **Note coding of** \( y_i \).

- A **perceptron** (Rosenblatt, 1957) is a classifier of the form:

\[
    h_{\mathbf{w}, w_0}(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + w_0) = \begin{cases} 
    +1 & \text{if } \mathbf{w} \cdot \mathbf{x} + w_0 \geq 0 \\
    -1 & \text{otherwise}
    \end{cases}
\]

Here, \( \mathbf{w} \) is a vector of weights, “\( \cdot \)” denotes the dot product, and \( w_0 \) is a constant offset. (**Note** \( x_0 = 1 \) **is omitted**.)

- The decision boundary is \( \mathbf{w} \cdot \mathbf{x} + w_0 = 0 \).

- Perceptrons output a **class**, not a probability

- An example \( \langle \mathbf{x}, y \rangle \) is classified correctly if and only if:

\[
    y \cdot (\mathbf{w} \cdot \mathbf{x} + w_0) > 0
\]
The data set is *linearly separable* if and only if there exists \( w, w_0 \) such that:
- For all \( i \), \( y_i(w \cdot x_i + w_0) > 0 \).
- Or equivalently, the 0-1 loss is zero for some set of parameters \((w, w_0)\).
A gradient descent-like learning rule

- Consider the following procedure:
  1. Initialize $w$ and $w_0$ randomly
  2. While any training examples remain incorrectly classified
     1. Loop through all misclassified examples
     2. For misclassified example $i$, perform the updates:

\[
    w \leftarrow w + \gamma y_i x_i, \quad w_0 \leftarrow w_0 + \gamma y_i
\]

where $\gamma$ is a step-size parameter.

- The update equation, or sometimes the whole procedure, is called the \textit{perceptron learning rule}.

- Intuition: For positive examples misclassified as negative, change $w$ to increase $w \cdot x_i + w_0$, and vice versa.
Gradient descent interpretation

- The perceptron learning rule can be interpreted as a gradient descent procedure, but with the following perceptron optimization criterion:

\[
\text{Err}(\mathbf{w}, w_0) = \sum_{i=1}^{n} \begin{cases} 
0 & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 0 \\
-y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) < 0
\end{cases}
\]

- For correctly classified examples, the error is zero.
- For incorrectly classified examples, the error is by how much \( \mathbf{w} \cdot \mathbf{x}_i + w_0 \) is on the wrong side of the decision boundary.
- \( \text{Err} \) is piecewise linear, so it has a gradient almost everywhere; stochastic gradient descent gives the perceptron learning rule.
- \( J \) is zero if and only if all examples are classified correctly – just like the 0-1 loss function.
The perceptron convergence theorem states that if the perceptron learning rule is applied to a linearly separable data set, a solution will be found after some finite number of updates.

The number of updates depends on the data set, and also on the step size parameter.

If the data is not linearly separable, there will be oscillation (which can be detected automatically).
Perceptron learning example–separable data

\[ w = [0, 0] \quad w_0 = 0 \]
Perceptron learning example—separable data

\[ w = [4.111, 3.8704] \quad w_0 = -4 \]
Weight as a combination of input vectors

- Recall perceptron learning rule:

\[ \mathbf{w} \leftarrow \mathbf{w} + \gamma y_i \mathbf{x}_i, \quad w_0 \leftarrow w_0 + \gamma y_i \]

- If initial weights are zero, then at any step, the weights are a linear combination of feature vectors of the examples:

\[ \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i, \quad w_0 = \sum_{i=1}^{n} \alpha_i y_i \]

where \( \alpha_i \) is the sum of step sizes used for all updates based on example \( i \).

- This is called the *dual representation* of the classifier.

- Even by the end of training, some example may have never participated in an update, so the corresponding \( \alpha_i = 0 \).
Example used (bold) and not used (faint) in updates

\[ w = [4.111 \ 3.8704] \quad w_0 = -4 \]
Comment: Solutions are nonunique

Solutions depend on the set of instances and the order of sampling in updates.
Perceptron summary

- Perceptrons can be learned to fit linearly separable data, using a gradient descent rule.
- There are other fitting approaches – e.g., formulation as a linear constraint satisfaction problem / linear program.
- Solutions are non-unique.
- For non-linearly separable data:
  - Perhaps data can be linearly separated in a different feature space?
  - Perhaps we can relax the criterion of separating all the data?
Support Vector Machines

- Support vector machines (SVMs) for binary classification can be viewed as a way of training perceptrons.
- There are three main new ideas:
  - An alternative optimization criterion (the “margin”), which eliminates the non-uniqueness of solutions and has theoretical advantages.
  - A way of handling nonseparable data by allowing mistakes.
  - An efficient way of operating in expanded feature spaces – the “kernel trick”.
- SVMs can also be used for multiclass classification and regression.
Returning to the non-uniqueness issue

- Consider a linearly separable binary classification data set \( \{x_i, y_i\}_{i=1}^n \).
- There is an infinite number of hyperplanes that separate the classes:

Which plane is best?

Relatedly, for a given plane, for which points should we be most confident in the classification?
The margin, and linear SVMs

- For a given separating hyperplane, the *margin* is two times the (Euclidean) distance from the hyperplane to the nearest training example.

- It is the width of the “strip" around the decision boundary containing no training examples.

- A linear SVM is a perceptron for which we choose $w, w_0$ so that margin is maximized.
Distance to the decision boundary

- Suppose we have a decision boundary that separates the data.

- Let $\gamma_i$ be the distance from instance $x_i$ to the decision boundary.
- How can we write $\gamma_i$ in term of $x_i, y_i, w, w_0$?
Distance to the decision boundary (II)

- The vector $\mathbf{w}$ is normal to the decision boundary. Thus, $\frac{\mathbf{w}}{||\mathbf{w}||}$ is the unit normal.
- The vector from B to A is $\gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}$.
- B, the point on the decision boundary nearest $\mathbf{x}_i$, is $\mathbf{x}_i - \gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}$.
- As B is on the decision boundary,
  \[ \mathbf{w} \cdot \left( \mathbf{x}_i - \gamma_i \frac{\mathbf{w}}{||\mathbf{w}||} \right) + w_0 = 0 \]
- Solving for $\gamma_i$ yields, for a positive example:
  \[ \gamma_i = \frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x}_i + \frac{w_0}{||\mathbf{w}||} \]
The margin [HTF Ch. 4.5, Ch 12]

- The *margin of the hyperplane* is $2M$, where $M = \min_i \gamma_i$
- The most direct statement of the problem of finding a maximum margin separating hyperplane is thus
  \[
  \max_{w, w_0} \min_i \gamma_i \\
  \equiv \max_{w, w_0} \min_i y_i \left( \frac{w}{||w||} \cdot x_i + \frac{w_0}{||w||} \right)
  \]
- This turns out to be inconvenient for optimization, however...
Treating the $\gamma_i$ as constraints

- From the definition of margin, we have:

$$M \leq \gamma_i = y_i \left( \frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x}_i + \frac{w_0}{||\mathbf{w}||} \right) \quad \forall i$$

- This suggests:

$$\text{maximize} \quad M$$

$$\text{with respect to} \quad \mathbf{w}, w_0$$

$$\text{subject to} \quad y_i \left( \frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x}_i + \frac{w_0}{||\mathbf{w}||} \right) \geq M \quad \text{for all } i$$

- Problems:

  - $\mathbf{w}$ appears nonlinearly in the constraints.
  - This problem is underconstrained. If $(\mathbf{w}, w_0, M)$ is an optimal solution, then so is $(\beta \mathbf{w}, \beta w_0, M)$ for any $\beta > 0$. 
Adding a constraint

Let’s add the constraint that \( M = 1/\|w\| \):

\[
\begin{align*}
\text{maximize} & \quad 1/\|w\| \\
\text{with respect to} & \quad w, w_0 \\
\text{subject to} & \quad y_i \left( \frac{w}{\|w\|} \cdot x_i + \frac{w_0}{\|w\|} \right) \geq 1/\|w\| \quad \text{for all } i
\end{align*}
\]

\[
\begin{align*}
\text{maximize} & \quad 1/\|w\| \\
\text{with respect to} & \quad w, w_0 \\
\text{subject to} & \quad y_i (w \cdot x_i + w_0) \geq 1 \quad \text{for all } i
\end{align*}
\]

• This allows us to rewrite the objective function:

\[
\begin{align*}
\min & \quad \|w\| \\
\text{w.r.t.} & \quad w, w_0 \\
\text{s.t.} & \quad y_i (w \cdot x_i + w_0) \geq 1
\end{align*}
\]

• This is really nice because the constraints are linear.

• The objective function \( \|w\| \) is still a bit awkward.
Final formulation

• Let’s minimize $\|\mathbf{w}\|^2$ instead of $\|\mathbf{w}\|$. (Taking the square is a monotone transformation, as $\|\mathbf{w}\|$ is positive, so this doesn’t change the optimal solution.)

• This gets us to:
  \[
  \min_{\mathbf{w}, \mathbf{w}_0} \|\mathbf{w}\|^2 \\
  \text{w.r.t.} \quad \mathbf{w}, \mathbf{w}_0 \\
  \text{s.t.} \quad y_i (\mathbf{w} \cdot \mathbf{x}_i + \mathbf{w}_0) \geq 1
  \]

• This we can solve! How?
  • It is a convex quadratic programming (QP) problem—a standard type of optimization problem for which many efficient packages are available.
We have a solution, but no “support vectors” yet...
What are “Support Vectors”? 

- We wanted to solve (rewritten slightly):
  \[
  \min \frac{1}{2} \|w\|^2 \\
  \text{w.r.t. } w, w_0 \\
  \text{s.t. } 1 - y_i(w \cdot x_i + w_0) \leq 0
  \]

- Turns out [HTF Ch. 4.5.2] we can write:
  \[
  w = \sum_i \alpha_i y_i x_i, \text{ where } \alpha_i \geq 0
  \]

⇒ Just like for the perceptron with zero initial weights, the optimal solution for \(w\) is a linear combination of the \(x_i\), and likewise for \(w_0\).

- The output is therefore:
  \[
  h_{w,w_0}(x) = \text{sign} \left( \sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x) + w_0 \right)
  \]

⇒ Output depends on weighted dot product of input vector with training examples
Solving “the dual”

- We can actually solve directly for the $\alpha_i$ (again see [HTF Ch. 4.5.2]):

$$\max_\alpha \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j (x_i \cdot x_j)$$

with constraints: $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

- This is also a QP
The support vectors

- Suppose we find optimal $\alpha$s (e.g., using a standard QP package)
- The $\alpha_i$ will be $> 0$ only for the points for which
  $1 - y_i(w \cdot x_i + w_0) = 0$
- These are the points lying on the edge of the margin, and they are called *support vectors*, because they define the decision boundary
- The output of the classifier for query point $x$ is computed as:

$$\text{sgn} \left[ \left( \sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x) \right) + w_0 \right]$$

Hence, the output is determined by computing the *dot product of the point with the support vectors*!
Support vectors are in bold
But why all this work?

- SVMs are state-of-the-art for classification when you don’t need probability estimates.
- Intuitively, the large-margin property makes sense. Theory backs this up.
- SVMs offer “off-the-shelf” non-linear classification without having to do explicit feature construction, as we will see.
Soft margin classifiers

• Recall that in the linearly separable case, we compute the solution to the following optimization problem:
  \[
  \min_{w, w_0} \frac{1}{2} \|w\|^2 \\
  \text{w.r.t. } w, w_0 \\
  \text{s.t. } y_i(w \cdot x_i + w_0) \geq 1
  \]

• If we want to allow misclassifications, we relax the constraints to:
  \[
  y_i(w \cdot x_i + w_0) \geq 1 - \xi_i
  \]

• If \( \xi_i \in (0, 1) \), the data point is within the margin
• If \( \xi_i \geq 1 \), then the data point is misclassified
• We define the soft error as \( \sum_i \xi_i \); each \( \xi_i \) is a slack variable
• We will have to change the optimization to account for soft errors
New problem formulation with soft errors

\[
\begin{align*}
\min & \quad \frac{1}{2}\|w\|^2 \\
\text{w.r.t.} & \quad w, w_0 \\
\text{s.t.} & \quad y_i(w \cdot x_i + w_0) \geq 1
\end{align*}
\]

we want to solve:

\[
\begin{align*}
\min & \quad \frac{1}{2}\|w\|^2 + C \sum_i \xi_i \\
\text{w.r.t.} & \quad w, w_0, \xi_i \\
\text{s.t.} & \quad y_i(w \cdot x_i + w_0) \geq 1 - \xi_i, \quad \xi_i \geq 0
\end{align*}
\]

• Note that soft errors include points that are misclassified, as well as points within the margin
• There is a linear penalty for both categories
• The choice of the constant \(C\) controls overfitting
A built-in overfitting knob

\[
\begin{align*}
\min & \quad \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \\
\text{w.r.t.} & \quad w, w_0, \xi_i \\
\text{s.t.} & \quad y_i (w \cdot x_i + w_0) \geq 1 - \xi_i \\
& \quad \xi_i \geq 0
\end{align*}
\]

- If \( C \) is 0, there is no penalty for soft errors, so the focus is on maximizing the margin, even if this means more mistakes.
- If \( C \) is very large, the emphasis on the soft errors will decrease the margin, if this helps to classify more examples correctly.
- Internal cross-validation is a good way to choose \( C \) appropriately.
Dual form for the soft margin problem

- Like before, we can formulate a “dual” problem that identifies the support vectors:

- Primal form:
  \[
  \begin{align*}
  \text{min } & \quad \|w\|^2 + C \sum \xi_i \\
  \text{w.r.t. } & \quad w, w_0, \xi_i \\
  \text{s.t. } & \quad y_i(w \cdot x_i + w_0) \geq (1 - \xi_i) \\
  & \quad \xi_i \geq 0
  \end{align*}
  \]

- Dual form:
  \[
  \begin{align*}
  \text{max } & \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y_i y_j \alpha_i \alpha_j x_i \cdot x_j \\
  \text{w.r.t. } & \quad \alpha_i \\
  \text{s.t. } & \quad 0 \leq \alpha_i \leq C \\
  & \quad \sum_{i=1}^n \alpha_i y_i = 0
  \end{align*}
  \]

- All the previously described machinery can be used to solve this problem
• A linear boundary might be too simple to capture the class structure.

• One way of getting a nonlinear decision boundary in the input space is to find a linear decision boundary in an expanded space (e.g., for polynomial regression.)

• Thus, \( x_i \) is replaced by \( \phi(x_i) \), where \( \phi \) is called a feature mapping.
Separability by adding features
Separability by adding features

\[(w_0 + w(1)x + w(2)x^2)\]
Separability by adding features

\[(w_0 + w(1)x + w(2)x^2)\]

more flexible decision boundary \(\approx\) enriched feature space
Margin optimization in feature space

- Replacing $x_i$ with $\phi(x_i)$, the dual form becomes:
  $$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \phi(x_i) \cdot \phi(x_j)$$
  w.r.t. $\alpha_i$
  s.t. $0 \leq \alpha_i \leq C$
  $\sum_{i=1}^{n} \alpha_i y_i = 0$

- Classification of an input $x$ is given by:
  $$h_{w,w_0}(x) = \text{sign} \left( \sum_{i=1}^{n} \alpha_i y_i \phi(x_i) \cdot \phi(x) + w_0 \right)$$

⇒ Note that in the dual form, to do both SVM training and prediction, we only ever need to compute dot-products of feature vectors.
Kernel functions

- Whenever a learning algorithm (such as SVMs) can be written in terms of dot-products, it can be generalized to kernels.

- A *kernel* is any function $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ which corresponds to a dot product for some feature mapping $\phi$:

  \[ K(x_1, x_2) = \phi(x_1) \cdot \phi(x_2) \text{ for some } \phi. \]

- Conversely, by choosing feature mapping $\phi$, we implicitly choose a kernel function.

- Recall that $\phi(x_1) \cdot \phi(x_2) \propto \cos \angle(x_1, x_2)$ where $\angle$ denotes the angle between the vectors, so a kernel function can be thought of as a notion of *similarity*. 
Example: Quadratic kernel

- Let $K(x, z) = (x \cdot z)^2$.
- Is this a kernel?

\[
K(x, z) = \left( \sum_{i=1}^{p} x_i z_i \right) \left( \sum_{j=1}^{p} x_j z_j \right) = \sum_{i,j \in \{1...p\}} x_i z_i x_j z_j \\
= \sum_{i,j \in \{1...p\}} (x_i x_j) (z_i z_j)
\]

- Hence, it is a kernel, with feature mapping:

\[
\phi(x) = \langle x_1^2, x_1 x_2, \ldots, x_1 x_p, x_2 x_1, x_2^2, \ldots, x_p^2 \rangle
\]

Feature vector includes all squares of elements and all cross terms.
- Note that computing $\phi$ takes $O(p^2)$ but computing $K$ takes only $O(p)$!
Polynomial kernels

- More generally, $K(x, z) = (x \cdot z)^d$ is a kernel, for any positive integer $d$:
  $$K(x, z) = \left( \sum_{i=1}^{n} x_i z_i \right)^d$$

- If we expanded the sum above in the obvious way, we get $n^d$ terms (i.e. feature expansion)
- Terms are monomials (products of $x_i$) with total power equal to $d$.
- If we use the primal form of the SVM, each of these will have a weight associated with it!
- **Curse of dimensionality**: it is very expensive both to optimize and to predict with an SVM in primal form
- However, *evaluating the dot-product of any two feature vectors can be done using $K$ in $O(n)$!*
The “kernel trick"

- If we work with the dual, we do not actually have to ever compute the feature mapping $\phi$. We just have to compute the similarity $K$.
- That is, we can solve the dual for the $\alpha_i$:
  \[
  \max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i, x_j)
  \]
  w.r.t. $\alpha_i$
  s.t. $0 \leq \alpha_i \leq C$
  \[
  \sum_{i=1}^{n} \alpha_i y_i = 0
  \]
- The class of a new input $x$ is computed as:
  \[
  h_{w,w_0}(x) = \text{sign} \left( \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + w_0 \right)
  \]
- Often, $K(\cdot, \cdot)$ can be evaluated in $O(p)$ time—a big savings!
Some other (fairly generic) kernel functions

- \( K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})^d \) – feature expansion has all monomial terms of \( \leq d \) total power.

- Radial basis/Gaussian kernel:

  \[
  K(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2/2\sigma^2)
  \]

  The kernel has an infinite-dimensional feature expansion, but dot-products can still be computed in \( O(n) \)!

- Sigmoidal kernel:

  \[
  K(\mathbf{x}, \mathbf{z}) = \tanh(c_1 \mathbf{x} \cdot \mathbf{z} + c_2)
  \]
Example: Gaussian kernel

Note the non-linear decision boundary
Kernels beyond SVMs

- Remember, a kernel is a special kind of similarity measure.
- A lot of current research has to do with defining new kernels functions, suitable to particular tasks / kinds of input objects.
- Many kernels are available:
  - Information diffusion kernels (Lafferty and Lebanon, 2002)
  - Diffusion kernels on graphs (Kondor and Jebara 2003)
  - String kernels for text classification (Lodhi et al, 2002)
  - String kernels for protein classification (e.g., Leslie et al, 2002)

... and others!
Example: String kernels

- Very important for DNA matching, text classification, ...
- Example: in DNA matching, we use a sliding window of length $k$ over the two strings that we want to compare
- The window is of a given size, and inside we can do various things:
  - Count exact matches
  - Weigh mismatches based on how bad they are
  - Count certain markers, e.g. AGT
- The kernel is the sum of these similarities over the two sequences
- How do we prove this is a kernel?
- Many other machine learning algorithms have a “dual formulation”, in which dot-products of features can be replaced with kernels.
Second brush with “feature construction”

- With polynomial regression, we saw how to construct features to increase the size of the hypothesis space.
- This gave more flexible regression functions.
- Kernels offer a similar function with SVMs: More flexible decision boundaries.
- Often not clear what kernel is appropriate for the data at hand.
“This paper investigates the potential of support vector machines (SVM)-based classification approach to assess the liquefaction potential from actual standard penetration test (SPT) and cone penetration test (CPT) field data. SVMs are based on statistical learning theory and found to work well in comparison to neural networks in several other applications.”

- SPT Data: 85 instances, 59 for train, 26 for test
- CPT Data: 109 instances, 74 for train, 35 for test
- Tried two kernels, \( K(x, x') = (x^T x' + 1)^d \) and \( K(x, x') = \exp(-\gamma ||x - x'||^2) \)

“Thus, the choice of parameters \( d \) and \( \gamma \) associated with both kernels as well as the value of \( C \) may affect the level of classification accuracies achieved with SVMs. A number of trials were carried out to reach at a suitable value of parameters \( d, \gamma \) and \( C \) using [the SPT features] as input parameter with SPT data set and [the CPT features] as input parameters with CPT data set. Further, classifications were carried out using different combination of
Truth in advertising?

- They also tried a number of different feature subsets while varying $C$, $\gamma$, and $d$, for a total of at least 10 models, probably more.
- Results: Best two models gave 96.15% and 97.14% on the “test set.”
- **Pal Claim:** Better than the Neural Networks by Goh (1994), which achieve 92%.
  
  http://ascelibrary.org/gto/resource/1/jgendz/v120/i9/p1467_s1
  
  - Goh does the same thing: optimizing on the “test set.”
- **Goh claim:** Better than the single SPT feature by Seed et al. (1985), which gave 84%.
  
  http://ascelibrary.org/gto/resource/1/jgendz/v120/i9/p1467_s1
- **Who do you believe?**
Getting SVMs to work in practice

- Two important choices:
  - Kernel (and kernel parameters)
  - Regularization parameter $C$
- The parameters may interact!
  E.g. for Gaussian kernel, the larger the width of the kernel, the more biased the classifier, so low $C$ is better
- Together, these control overfitting: always do a *within-fold parameter search, using a validation set*!
- Clues you might be overfitting:
  - Low margin (large weights)
  - Large fraction of instances are support vectors
Application: Ovarian cancer classification (Furey, 2000)

http://bioinformatics.oxfordjournals.org/content/16/10/906

- E.g. Ovarian cancer data - 31 tissue samples of 3 classes
- 97,802 features per sample. Yikes
- Idea: Rank according to

\[
F(x_j) = \left| \frac{\mu_j^+ - \mu_j^-}{\sigma_j^+ + \sigma_j^-} \right|
\]

within each fold. Then give to an SVM.