CS 6140: Machine Learning
Spring 2017

Instructor: Lu Wang
College of Computer and Information Science
Northeastern University
Webpage: www.ccs.neu.edu/home/luwang
Email: luwang@ccs.neu.edu

Logistics

• Assignment 1 is out
  – Due 2/9/2017
  – Start early!

What we learned last time

• Evaluation metrics
• Decision Tree
• Generative Models
• Generative Model and Discriminative Model
• Logistic Regression

Confusion Matrix

accuracy = (a+d) / (a+b+c+d)

Precision/Recall

Precision = a / (a + c)
Recall = a / (a + b)
ROC Plot

- Sensitivity = a/(a+b) = Recall
  - True positive rate
- 1 - Specificity = 1 - d/(c+d) = c/(c+d)
  - False positive rate

<table>
<thead>
<tr>
<th>Pred. 1</th>
<th>Pred. 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>True:</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>False:</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Decision Tree

- Play tennis?

  - Each internal node: test one feature $X_i$
  - Each branch from a node: selects one value for $X_i$
  - Each leaf node: predict $Y$ (or $P(Y|X \in \text{leaf})$)

Top-Down Induction of Decision Trees

- Which attribute to use for split?

  Would we prefer to split on $X_1$ or $X_2$?

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Information Gain

- Gain$(S,A)=$expected reduction in entropy due to sorting on $A$

  $Entropy(S) - \sum_{c \in \text{Values}(A)} \frac{|S_c|}{|S|} Entropy(S_c)$
**Information Gain**

\[ \text{Entropy}(S) = -\sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v) \]

Gain (H, Rain) = \(-0.302\)
Gain (H, Wind) = \(-0.414\)

**Naïve Bayes**

- **Given:**
  - Prior \( P(Y) \)
  - \( n \) conditionally independent features \( X_1, \ldots, X_n \) given the class \( Y \)
  - For each feature \( i \), we specify \( P(X_i | Y) \)

- **Classification decision rule:**
  \[
  \hat{y} = h_N(x) = \arg \max_y P(y) P(x_1, \ldots, x_n | y) = \arg \max_y P(y) \prod_i P(x_i | y)
  \]

If certain assumption holds, NB is optimal classifier! (they typically don’t)

**Overfitting**

**Bayesian model**

- \( H \): Hypothesis space of possible concepts
- \( X \): \( n \) examples of a concept \( C \)
- Evaluate hypotheses given data using Bayes’ rule:
  \[
  p(h | X) = \frac{p(X | h) p(h)}{\sum_{h \in H} p(X | h) p(h)}
  \]
  - \( p(h) \) ("prior"): domain knowledge, pre-existing biases
  - \( p(X | h) \) ("likelihood"): statistical information in examples.
  - \( p(h | X) \) ("posterior"): degree of belief that \( h \) is the true extension of \( C \).

**Maximum Likelihood Estimation**

- **Given dataset**
  - \( \text{Count}(A=a, B=b) \) \( \leftarrow \) number of examples where \( A=a \) and \( B=b \)
- **MLE for discrete NB, simply:**
  - Prior:
    \[
    P(Y = y) = \frac{\text{Count}(Y = y)}{\sum_y \text{Count}(Y = y)}
    \]
  - Observation distribution:
    \[
    P(X_i = x | Y = y) = \frac{\text{Count}(X_i = x, Y = y)}{\sum_{x'} \text{Count}(X_i = x', Y = y)}
    \]
**Maximum Likelihood Estimation**

- Given dataset
  - \( \text{Count}(A=a, B=b) \) \( \rightarrow \) number of examples where \( A=a \) and \( B=b \)
- MAP estimation for discrete NB, simply:
  - Prior:
    \[ P(Y = y) = \frac{\text{Count}(Y = y)}{\sum_{y'} \text{Count}(Y = y')} \]
  - Observation distribution:
    \[ P(X_i = x|Y = y) = \frac{\text{Count}(X_i = x, Y = y) + \alpha}{\sum_{x'} \text{Count}(X_i = x', Y = y) + \alpha} \]
- Called "smoothing". Corresponds to Dirichlet prior!

**Generative VS. Discriminative Model**

- \( P(Y|X) = \frac{P(X,Y)}{P(X)} \)
- Generative model
  - Learn \( P(X, Y) \) from training sample
  - \( P(X, Y) = P(Y)P(X|Y) \)
  - Specifies how to generate the observed features \( x \) for \( y \)
- Discriminative model
  - Learn \( P(Y|X) \) from training sample
  - Directly models the mapping from features \( x \) to \( y \)

**Logistic Regression**

*Learn \( P(Y|X) \) directly!*

- Assume a particular functional form
- Sigmoid applied to a linear function of the data:

\[
\begin{align*}
P(Y = 1|X) &= \frac{1}{1 + \exp(w_0 + \sum w_i x_i)} \\
P(Y = 0|X) &= \frac{\exp(w_0 + \sum w_i x_i)}{1 + \exp(w_0 + \sum w_i x_i)}
\end{align*}
\]

**Sigmoid function**

- Definition
  \[ \text{sign}(\eta) \triangleq \frac{1}{1 + e^{-\eta}} = \frac{e^\eta}{e^\eta + 1} \]

**Logistic Regression**

*Prediction: Output the \( Y \) with highest \( P(Y|X) \)*

- For binary \( Y \), output \( Y=0 \) if
  \[ 1 < \frac{P(Y = 0|X)}{P(Y = 1|X)} \quad 1 < \exp(w_0 + \sum w_i x_i) \]
  \[ 0 < w_0 + \sum w_i x_i \]

*A Linear Classifier!"
Maximizing Log Likelihood

\[
I(w) = \ln \prod_{i=1}^{n} \frac{1}{1 + e^{w_0 + \sum_{j} w_j x_j}}
\]

\[
= \sum_{j} y_j (w_0 + \sum_{j} w_j x_j) - \ln(1 + \exp(w_0 + \sum_{j} w_j x_j))
\]

0 or 1

Bad news: no closed-form solution to maximize \(I(w)\)

Good news: \(I(w)\) is concave function of \(w\)

No local maxima

Concave functions easy to optimize

**Gradient Descent**

- Example

\[
f(\theta) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2
\]

\[\eta = 0.1\]

\[\eta = 0.6\]

**Changing Step Size**

- Conditional likelihood for Logistic Regression is concave →

\[
\nabla_w I(w) = \frac{\partial I(w)}{\partial w_0}, \ldots, \frac{\partial I(w)}{\partial w_j}
\]

Update rule:

\[
\Delta w = \eta \nabla_w I(w)
\]

\[
w_i^{(t+1)} = w_i^{(t)} + \eta \frac{\partial I(w)}{\partial w_i}
\]

Gradient ascent algorithm: (learning rate \(\eta > 0\))

\[
do:\]

\[
w_i^{(t+1)} = w_i^{(t)} + \eta \sum_{j} [y_j - P(Y = 1 \mid x_i, w)]
\]

For \(i = 1 \to n\): (iterate over features)

\[
w_i^{(t+1)} = w_i^{(t)} + \eta \sum_{j} [y_j - P(Y = 1 \mid x_i, w)]
\]

until "change" < \(\epsilon\)

Loop over training examples

(could also do stochastic GD)
Adding Prior

\[ p(w \mid Y, X) \propto P(Y \mid X, w)p(w) \]

- **One common approach is to define priors on** \( w \)
  - Normal distribution, zero mean, identity covariance
  - “Pushes” parameters towards zero
  \[ p(w) = \prod_{i=1}^{\lambda} \frac{1}{\sqrt{2\pi}} e^{-w_i^2/2} \]
- **Regularization**
  - Helps avoid very large weights and overfitting
- **MAP estimate:**
  \[ w^* = \arg \max_w \ln \left[ p(w) \prod_{j=1}^{\lambda} P(y_j \mid x_i, w) \right] \]

- Adds log \( p(w) \) to objective:
  \[ \ln p(w) \propto -\frac{1}{2} \sum_{i=1}^{\lambda} w_i^2 \]
  \[ \frac{\partial \ln p(w)}{\partial w_i} = -\lambda w_i \]
  - Quadratic penalty: drives weights towards zero
  - Adds a negative linear term to the gradients

Today’s Outline

- Perceptron (and kernels)
- Support Vector Machines

Perceptron

[Some of the slides are borrowed from Alex Smola’s tutorial]

Biology and Learning

- Idea 1: Good behavior should be rewarded, bad behavior punished (or not awarded).
  - Raising a dog.
- Idea 2: Correlated events should be combined.
  - Babies learn language.

Biology and Learning

- Idea 1: Good behavior should be rewarded, bad behavior punished (or not awarded).
  - Raising a dog.
- Idea 2: Correlated events should be combined.
  - Babies learn language.
- Training Mechanisms
  - Behavioral modification of individuals (learning)
    - Feeding the dog, then the dog learns to stand and sit.
  - Hard-coded behavior in the genes (instinct)
    - The wrongly coded animal dies.
Neurons

- **Soma**: Cell body. Here the signals are combined ("CPU").
- **Dendrite**: Combines the inputs from several other nerve cells ("input bus").
- **Synapse**: Interface between two neurons ("connector").
  - This may be up to 1 μm long and will transport the activation signal to nerve cells at different locations ("output cable").

Perceptron

- **Weighted combination**
  - The output of the neuron is a linear combination of the inputs.
- **Decision Function**
  - At the end the results are combined into:
    \[ f(x) = \sigma \left( \sum_{i=1}^{n} w_i x_i + b \right) \]
  - Where \( w \) is the weight, \( x \) is the feature vectors, \( W, X \in \mathbb{R}^m \), \( b \) is the bias, \( b \in \mathbb{R} \).
- **Biological Interpretation**
  - The weights \( w \) correspond to the synaptic weights, the multiplication corresponds to the processing of inputs via the synapses, and the summation is the combination of signals in the cell body (soma).

Learning Goal: Linear Separation

\[ f(x) = \langle w, x \rangle + b \]

Perceptron Algorithm

- **argument**: \( X := \{x_1, \ldots, x_m\} \subset \mathcal{X} \) (data)
- **Y**: \( \{y_1, \ldots, y_m\} \subset \{\pm 1\} \) (labels)
- **function** \( (w, b) = \text{Perceptron}(X, Y) \)
  - **initialize** \( w, b = 0 \)
  - **repeat**
    - Pick \( (x, y) \) from data
    - if \( y (w \cdot x_i + b) \leq 0 \) then
      - \( w' = w + y x_i \)
      - \( b' = b + y \)
    - until \( y_i (w \cdot x_i + b) > 0 \) for all \( i \)
  - **end**
Perceptron Algorithm

• Nothing happens if we classify \((x_i, y_i)\) correctly
• If we see incorrectly classified observation we update \(w\) and \(b\)
• Positive reinforcement of observations
  
  argument: \(X := \{x_1, ..., x_n\} \subseteq \mathbb{R}^d\) (data)
  \(Y := \{y_1, ..., y_n\} \subseteq \{\pm 1\}\) (labels)
  function \((w, b) \rightarrow \text{Perceptron}(X, Y)\)
  initialize \(w, b = 0\)
  repeat
  \(\text{Pick } (x, y) \text{ from data}\)
  \(\text{if } y(w \cdot x + b) \leq 0 \text{ then}\)
  \(w' = w + yx\)
  \(b' = b + y\)
  until \(y(w \cdot x + b) > 0\) for all \(i\)
  end

Perceptron Algorithm

• About the solution
  
  Weight vector is linear combination of observations \(x_i\):
  \[w \leftarrow w + y_i x_i\]
  
  argument: \(X := \{x_1, ..., x_n\} \subseteq \mathbb{R}^d\) (data)
  \(Y := \{y_1, ..., y_n\} \subseteq \{\pm 1\}\) (labels)
  function \((w, b) \rightarrow \text{Perceptron}(X, Y)\)
  initialize \(w, b = 0\)
  repeat
  \(\text{Pick } (x, y) \text{ from data}\)
  \(\text{if } y(w \cdot x + b) \leq 0 \text{ then}\)
  \(w' = w + yx_i\)
  \(b' = b + y\)
  until \(y(w \cdot x + b) > 0\) for all \(i\)
  end

Geometrical Interpretation

![Geometrical Interpretation](image)

[Slide by Rong Jin]

Problems with the perceptron algorithm

• If the data isn’t linearly separable, no guarantees of convergence or training accuracy
• Even if the training data is linearly separable, perceptron can overfit
• Averaged perceptron is an algorithmic modification that helps with both issues
  
  – Averages the weight vectors across all iterations

Pseudocode

```python
for i in range(m):
    ytest = numpy.dot(w, x[:,i]) + b
    if ytest * y[i] <= 0:
        w += y[i] * x[:,i]
        b += y[i]
```
The XOR Problem

- Perceptrons cannot learn such linearly inseparable functions!

Problem

- Linear functions are often too simple to provide good estimators.

- Idea:
  - Map to a higher dimensional feature space via $\Phi: x \rightarrow \Phi(x)$
  - Replace every $\langle x, x' \rangle$ by $\langle \Phi(x), \Phi(x') \rangle$ in the perceptron algorithm.
Perceptron on Features

argument: \( X := \{ x_1, \ldots, x_n \} \subseteq \mathbb{X} \) (data)
\( Y := \{ y_1, \ldots, y_n \} \subseteq \{ \pm 1 \} \) (labels)

function \((w, b) = \text{Perceptron}(X, Y, \alpha)\)
Initialize \( w, b = 0 \)
repeat
Pick \((x_i, y_i)\) from data
if \( y_i(w \cdot \Phi(x_i) + b) \leq 0 \) then
    \( w' = w + y_i \Phi(x_i) \)
    \( b' = b + y_i \)
until \( y_i(w' \cdot \Phi(x_i) + b) > 0 \) for all \( i \)

Important detail
\( w = \sum_i y_i \Phi(x_i) \) and hence \( f(x) = \sum_i (y_i \Phi(x_i) \cdot \Phi(x)) + b \)

Problems with Constructing Features

- Need to be an expert in the domain (e.g. Chinese characters).
- Can be expensive to compute.

Polynomial Features

- Dimension = 1
  \( \phi(u), \phi(v) = \left( \begin{array}{c} u_1 \\ u_2 \end{array} \right) \cdot \left( \begin{array}{c} v_1 \\ v_2 \end{array} \right) = u_1 v_1 + u_2 v_2 = u \cdot v \)
- Dimension = 2
  \( \phi(u), \phi(v) = \left( \begin{array}{c} u_1^2 \\ u_1 u_2 \\ u_2 \end{array} \right) \cdot \left( \begin{array}{c} v_1^2 \\ v_1 v_2 \\ v_2 \end{array} \right) = u_1^2 v_1^2 + 2u_1 u_2 v_1 v_2 + u_2^2 v_2^2 = (u \cdot v)^2 \)
- Dimension = \( d \) (skip proof)
  \( \phi(u), \phi(v) = (u \cdot v)^d \)

Some choices of kernel functions

- Linear
  \( \langle x, x' \rangle \)
- Laplacian RBF
  \( \exp(-\lambda \|x - x'\|) \)
- Gaussian RBF
  \( \exp(-\lambda \|x - x'\|^2) \)
- Polynomial
  \( \langle (x \cdot x') + c \rangle^d, c \geq 0, d \in \mathbb{N} \)

RBF kernel: Radial basis function kernel

Kernels

- Definition
- A kernel function \( k : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R} \) is a symmetric function in its arguments for which the following property holds
  \( k(x, x') = \langle \Phi(x), \Phi(x') \rangle \) for some feature map \( \Phi \)
Which of these linear separators is optimal?

argumt: $X \in \{x_1, \ldots, x_n\} \subset \mathbb{X}$ (data)
$Y \in \{y_1, \ldots, y_n\} \subset \{\pm 1\}$ (labels)
function $f = \text{Perceptron}(X, Y, \eta)$
initialize $I = 0$
repeat
pick $(x_i, y_i)$ from data
if $y_i f(x_i) \leq 0$ then
$f(x) = f(x) + y_i k(x_i, x) + y_i$
until $y_i f(x) > 0$ for all $i$
end

$w = \sum k(x_i)$ and hence $f(x) = \sum k(x_i, x) + b$.
Outline

- Perceptron (and kernels)
- Support Vector Machines

Support Vector Machine (SVM)

- SVMs (Vapnik, 1990’s) choose the linear separator with the largest margin.

- Reasons:
  - Intuition
  - Theoretical guarantee (skip here)
  - In practical tasks: SVM became famous when, using images as input, it gave accuracy comparable to neural-network with hand-designed features in a handwriting recognition task.

Planes

- A plane can be specified as the set of points given by
  
  \[ p = a + su + tv, \quad (s, t) \in \mathbb{R}, \]

  Vector from origin to a point in the plane
  Two non-parallel directions in the plane

- A plane can be specified as the set of points given by
  
  \[ p = a + su + tv, \quad (s, t) \in \mathbb{R}. \]

  Vector from origin to a point in the plane
  Two non-parallel directions in the plane
  Normal Vector: decide the direction of the plane
Normal to a plane

Scale invariance

What is the distance $\gamma$?

Final result: can maximize margin by minimizing $||w||$

Support vector machines

What if the data is not linearly separable?
What if the data is not linearly separable?

• More features

\[ x^{(1)}, \ldots, x^{(m)} \] \quad m features

\[ y_i \in \{-1, +1\} \] \quad class

\[ \phi(x) = \begin{pmatrix} x^{(1)} \\ \vdots \\ x^{(D)}(1) \\ x^{(D)}(2) \\ \vdots \\ x^{(D)}(m) \end{pmatrix} \]

Old objective

\[ \min_{w, b} \quad w^T w \]

\[ (w \cdot x_j + b) y_j \geq 1, \forall j \]

New objective

\[ \min_{w, b} \quad w^T w + C \#(\text{mistakes}) \]

\[ (w \cdot x_j + b) y_j \geq 1, \forall j \]

Jointly minimize \( w \cdot w \) and number of training mistakes!

Allowing for slack: “Soft margin SVM”

\[ \min_{w, b} \quad w^T w + C \sum \zeta_j \]

\[ (w \cdot x_j + b) y_j \geq 1 - \zeta_j, \forall j, \zeta_j \geq 0 \]

Slack penalty \( C > 0 \):

\[ C \rightarrow \infty \rightarrow \text{have to separate the data!} \]

\[ C \rightarrow 0 \rightarrow \text{ignores the data entirely!} \]

Select using validation data

For each data point:

• If margin \( \geq 1 \), don’t care
• If margin < 1, pay linear penalty

Allowing for slack: “Soft margin SVM”

\[ \min_{w, b} \quad w^T w + C \sum \zeta_j \]

\[ (w \cdot x_j + b) y_j \geq 1 - \zeta_j, \forall j, \zeta_j \geq 0 \]

Slack variables

\[ \zeta_j \rightarrow \max(0, 1 - (w \cdot x_j + b) y_j) \]

What is the (optimal) value of \( \zeta_j \) as a function of \( w \) and \( b \)?

If \( (w \cdot x_j + b) y_j \geq 1 \), then \( \zeta_j = 0 \)

If \( (w \cdot x_j + b) y_j < 1 \), then \( \zeta_j = 1 - (w \cdot x_j + b) y_j \)
Popular Tools for SVMs

• LIBSVM (c++)
  – https://www.csie.ntu.edu.tw/~cjlin/libsvm/

• SVM light (c)
  – http://svmlight.joachims.org/

• Scikit-learn (python)
  – http://scikit-learn.org

How do we optimize the objective?

• Quadratic programming

\[
\begin{align*}
\text{argmin}_{w, \xi_i \geq 0} & \quad \frac{1}{2} ||w||^2 + C \sum_{i=1}^{m} \xi_i \\
\text{s.t.} & \quad \forall i, \ y_i \langle w, x_i \rangle \geq 1 - \xi_i
\end{align*}
\]

More "natural" form:

\[
\begin{align*}
\text{argmin}_{w} & \quad \frac{\lambda}{2} ||w||^2 + \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i \langle w, x_i \rangle)
\end{align*}
\]

\[
\begin{align*}
f(w) & \approx \frac{\lambda}{2} ||w||^2 + \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i \langle w, x_i \rangle)
\end{align*}
\]

Kernels

• Definition
  - A kernel function \( k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \) is a symmetric function in its arguments for which the following property holds

\[
k(x, x') = \langle \Phi(x), \Phi(x') \rangle \text{ for some feature map } \Phi
\]

Constrained optimization

\[
\begin{align*}
\min_x & \quad x^2 \\
\text{s.t.} & \quad x \geq b
\end{align*}
\]

No place to apply the kernel trick
Constrained optimization

\[
\min_x x^2 ~ \text{s.t.} ~ x \geq b
\]

No Constraint

\[
\begin{align*}
\text{x} \geq -1 \\
\text{x} \geq 1
\end{align*}
\]

Lagrange multipliers – Dual variables

• Lagrange Multipliers

\[
\min_x x^2 \quad \text{s.t.} \quad x \geq b
\]

Introduce Lagrangian (objective):

\[
L(x, \alpha) = x^2 - \alpha(x - b)
\]

Why is this equivalent?

- \( \min \) is fighting max!
  - \( x \geq 0 \Rightarrow (x \cdot 0) = 0 \Rightarrow \max_{x \geq 0} \alpha(x-b) = \infty \)
  - \( \alpha \) won’t let this happen!

\[
\begin{align*}
\text{x} \geq 0 & \Rightarrow \text{a} \geq 0 \\
\text{x} \geq b & \Rightarrow \alpha \geq 0
\end{align*}
\]

\[
\text{x} \geq 0 \Rightarrow \alpha \geq 0
\]

Back to SVM (hard margin)

Original optimization problem:

\[
\begin{align*}
\text{minimize}_{w, b} & ~ \frac{1}{2}w \cdot w \\
\text{s.t.} & ~ (w \cdot x_j + b) y_j \geq 1, \forall j
\end{align*}
\]
Dual SVM derivation

Original optimization problem:

\[ \begin{align*}
\text{minimize}_{w, b} & \quad \frac{1}{2} w \cdot w \\
\text{subject to} & \quad \sum_j \alpha_j \left( \langle w \cdot x_j + b \rangle y_j - 1 \right) \geq 1, \quad \forall j
\end{align*} \]

Lagrangian:

\[ L(w, \alpha) = \frac{1}{2} w \cdot w - \sum_j \alpha_j \left( \langle w \cdot x_j + b \rangle y_j - 1 \right) \]
\[ \alpha_j \geq 0, \quad \forall j \]

Our goal now is to solve:

\[ \min_{w, b} \max_{\alpha \geq 0} L(w, \alpha) \]

Dual SVM derivation

\[ \begin{align*}
\text{(Dual)} \quad & \max_{\alpha \geq 0} \min_{w, b} \frac{1}{2} \|w\|_2^2 - \sum_j \alpha_j \left( \langle w \cdot x_j + b \rangle y_j - 1 \right) \\
\text{Can solve for optimal } w, b \text{ as function of } \alpha : & \quad \nabla L = 0 \\
\frac{\partial L}{\partial w} = -\sum_j \alpha_j x_j & \Rightarrow w = \sum_j \alpha_j y_j x_j \\
\frac{\partial L}{\partial b} = -\sum_j \alpha_j y_j & \Rightarrow \sum_j \alpha_j y_j = 0 \\
\sum_{\alpha_j \geq 0} & \left( \sum_j \alpha_j \right) \frac{1}{2} \|w\|_2^2 - \sum_j \alpha_j \left( \langle w \cdot x_j + b \rangle y_j - 1 \right) \\
\text{Substituting these values back in (and simplifying), we obtain:} & \quad \sum_{\alpha_j \geq 0} \sum_j \alpha_j \frac{1}{2} \sum_{i \neq j} \alpha_i \alpha_j \langle x_i \cdot x_j \rangle
\end{align*} \]

So, in dual formulation we will solve for \( \alpha \) directly:

- \( w \) and \( b \) are computed from \( \alpha \) (if needed)

To get \( w \) and \( b \)

Lagrangian:

\[ L(w, \alpha) = \frac{1}{2} w \cdot w - \sum_j \alpha_j \left( \langle w \cdot x_j + b \rangle y_j - 1 \right) \]
\[ \alpha_j \geq 0, \quad \forall j \]

\[ \alpha_j > 0 \text{ for some } j \text{ implies constraint is tight. We use this to obtain } b : \]
\[ y_j \left( \langle a^* \cdot x_j + b \rangle - 1 \right) = 1 \quad (1) \]
\[ y_j b_j = \frac{1}{2} \langle a^* \cdot x_j + b \rangle \quad (2) \]
\[ y_j \left( \langle a^* \cdot x_j + b \rangle - y_j \right) = 0 \quad (3) \]

\[ w = \sum_j \alpha_j y_j x_j \]
\[ b = y_j - w \cdot x_j \]

Classification rule using dual solution

Using dual solution dot product

\[ y = \text{sign} \left( \sum_j \alpha_j \langle x_i \cdot x_j \rangle + b \right) \]

\[ w = \sum_j \alpha_j y_j x_j \]

\[ b = y_j - w \cdot x_j \]

for any \( x_i \) where \( a_i > a_k > 0 \)
Dual for the non-separable case

Primal:
\[ \text{minimize}_{w,b} \quad \frac{1}{2} w^T w + C \sum_j \xi_j \]
\[ \text{subject to} \quad (w \cdot x_j + b) y_j \geq 1 - \xi_j, \quad \forall j \]
\[ \xi_j \geq 0, \quad \forall j \]

Dual:
\[ \text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j \]
\[ \text{subject to} \quad \sum_i \alpha_i y_i = 0 \]
\[ C \geq \alpha_i \geq 0 \]

Solve for \( w, b, \alpha \):
\[ w = \sum_i \alpha_i y_i x_i \]
\[ b = y_k - w \cdot x_k \]
for any \( k \) where \( C > \alpha_k > 0 \)

What changed?
- Added upper bound of \( C \) on \( \alpha_j \)
- Intuitive explanation:
  - Without slack, \( \alpha_j \rightarrow \infty \) when constraints are violated (points misclassified)
  - Upper bound of \( C \) limits the \( \alpha_j \), so misclassifications are allowed

How to interpret dual form

Final solution tends to be sparse
- \( \alpha_j = 0 \) for most \( j \)
- Don’t need to store these points to compute \( w \) or make predictions

Support Vectors:
- \( d \geq 0 \)
- Moving them will not change \( w \)

Non-support Vectors:
- \( \alpha_j \neq 0 \)
- \( \Phi \) mapping features of features
- Feature space can get really large really quickly!

For example: Higher order polynomials

\[ \text{num. terms} = \binom{d + m - 1}{d} = \frac{(d + m - 1)!}{d!(m - 1)!} \]

m – input features
\( d \) – degree of polynomial

- Grows fast
- \( d = 6, m = 100 \)
- About 1.6 billion terms

Dual formulation only depends on dot-products of the features!

First, we introduce a feature mapping:
\[ x_i x_j \rightarrow \Phi(x_i) \cdot \Phi(x_j) \]

Next, replace the dot product with an equivalent kernel function:
\[ \text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j) \]
\[ K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \]
\[ \sum_i \alpha_i y_i = 0 \]
Dual formulation only depends on dot-products of the features!

First, we introduce a feature mapping:

\[ x_i(x_j) \rightarrow \Phi(x_i) \cdot \Phi(x_j) \]

Next, replace the dot product with an equivalent kernel function:

\[
\begin{align*}
\text{maximize } & \alpha, \quad \alpha_i = 0 \quad \rightarrow \\
& \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\
& K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \\
& \sum_i \alpha_i y_i = 0
\end{align*}
\]

Kernels

- Definition
- A kernel function \( k : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R} \) is a symmetric function in its arguments for which the following property holds:
  \[ k(x, x') = (\Phi(x), \Phi(x')) \]
for some feature map \( \Phi \)

Kernel Trick

\[
\begin{align*}
d=1 & \quad \phi(u), \phi(v) = \left( \begin{array}{c} u_1 \\ u_2 \end{array} \right) \cdot \left( \begin{array}{c} v_1 \\ v_2 \end{array} \right) = u_1 v_1 + u_2 v_2 = u \cdot v \\
n=2 & \quad \phi(u), \phi(v) = \left( \begin{array}{c} u_1^2 \\ u_1 u_2 \\ u_2^2 \end{array} \right) \cdot \left( \begin{array}{c} v_1^2 \\ v_1 v_2 \\ v_2^2 \end{array} \right) = u_1^2 v_1^2 + 2u_1 u_2 v_1 v_2 + u_2^2 v_2^2 \\
& \quad = (u \cdot v)^2
\end{align*}
\]

For any \( d \) (we will skip proof):

\[ \phi(u), \phi(v) = (u \cdot v)^d \]

Soft margin SVM with kernel

\[
\begin{align*}
\text{maximize } & \alpha, \quad \alpha_i = 0 \quad \rightarrow \\
& \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\
& K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \\
& \sum_i \alpha_i y_i = 0 \\
& C \geq \alpha_i \geq 0
\end{align*}
\]

- Never compute features explicitly!!!
  - Compute dot products in closed form
  - \( O(n^2) \) time in size of dataset to compute objective
  - much work on speeding up

Common kernels for SVM

- Polynomials of degree exactly \( d \):
  \[ K(u, v) = (u \cdot v)^d \]
- Polynomials of degree up to \( d \):
  \[ K(u, v) = (u \cdot v + 1)^d \]
- Gaussian kernels:
  \[ K(\hat{x}, \hat{y}) = \exp \left( \frac{-\|\hat{x} - \hat{y}\|^2}{2\sigma^2} \right) \]
  Aka Gaussian Radial basis function (RBF) kernel

Overfitting

- Huge feature space with kernels: should we worry about overfitting?
  - SVM objective seeks a solution with large margin
  - Good theoretical guarantee
  - But everything overfits sometimes
Overfitting

- Huge feature space with kernels: should we worry about overfitting?
  - SVM objective seeks a solution with large margin
  - Good theoretical guarantee
  - But everything overfits sometimes
- Can control by:
  - Setting C
  - Choosing a better kernel
  - Varying parameters of the kernels

Dual for the non-separable case

Primal:
\[
\begin{align*}
\text{minimize}_{w,b} & \quad \frac{1}{2}w \cdot w + C \sum \xi_j \\
\text{subject to} & \quad y_j (w \cdot x_j + b) \geq 1 - \xi_j, \quad \forall j \\
& \quad \xi_j \geq 0, \quad \forall j
\end{align*}
\]

Dual:
\[
\begin{align*}
\sum \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \\
\sum \alpha_i y_i = 0 \\
C \geq \alpha_i \geq 0
\end{align*}
\]

What changed?
- Added upper bound of C on \( \alpha_i \)
- Intuitive explanation:
  - Without slack, \( \alpha_i \to \infty \) when constraints are violated (points misclassified)
  - Upper bound of C limits the \( \alpha_i \), so misclassifications are allowed
• Changing C
  – For clean data C doesn’t matter much.
  – For noisy data, large C leads to narrow margin
    (SVM tries to do a good job at separating, even though it isn’t possible)

• Noisy data
  – Clean data has few support vectors
  – Noisy data leads to data in the margins
  – More support vectors for noisy data
Insights

- Changing C
  - For clean data C doesn’t matter much.
  - For noisy data, large C leads to more complicated margin (SVM tries to do a good job at separating, even though it isn’t possible)
  - **Overfitting for large C**

- Noisy data
  - Clean data has few support vectors
  - Noisy data leads to data in the margins
  - More support vectors for noisy data

Common kernels for SVM

- Polynomials of degree exactly $d$
  \[ K(u, v) = (u \cdot v)^d \]

- Polynomials of degree up to $d$
  \[ K(u, v) = (u \cdot v + 1)^d \]

- Gaussian kernels
  \[ K(u, v) = \exp \left( -\frac{||u - v||^2}{2\sigma^2} \right) \]
  Aka Gaussian Radial basis function (RBF) kernel
Insights

- Changing $\sigma$
  - For clean data, $\sigma$ doesn’t matter much.
  - For noisy data, small $\sigma$ leads to more complicated margin (SVM tries to do a good job at separating, even though it isn’t possible)
  - Lots of overfitting for small $\sigma$
- Noisy data
  - Clean data has few support vectors
  - Noisy data leads to data in the margins
  - More support vectors for noisy data

Homework (part of assignment 2)

- Study the “Sequential Minimal Optimization” algorithm and implement an SVM classifier by yourself
- References
  - Fast Training of Support Vector Machines using Sequential Minimal Optimization
What we learned today

• Perceptron (and kernels)
• Support Vector Machines

Homework

• Read Murphy CH 14.1-14.2, 14.4-14.5.
• Assignment 1 is out. Due in two weeks.