Introduction to Machine Learning

Linear Classifiers

Lisbon Machine Learning School, 2014

Ryan McDonald

Google Inc., London
E-mail: ryanmcd@google.com
Linear Classifiers

▶ Go onto ACL Anthology
▶ Do the same on Google Scholar
  ▶ “Maximum Entropy” & “NLP” 9,000 hits, 240 before 2000
  ▶ “SVM” & “NLP” 11,000 hits, 556 before 2000
  ▶ “Perceptron” & “NLP”, 3,000 hits, 147 before 2000
▶ All are examples of linear classifiers
▶ All have become tools in any NLP/CL researchers tool-box in past 15 years
  ▶ Arguably the most important tool
Experiment

- Document 1 – label: 0; words: ★ ◊ ○
- Document 2 – label: 0; words: ★ ♥ △
- Document 3 – label: 1; words: ★ △ ♠
- Document 4 – label: 1; words: ◊ △ ○
### Experiment

- Document 1 – label: 0; words: ★ ◊ ○
- Document 2 – label: 0; words: ★ ♥ △
- Document 3 – label: 1; words: ★ △ ♠
- Document 4 – label: 1; words: ◊ △ ○
- New document – words: ★ ◊ ○; label ?
Experiment

- Document 1 – label: 0; words: ⭐ ⧧ ○
- Document 2 – label: 0; words: ⭐ ♥ △
- Document 3 – label: 1; words: ⭐ △ ♠
- Document 4 – label: 1; words: ⧧ △ ○
- New document – words: ⭐ ⧧ ○; label ?
- New document – words: ⭐ ⧧ ♥; label ?
Experiment

- Document 1 – label: 0; words: ⋆ ◊ ◦
- Document 2 – label: 0; words: ⋆ ♥ △
- Document 3 – label: 1; words: ⋆ △ ♠
- Document 4 – label: 1; words: ◆ △ ◦

- New document – words: ⋆ ◊ ◦; label ?
- New document – words: ⋆ ◊ ♥; label ?
- New document – words: ⋆ △ ◦; label ?
Experiment

- Document 1 – label: 0; words: ★ ◊ ○
- Document 2 – label: 0; words: ★ ♥ △
- Document 3 – label: 1; words: ★ △ ♠
- Document 4 – label: 1; words: ◊ △ ○

- New document – words: ★ ◊ ○; label ?
- New document – words: ★ ◊ ♥; label ?
- New document – words: ★ △ ○; label ?

Why can we do this?
Experiment

- Document 1 – label: 0; words: ★ ◊ ○
- Document 2 – label: 0; words: ★ ♠ △
- Document 3 – label: 1; words: ★ △ ♠
- Document 4 – label: 1; words: ◊ △ ○
- New document – words: ★ ◊ ♥; label 0

<table>
<thead>
<tr>
<th>Label 0</th>
<th>Label 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(0</td>
<td>★) = \frac{\text{count}(★ \text{ and } 0)}{\text{count}(★)} = \frac{2}{3} = 0.67 ) vs. ( P(1</td>
</tr>
<tr>
<td>( P(0</td>
<td>♥) = \frac{\text{count}(♥ \text{ and } 0)}{\text{count}(♥)} = \frac{1}{1} = 1.0 ) vs. ( P(1</td>
</tr>
</tbody>
</table>
Experiment

- **Document 1** – label: 0; words: ⭐ ◇ ●
- **Document 2** – label: 0; words: ⭐ ♦ ♤
- **Document 3** – label: 1; words: ⭐ ♤ ♠
- **Document 4** – label: 1; words: ◇ ♤ ●

- **New document** – words: ⭐ ♤ ●; label ?

<table>
<thead>
<tr>
<th>Label 0</th>
<th>Label 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(0</td>
<td>\ast) = \frac{\text{count}(\ast \text{ and } 0)}{\text{count}(\ast)} = \frac{2}{3} = 0.67 ) vs. ( P(1</td>
</tr>
<tr>
<td>( P(0</td>
<td>\bigtriangleup) = \frac{\text{count}(\bigtriangleup \text{ and } 0)}{\text{count}(\bigtriangleup)} = \frac{1}{3} = 0.33 ) vs. ( P(1</td>
</tr>
<tr>
<td>( P(0</td>
<td>\bigcirc) = \frac{\text{count}(\bigcirc \text{ and } 0)}{\text{count}(\bigcirc)} = \frac{1}{2} = 0.5 ) vs. ( P(1</td>
</tr>
</tbody>
</table>
Machine Learning

- Machine learning is well motivated counting
- Typically, machine learning models
  1. Define a model/distribution of interest
  2. Make some assumptions if needed
  3. Count!!

- Model: \( P(\text{label}|\text{doc}) = P(\text{label}|\text{word}_1, \ldots, \text{word}_n) \)
  - Prediction for new doc = \( \arg \max_{\text{label}} P(\text{label}|\text{doc}) \)
- Assumption: \( P(\text{label}|\text{word}_1, \ldots, \text{word}_n) = \frac{1}{n} \sum_i P(\text{label}|\text{word}_i) \)
- Count (as in example)
Lecture Outline

- Preliminaries
  - Data: input/output, assumptions
  - Feature representations
  - Linear classifiers and decision boundaries

- Classifiers
  - Naive Bayes
  - Generative versus discriminative
  - Logistic-regression
  - Perceptron
  - Large-Margin Classifiers (SVMs)

- Regularization

- Online learning

- Non-linear classifiers
Inputs and Outputs

- **Input:** \( x \in \mathcal{X} \)
  - e.g., document or sentence with some words \( x = w_1 \ldots w_n \), or a series of previous actions

- **Output:** \( y \in \mathcal{Y} \)
  - e.g., parse tree, document class, part-of-speech tags, word-sense

- **Input/Output pair:** \((x, y) \in \mathcal{X} \times \mathcal{Y}\)
  - e.g., a document \( x \) and its label \( y \)
  - Sometimes \( x \) is explicit in \( y \), e.g., a parse tree \( y \) will contain the sentence \( x \)
General Goal

When given a new input $x$ predict the correct output $y$

But we need to formulate this computationally!
Feature Representations

- We assume a mapping from input $x$ to a high dimensional feature vector
  - $\phi(x) : \mathcal{X} \rightarrow \mathbb{R}^m$
- For many cases, more convenient to have mapping from input-output pairs $(x, y)$
  - $\phi(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- Under certain assumptions, these are equivalent
- Most papers in NLP use $\phi(x, y)$
Feature Representations

- We assume a mapping from input $x$ to a high dimensional feature vector
  - $\phi(x) : \mathcal{X} \rightarrow \mathbb{R}^m$
- For many cases, more convenient to have mapping from input-output pairs $(x, y)$
  - $\phi(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- Under certain assumptions, these are equivalent
- Most papers in NLP use $\phi(x, y)$
- Not common in NLP: $\phi \in \mathbb{R}^m$
- More common: $\phi_i \in \{1, \ldots, F_i\}$, $F_i \in \mathbb{N}^+$ (categorical)
- Very common: $\phi \in \{0, 1\}^m$ (binary)
Feature Representations

- We assume a mapping from input $x$ to a high dimensional feature vector
  - $\phi(x) : \mathcal{X} \rightarrow \mathbb{R}^m$
- For many cases, more convenient to have mapping from input-output pairs $(x, y)$
  - $\phi(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- Under certain assumptions, these are equivalent
- Most papers in NLP use $\phi(x, y)$
- Not common in NLP: $\phi \in \mathbb{R}^m$
- More common: $\phi_i \in \{1, \ldots, F_i\}$, $F_i \in \mathbb{N}^+$ (categorical)
- Very common: $\phi \in \{0, 1\}^m$ (binary)
- For any vector $v \in \mathbb{R}^m$, let $v_j$ be the $j^{th}$ value
Examples

- $x$ is a document and $y$ is a label

$$
\phi_j(x, y) = \begin{cases} 
1 & \text{if } x \text{ contains the word "interest"} \\
& \text{and } y = "\text{financial}" \\
0 & \text{otherwise}
\end{cases}
$$

$$
\phi_j(x, y) = \% \text{ of words in } x \text{ with punctuation and } y = "\text{scientific}" 
$$

- $x$ is a word and $y$ is a part-of-speech tag

$$
\phi_j(x, y) = \begin{cases} 
1 & \text{if } x = "\text{bank}" \text{ and } y = \text{Verb} \\
0 & \text{otherwise}
\end{cases}
$$
Example 2

- $x$ is a name, $y$ is a label classifying the name

\[
\phi_0(x, y) = \begin{cases} 
1 & \text{if } x \text{ contains “George” and } y = “Person” \\
0 & \text{otherwise}
\end{cases}
\]

\[
\phi_1(x, y) = \begin{cases} 
1 & \text{if } x \text{ contains “Washington” and } y = “Person” \\
0 & \text{otherwise}
\end{cases}
\]

\[
\phi_2(x, y) = \begin{cases} 
1 & \text{if } x \text{ contains “Bridge” and } y = “Person” \\
0 & \text{otherwise}
\end{cases}
\]

\[
\phi_3(x, y) = \begin{cases} 
1 & \text{if } x \text{ contains “General” and } y = “Person” \\
0 & \text{otherwise}
\end{cases}
\]

\[
\phi_4(x, y) = \begin{cases} 
1 & \text{if } x \text{ contains “George” and } y = “Object” \\
0 & \text{otherwise}
\end{cases}
\]

\[
\phi_5(x, y) = \begin{cases} 
1 & \text{if } x \text{ contains “Washington” and } y = “Object” \\
0 & \text{otherwise}
\end{cases}
\]

\[
\phi_6(x, y) = \begin{cases} 
1 & \text{if } x \text{ contains “Bridge” and } y = “Object” \\
0 & \text{otherwise}
\end{cases}
\]

\[
\phi_7(x, y) = \begin{cases} 
1 & \text{if } x \text{ contains “General” and } y = “Object” \\
0 & \text{otherwise}
\end{cases}
\]

- $x=$ General George Washington, $y=$ Person $\rightarrow \phi(x, y) = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$

- $x=$ George Washington Bridge, $y=$ Object $\rightarrow \phi(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0]$

- $x=$ George Washington George, $y=$ Object $\rightarrow \phi(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0]$
Block Feature Vectors

▶ $x=$ General George Washington, $y=$ Person $\rightarrow \phi(x, y) = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$
▶ $x=$ General George Washington, $y=$ Object $\rightarrow \phi(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1]$
▶ $x=$ George Washington Bridge, $y=$ Object $\rightarrow \phi(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0]$
▶ $x=$ George Washington George, $y=$ Object $\rightarrow \phi(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0]$

▶ Each equal size block of the feature vector corresponds to one label
▶ Non-zero values allowed only in one block
Feature Representations - $\phi(x)$

- Instead of $\phi(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$ over input/outputs $(x, y)$

- Let $\phi(x) : \mathcal{X} \rightarrow \mathbb{R}^{m'}$ (e.g., $m' = m/|\mathcal{Y}|$)
  - i.e., Feature representation only over inputs $x$

- Equivalent when $\phi(x, y) = \phi(x) \times \mathcal{Y}$

- Advantages: Can make math cleaner, e.g., binary classification; Can use less parameters.

- Disadvantages: No complex features over properties of labels
Feature Representations - $\phi(x) \text{ vs. } \phi(x, y)$

- $\phi(x, y)$
  - $x=$ General George Washington, $y=$ Person $\rightarrow \phi(x, y) = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$
  - $x=$ General George Washington, $y=$ Object $\rightarrow \phi(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1]$

- $\phi(x)$
  - $x=$ General George Washington $\rightarrow \phi(x) = [1 \ 1 \ 0 \ 1]$

- Different ways of representing same thing
- Can deterministically map from $\phi(x)$ to $\phi(x, y)$ given $y$
Linear Classifiers

▶ **Linear classifier**: score (or probability) of a particular classification is based on a linear combination of features and their weights

▶ Let $\omega \in \mathbb{R}^m$ be a high dimensional weight vector

▶ Assume that $\omega$ is known

▶ **Multiclass Classification**: $\mathcal{Y} = \{0, 1, \ldots, N\}$

$$y = \arg\max_y \omega \cdot \phi(x, y)$$

$$= \arg\max_y \sum_{j=0}^{m} \omega_j \times \phi_j(x, y)$$

▶ **Binary Classification** just a special case of multiclass
Linear Classifiers – $\phi(x)$

- Define $|\mathcal{Y}|$ parameter vectors $\omega_y \in \mathbb{R}^{m'}$
  - i.e., one parameter vector per output class $y$

- Classification

$$y = \arg \max_y \omega_y \cdot \phi(x)$$
Linear Classifiers – \( \phi(x) \)

- Define \(|\mathcal{Y}|\) parameter vectors \( \omega_y \in \mathbb{R}^{m'} \)
  - I.e., one parameter vector per output class \( y \)

- Classification
  
  \[
  y = \arg \max_y \omega_y \cdot \phi(x)
  \]

- \( \phi(x, y) \)
  - \( x=\text{General George Washington}, \ y=\text{Person} \rightarrow \phi(x, y) = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0] \)
  - \( x=\text{General George Washington}, \ y=\text{Object} \rightarrow \phi(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1] \)
  - Single \( \omega \in \mathbb{R}^{8} \)

- \( \phi(x) \)
  - \( x=\text{General George Washington} \rightarrow \phi(x) = [1 \ 1 \ 0 \ 1] \)
  - Two parameter vectors \( \omega_0 \in \mathbb{R}^{4}, \ \omega_1 \in \mathbb{R}^{4} \)
Linear Classifiers - Bias Terms

- Often linear classifiers presented as

\[ y = \arg \max_y \sum_{j=0}^{m} \omega_j \times \phi_j(x, y) + b_y \]

- Where \( b \) is a bias or offset term
- Sometimes this is folded into \( \phi \)

\( x= \text{General George Washington}, \ y= \text{Person} \rightarrow \phi(x, y) = [1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0] \)

\( x= \text{General George Washington}, \ y= \text{Object} \rightarrow \phi(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1] \)

\( \phi_4(x, y) = \begin{cases} 
1 & \text{if } y = \text{“Person”} \\
0 & \text{otherwise}
\end{cases} \quad \phi_9(x, y) = \begin{cases} 
1 & \text{if } y = \text{“Object”} \\
0 & \text{otherwise}
\end{cases} \)

- \( \omega_4 \) and \( \omega_9 \) are now the bias terms for the labels
Binary Linear Classifier

Let’s say \( \omega = (1, -1) \) and \( b_y = 1, \forall y \)

Then \( \omega \) is a line (generally a hyperplane) that divides all points:
Binary Linear Classifier - Block Features

\[ \phi(x, y) = [v, 0] \text{ or } [0, v] \text{ in block features} \]
Multiclass Linear Classifier

Defines regions of space. Visualization difficult.

\[ \text{i.e., } + \text{ are all points } (x, y) \text{ where } + = \arg \max_y \omega \cdot \phi(x, y) \]
Separability

- A set of points is separable, if there exists a $\omega$ such that classification is perfect.

Separable

Not Separable

- This can also be defined mathematically (and we will shortly).
Machine Learning – finding $\omega$

- **Supervised Learning**
- Input: training examples $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}}$
- Input: feature representation $\phi$
- Output: $\omega$ that maximizes some important function on the training set
  - $\omega = \arg \max \mathcal{L} (\mathcal{T}; \omega)$
Machine Learning – finding $\omega$

- **Supervised Learning**
- Input: training examples $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}}$
- Input: feature representation $\phi$
- Output: $\omega$ that maximizes some *important function* on the training set
  - $\omega = \arg\max \mathcal{L}(\mathcal{T}; \omega)$
- Equivalently minimize: $\omega = \arg\min -\mathcal{L}(\mathcal{T}; \omega)$
Objective Functions

- $\mathcal{L}(\cdot)$ is called the **objective function**
- Usually we can decompose $\mathcal{L}$ by training pairs $(x, y)$
  - $\mathcal{L}(\mathcal{T}; \omega) \propto \sum_{(x,y) \in \mathcal{T}} \text{loss}((x, y); \omega)$
  - *loss* is a function that measures some value correlated with errors of parameters $\omega$ on instance $(x, y)$

- Defining $\mathcal{L}(\cdot)$ and *loss* is core of linear classifiers in machine learning
Supervised Learning – Assumptions

- Assumption: \((x_t, y_t)\) are sampled i.i.d.
  - i.i.d. = independent and identically distributed
  - independent = each sample independent of the other
  - identically = each sample from same probability distribution

- Sometimes assumption: The training data is separable
  - Needed to prove convergence for Perceptron
  - Not needed in practice
For a moment, forget linear classifiers and parameter vectors $\omega$.

Let’s assume our goal is to model the conditional probability of output labels $y$ given inputs $x$ (or $\phi(x)$).

I.e., $P(y|x)$.

If we can define this distribution, then classification becomes $\arg\max_y P(y|x)$. 
Bayes Rule

- One way to model $P(y|x)$ is through Bayes Rule:

$$P(y|x) = \frac{P(y)P(x|y)}{P(x)}$$

$$\arg\max_y P(y|x) \propto \arg\max_y P(y)P(x|y)$$

- Since $x$ is fixed

- $P(y)P(x|y) = P(x, y)$: a joint probability

- Modeling the joint input-output distribution is at the core of generative models
  - Because we model a distribution that can randomly generate outputs and inputs, not just outputs
  - More on this later
Naive Bayes (NB)

- Use $\phi(x) \in \mathbb{R}^m$ instead of $\phi(x, y)$

- $P(x|y) = P(\phi(x)|y) = P(\phi_1(x), \ldots, \phi_m(x)|y)$

**Naive Bayes Assumption**
*(conditional independence)*

$$P(\phi_1(x), \ldots, \phi_m(x)|y) = \prod_i P(\phi_i(x)|y)$$

$$P(y)P(\phi_1(x), \ldots, \phi_m(x)|y) = P(y)\prod_{i=1}^{m} P(\phi_i(x)|y)$$
Naive Bayes – Learning

- **Input:** $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}}$

- Let $\phi_i(x) \in \{1, \ldots, F_i\}$ – categorical; common in NLP

- **Parameters** $\mathcal{P} = \{P(y), P(\phi_i(x)|y)\}$
  - Both $P(y)$ and $P(\phi_i(x)|y)$ are multinomials

- **Objective:** Maximum Likelihood Estimation (MLE)

$$
\mathcal{L}(\mathcal{T}) = \prod_{t=1}^{\mathcal{T}} P(x_t, y_t) = \prod_{t=1}^{\mathcal{T}} \left( P(y_t) \prod_{i=1}^{m} P(\phi_i(x_t)|y_t) \right)
$$

$$
\mathcal{P} = \arg \max_{\mathcal{P}} \prod_{t=1}^{\mathcal{T}} \left( P(y_t) \prod_{i=1}^{m} P(\phi_i(x_t)|y_t) \right)
$$
Naive Bayes – Learning

MLE has closed form solution!! (more later)

\[ P = \arg \max_{\mathcal{P}} \prod_{t=1}^{\mid \mathcal{T} \mid} \left( P(y_t) \prod_{i=1}^{m} P(\phi_i(x_t)|y_t) \right) \]

\[ P(y) = \frac{\sum_{t=1}^{\mid \mathcal{T} \mid} [y_t = y]}{\mid \mathcal{T} \mid} \]

\[ P(\phi_i(x)|y) = \frac{\sum_{t=1}^{\mid \mathcal{T} \mid} [\phi_i(x_t) = \phi_i(x) \text{ and } y_t = y]}{\sum_{t=1}^{\mid \mathcal{T} \mid} [y_t = y]} \]

[[X]] is the identity function for property X

Thus, these are just normalized counts over events in \( \mathcal{T} \)
Naive Bayes Example

- $\phi_i(x) \in 0, 1, \forall i$
- doc 1: $y_1 = 0$, $\phi_0(x_1) = 1$, $\phi_1(x_1) = 1$
- doc 2: $y_2 = 0$, $\phi_0(x_2) = 0$, $\phi_1(x_2) = 1$
- doc 3: $y_3 = 1$, $\phi_0(x_3) = 1$, $\phi_1(x_3) = 0$

- Two label parameters $P(y = 0)$, $P(y = 1)$
- Eight feature parameters
  - 2 (labels) * 2 (features) * 2 (feature values)
  - E.g., $y = 0$ and $\phi_0(x) = 1$: $P(\phi_0(x) = 1|y = 0)$

- $P(y = 0) = 2/3$, $P(y = 1) = 1/3$
- $P(\phi_0(x) = 1|y = 0) = 1/2$, $P(\phi_1(x) = 0|y = 1) = 1/1$
Naive Bayes Document Classification

- doc 1: $y_1 = \text{sports, “hockey is fast”}$
- doc 2: $y_2 = \text{politics, “politicians talk fast”}$
- doc 3: $y_3 = \text{politics, “washington is sleazy”}$

- $\phi_0(x) = 1$ iff doc has word ‘hockey’, 0 o.w.
- $\phi_1(x) = 1$ iff doc has word ‘is’, 0 o.w.
- $\phi_2(x) = 1$ iff doc has word ‘fast’, 0 o.w.
- $\phi_3(x) = 1$ iff doc has word ‘politicians’, 0 o.w.
- $\phi_4(x) = 1$ iff doc has word ‘talk’, 0 o.w.
- $\phi_5(x) = 1$ iff doc has word ‘washington’, 0 o.w.
- $\phi_6(x) = 1$ iff doc has word ‘sleazy’, 0 o.w.
Deriving MLE

\[ \mathcal{P} = \arg \max_{\mathcal{P}} \prod_{t=1}^{\mid \mathcal{T} \mid} \left( P(y_t) \prod_{i=1}^{m} P(\phi_i(x_t)|y_t) \right) \]

\[ = \arg \max_{\mathcal{P}} \sum_{t=1}^{\mid \mathcal{T} \mid} \left( \log P(y_t) + \sum_{i=1}^{m} \log P(\phi_i(x_t)|y_t) \right) \]

\[ = \arg \max_{P(y)} \sum_{t=1}^{\mid \mathcal{T} \mid} \log P(y_t) + \arg \max_{P(\phi_i(x)|y)} \sum_{t=1}^{\mid \mathcal{T} \mid} \sum_{i=1}^{m} \log P(\phi_i(x_t)|y_t) \]

such that \( \sum_{y} P(y) = 1, \sum_{j=1}^{F_i} P(\phi_i(x) = j|y) = 1, P(\cdot) \geq 0 \)
Deriving MLE

\[ P = \arg \max_{P(y)} \sum_{t=1}^{|T|} \log P(y_t) + \arg \max_{P(\phi_i(x)|y)} \sum_{t=1}^{|T|} \sum_{i=1}^{m} \log P(\phi_i(x_t)|y_t) \]

Both optimizations are of the form

\[ \arg \max_{P} \sum_{v} \text{count}(v) \log P(v), \text{s.t.}, \sum_{v} P(v) = 1, P(v) \geq 0 \]

For example:

\[ \arg \max_{P(y)} \sum_{t=1}^{|T|} \log P(y_t) = \arg \max_{P(y)} \sum_{y} \text{count}(y, T) \log P(y) \]

such that \( \sum_{y} P(y) = 1, P(y) \geq 0 \)
Deriving MLE

\[
\arg \max_P \sum_v \text{count}(v) \log P(v)
\]
\[
s.t., \sum_v P(v) = 1, P(v) \geq 0
\]

Introduce Lagrangian multiplier \( \lambda \), optimization becomes

\[
\arg \max_{P, \lambda} \sum_v \text{count}(v) \log P(v) - \lambda (\sum_v P(v) - 1)
\]

Derivative w.r.t. \( P(v) \) is

\[
\frac{\text{count}(v)}{P(v)} - \lambda
\]

Setting this to zero \( P(v) = \frac{\text{count}(v)}{\lambda} \)

Combine with \( \sum_v P(v) = 1. P(v) \geq 0 \), then \( P(v) = \frac{\text{count}(v)}{\sum_v \text{count}(v')} \)
Put it together

\[ \mathcal{P} = \arg \max_{\mathcal{P}} \prod_{t=1}^{|\mathcal{T}|} \left( P(y_t) \prod_{i=1}^m P(\phi_i(x_t)|y_t) \right) \]

\[ = \arg \max_{P(y)} \sum_{t=1}^{|\mathcal{T}|} \log P(y_t) + \arg \max_{P(\phi_i(x)|y)} \sum_{t=1}^{|\mathcal{T}|} \sum_{i=1}^m \log P(\phi_i(x_t)|y_t) \]

\[ P(y) = \frac{\sum_{t=1}^{|\mathcal{T}|} [y_t = y]}{|\mathcal{T}|} \]

\[ P(\phi_i(x)|y) = \frac{\sum_{t=1}^{|\mathcal{T}|} [\phi_i(x_t) = \phi_i(x) \text{ and } y_t = y]}{\sum_{t=1}^{|\mathcal{T}|} [y_t = y]} \]
NB is a linear classifier

- Let $\omega_y = \log P(y), \forall y \in \mathcal{Y}$
- Let $\omega_{\phi_i(x), y} = \log P(\phi_i(x) | y), \forall y \in \mathcal{Y}, \phi_i(x) \in \{1, \ldots, F_i\}$
- Let $\omega$ be set of all $\omega_*$ and $\omega_*,*$

$$
\arg \max_y P(y | \phi(x)) \propto \arg \max_y P(\phi(x), y) = \arg \max_y P(y) \prod_{i=1}^m P(\phi_i(x) | y) \\
= \arg \max_y \log P(y) + \sum_{i=1}^m \log P(\phi_i(x) | y) \\
= \arg \max_y \omega_y + \sum_{i=1}^m \omega_{\phi_i(x), y} \\
= \arg \max_y \sum_{y'} \omega_y \psi_{y'}(y) + \sum_{i=1}^m \sum_{j=1}^{F_i} \omega_{\phi_i(x), y} \psi_{i,j}(x)
$$

where $\psi_* \in \{0, 1\}$, $\psi_{i,j}(x) = [[\phi_i(x) = j]]$, $\psi_{y'}(y) = [[y = y']]$
Smoothing

- doc 1: $y_1 = \text{sports, “hockey is fast”}$
- doc 2: $y_2 = \text{politics, “politicians talk fast”}$
- doc 3: $y_3 = \text{politics, “washington is sleazy”}$

- New doc: “washington hockey is fast”
- Both ‘sports’ and ‘politics’ have probabilities of 0

- Smoothing aims to assign a small amount of probability to unseen events

- E.g., Additive/Laplacian smoothing

$$P(v) = \frac{\text{count}(v)}{\sum_{v'} \text{count}(v')} \implies P(v) = \frac{\text{count}(v) + \alpha}{\sum_{v'} (\text{count}(v') + \alpha)}$$
Discriminative versus Generative

- Generative models attempt to model inputs and outputs
  - e.g., NB = MLE of joint distribution $P(x, y)$
  - Statistical model must explain generation of input

- Ocam’s Razor: why model input?

- Discriminative models
  - Use $\mathcal{L}$ that directly optimizes $P(y|x)$ (or something related)
  - Logistic Regression – MLE of $P(y|x)$
  - Perceptron and SVMs – minimize classification error

- Generative and discriminative models use $P(y|x)$ for prediction
- Differ only on what distribution they use to set $\omega$
Logistic Regression
Logistic Regression

Define a conditional probability:

\[ P(y|x) = \frac{e^{\omega \cdot \phi(x,y)}}{Z_x}, \quad \text{where } Z_x = \sum_{y' \in Y} e^{\omega \cdot \phi(x,y')} \]

Note: still a linear classifier

\[
\begin{align*}
\arg \max_y P(y|x) &= \arg \max_y \frac{e^{\omega \cdot \phi(x,y)}}{Z_x} \\
&= \arg \max_y e^{\omega \cdot \phi(x,y)} \\
&= \arg \max_y \omega \cdot \phi(x, y)
\end{align*}
\]
Logistic Regression

\[ P(y|x) = \frac{e^{\omega \cdot \phi(x,y)}}{Z_x} \]

- Q: How do we learn weights \( \omega \)?
- A: Set weights to maximize log-likelihood of training data:

\[
\omega = \arg \max_\omega \mathcal{L}(\mathcal{T}; \omega) = \arg \max_\omega \left| \mathcal{T} \right| \prod_{t=1}^{\left| \mathcal{T} \right|} P(y_t|x_t) = \arg \max_\omega \sum_{t=1}^{\left| \mathcal{T} \right|} \log P(y_t|x_t)
\]

- In a nut shell we set the weights \( \omega \) so that we assign as much probability to the correct label \( y \) for each \( x \) in the training set.
Logistic Regression

\[ P(y|x) = \frac{e^{\omega \cdot \phi(x,y)}}{Z_x}, \quad \text{where } Z_x = \sum_{y' \in \mathcal{Y}} e^{\omega \cdot \phi(x,y')} \]

\[ \omega = \arg \max_{\omega} \sum_{t=1}^{|T|} \log P(y_t|x_t) \quad (*) \]

- The objective function (*) is concave (take the 2nd derivative)
- Therefore there is a global maximum
- No closed form solution, but lots of numerical techniques
  - Gradient methods (gradient ascent, conjugate gradient, iterative scaling)
  - Newton methods (limited-memory quasi-newton)
Gradient Ascent

- Let \( \mathcal{L}(\mathcal{T}; \omega) = \sum_{t=1}^{\mid\mathcal{T}\mid} \log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_x} \right) \)
- Want to find \( \arg \max_{\omega} \mathcal{L}(\mathcal{T}; \omega) \)
  - Set \( \omega^0 = O^m \)
  - Iterate until convergence
    \[
    \omega^i = \omega^{i-1} + \alpha \nabla \mathcal{L}(\mathcal{T}; \omega^{i-1})
    \]
- \( \alpha > 0 \) and set so that \( \mathcal{L}(\mathcal{T}; \omega^i) > \mathcal{L}(\mathcal{T}; \omega^{i-1}) \)
- \( \nabla \mathcal{L}(\mathcal{T}; \omega) \) is gradient of \( \mathcal{L} \) w.r.t. \( \omega \)
  - A gradient is all partial derivatives over variables \( \omega_i \)
  - i.e., \( \nabla \mathcal{L}(\mathcal{T}; \omega) = \left( \frac{\partial}{\partial \omega_0} \mathcal{L}(\mathcal{T}; \omega), \frac{\partial}{\partial \omega_1} \mathcal{L}(\mathcal{T}; \omega), \ldots, \frac{\partial}{\partial \omega_m} \mathcal{L}(\mathcal{T}; \omega) \right) \)
- Gradient ascent will always find \( \omega \) to maximize \( \mathcal{L} \)
Gradient Descent

- Let $L(\mathcal{T}; \omega) = -\sum_{t=1}^{\lvert \mathcal{T} \rvert} \log \left( e^{\omega \cdot \phi(x_t, y_t)} / Z_x \right)$
- Want to find $\arg \min \omega L(\mathcal{T}; \omega)$
  - Set $\omega^0 = O^m$
  - Iterate until convergence

$$\omega^i = \omega^{i-1} - \alpha \nabla L(\mathcal{T}; \omega^{i-1})$$

- $\alpha > 0$ and set so that $L(\mathcal{T}; \omega^i) < L(\mathcal{T}; \omega^{i-1})$
- $\nabla L(\mathcal{T}; \omega)$ is gradient of $L$ w.r.t. $\omega$
  - A gradient is all partial derivatives over variables $\omega_i$
  - i.e., $\nabla L(\mathcal{T}; \omega) = (\frac{\partial}{\partial \omega_0} L(\mathcal{T}; \omega), \frac{\partial}{\partial \omega_1} L(\mathcal{T}; \omega), \ldots, \frac{\partial}{\partial \omega_m} L(\mathcal{T}; \omega))$

- Gradient ascent will always find $\omega$ to minimize $L$
The partial derivatives

- Need to find all partial derivatives \( \frac{\partial}{\partial \omega_i} \mathcal{L}(\mathcal{T}; \omega) \)

\[
\mathcal{L}(\mathcal{T}; \omega) = \sum_t \log P(y_t | x_t)
\]

\[
= \sum_t \log \frac{e^{\omega \cdot \phi(x_t, y_t)}}{\sum_{y' \in y} e^{\omega \cdot \phi(x_t, y')}}
\]

\[
= \sum_t \log \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}}
\]
Partial derivatives - some reminders

1. \[ \frac{\partial}{\partial x} \log F = \frac{1}{F} \frac{\partial}{\partial x} F \]
   - We always assume \( \log \) is the natural logarithm \( \log_e \)

2. \[ \frac{\partial}{\partial x} e^F = e^F \frac{\partial}{\partial x} F \]

3. \[ \frac{\partial}{\partial x} \sum_t F_t = \sum_t \frac{\partial}{\partial x} F_t \]

4. \[ \frac{\partial}{\partial x} \frac{F}{G} = \frac{G \frac{\partial}{\partial x} F - F \frac{\partial}{\partial x} G}{G^2} \]
The partial derivatives

\[
\frac{\partial}{\partial \omega_i} \mathcal{L}(T; \omega) = \frac{\partial}{\partial \omega_i} \sum_t \log \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}}
\]

\[
= \sum_t \frac{\partial}{\partial \omega_i} \log \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}}
\]

\[
= \sum_t \left( \frac{Z_{x_t}}{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}} \right) \left( \frac{\partial}{\partial \omega_i} \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}} \right)
\]
The partial derivatives

Now,

\[
\frac{\partial}{\partial \omega_i} \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}} = \frac{Z_{x_t} \frac{\partial}{\partial \omega_i} e^{\sum_j \omega_j \times \phi_j(x_t, y_t)} - e^{\sum_j \omega_j \times \phi_j(x_t, y_t)} \frac{\partial}{\partial \omega_i} Z_{x_t}}{Z_{x_t}^2}
\]

\[
= \frac{Z_{x_t} e^{\sum_j \omega_j \times \phi_j(x_t, y_t)} \phi_i(x_t, y_t) - e^{\sum_j \omega_j \times \phi_j(x_t, y_t)} \frac{\partial}{\partial \omega_i} Z_{x_t}}{Z_{x_t}^2}
\]

\[
= \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}^2} (Z_{x_t} \phi_i(x_t, y_t) - \frac{\partial}{\partial \omega_i} Z_{x_t})
\]

\[
= \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}^2} (Z_{x_t} \phi_i(x_t, y_t) - \sum_{y' \in Y} e^{\sum_j \omega_j \times \phi_j(x_t, y')} \phi_i(x_t, y'))
\]

because

\[
\frac{\partial}{\partial \omega_i} Z_{x_t} = \frac{\partial}{\partial \omega_i} \sum_{y' \in Y} e^{\sum_j \omega_j \times \phi_j(x_t, y')} = \sum_{y' \in Y} e^{\sum_j \omega_j \times \phi_j(x_t, y')} \phi_i(x_t, y')
\]
The partial derivatives

From before,
\[
\frac{\partial}{\partial \omega_i} \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}} = \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}^2} \left( Z_{x_t} \phi_i(x_t, y_t) - \sum_{y' \in \mathcal{Y}} e^{\sum_j \omega_j \times \phi_j(x_t, y')} \phi_i(x_t, y') \right)
\]

Sub this in,
\[
\frac{\partial}{\partial \omega_i} \mathcal{L}(T; \omega) = \sum_t \left( \frac{Z_{x_t}}{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}} \right) \left( \frac{\partial}{\partial \omega_i} \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}} \right)
\]
\[
= \sum_t \frac{1}{Z_{x_t}} \left( Z_{x_t} \phi_i(x_t, y_t) - \sum_{y' \in \mathcal{Y}} e^{\sum_j \omega_j \times \phi_j(x_t, y')} \phi_i(x_t, y') \right)
\]
\[
= \sum_t \phi_i(x_t, y_t) - \sum_t \sum_{y' \in \mathcal{Y}} \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y')}}{Z_{x_t}} \phi_i(x_t, y')
\]
\[
= \sum_t \phi_i(x_t, y_t) - \sum_t \sum_{y' \in \mathcal{Y}} P(y' | x_t) \phi_i(x_t, y')
\]
FINALLY!!!

- After all that,

\[
\frac{\partial}{\partial \omega_i} \mathcal{L}(\mathcal{T}; \omega) = \sum_t \phi_i(x_t, y_t) - \sum_t \sum_{y' \in \mathcal{Y}} P(y'|x_t) \phi_i(x_t, y')
\]

- And the gradient is:

\[
\nabla \mathcal{L}(\mathcal{T}; \omega) = \left( \frac{\partial}{\partial \omega_0} \mathcal{L}(\mathcal{T}; \omega), \frac{\partial}{\partial \omega_1} \mathcal{L}(\mathcal{T}; \omega), \ldots, \frac{\partial}{\partial \omega_m} \mathcal{L}(\mathcal{T}; \omega) \right)
\]

- So we can now use gradient ascent to find \( \omega!! \)
Logistic Regression Summary

- Define conditional probability

\[ P(y\mid x) = \frac{e^{\mathbf{\omega} \cdot \phi(x, y)}}{Z_x} \]

- Set weights to maximize log-likelihood of training data:

\[ \mathbf{\omega} = \arg \max_\mathbf{\omega} \sum_t \log P(y_t \mid x_t) \]

- Can find the gradient and run gradient ascent (or any gradient-based optimization algorithm)

\[ \frac{\partial}{\partial \omega_i} L(T; \mathbf{\omega}) = \sum_t \phi_i(x_t, y_t) - \sum_t \sum_{y' \in Y} P(y' \mid x_t) \phi_i(x_t, y') \]
Logistic Regression = Maximum Entropy

- Well known equivalence
- Max Ent: maximize entropy subject to constraints on features
  - Empirical feature counts must equal expected counts
- Quick intuition
  - Partial derivative in logistic regression
  
  \[
  \frac{\partial}{\partial \omega_i} \mathcal{L}(T; \omega) = \sum_t \phi_i(x_t, y_t) - \sum_t \sum_{y' \in \mathcal{Y}} P(y'|x_t) \phi_i(x_t, y')
  \]
  
  - First term is empirical feature counts and second term is expected counts
  - Derivative set to zero maximizes function
  - Therefore when both counts are equivalent, we optimize the logistic regression objective!
Perceptron
Perceptron

Choose a $\omega$ that minimizes error

$$\mathcal{L}(\mathcal{T}; \omega) = \sum_{t=1}^{\mid \mathcal{T} \mid} 1 - [y_t = \arg \max_y \omega \cdot \phi(x_t, y)]$$

$$\omega = \arg \min \sum_{t=1}^{\mid \mathcal{T} \mid} 1 - [y_t = \arg \max_y \omega \cdot \phi(x_t, y)]$$

$$[[p]] = \begin{cases} 
1 & p \text{ is true} \\
0 & \text{otherwise}
\end{cases}$$

This is a 0-1 loss function

- When minimizing error people tend to use hinge-loss
- We'll get back to this
Aside: Min error versus max log-likelihood

- Highly related but not identical

- Example: consider a training set $\mathcal{T}$ with 1001 points
  
  $1000 \times (x_i, y = 0) = [-1, 1, 0, 0]$ for $i = 1 \ldots 1000$
  
  $1 \times (x_{1001}, y = 1) = [0, 0, 3, 1]$

- Now consider $\omega = [-1, 0, 1, 0]$

- Error in this case is 0 – so $\omega$ minimizes error
  
  $[-1, 0, 1, 0] \cdot [-1, 1, 0, 0] = 1 > [-1, 0, 1, 0] \cdot [0, 0, -1, 1] = -1$
  
  $[-1, 0, 1, 0] \cdot [0, 0, 3, 1] = 3 > [-1, 0, 1, 0] \cdot [3, 1, 0, 0] = -3$

- However, log-likelihood = -126.9 (omit calculation)
Aside: Min error versus max log-likelihood

- Highly related but not identical

- Example: consider a training set $\mathcal{T}$ with 1001 points

  $1000 \times (x_i, y = 0) = [-1, 1, 0, 0] \text{ for } i = 1 \ldots 1000$

  $1 \times (x_{1001}, y = 1) = [0, 0, 3, 1]$

- Now consider $\omega = [-1, 7, 1, 0]$

- Error in this case is 1 – so $\omega$ does not minimizes error

  $[-1, 7, 1, 0] \cdot [-1, 1, 0, 0] = 8 > [-1, 7, 1, 0] \cdot [-1, 1, 0, 0] = -1$

  $[-1, 7, 1, 0] \cdot [0, 0, 3, 1] = 3 < [-1, 7, 1, 0] \cdot [3, 1, 0, 0] = 4$

- However, log-likelihood = -1.4

- Better log-likelihood and worse error
Aside: Min error versus max log-likelihood

- Max likelihood $\neq$ min error
- Max likelihood pushes as much probability on correct labeling of training instance
  - Even at the cost of mislabeling a few examples
- Min error forces all training instances to be correctly classified
  - Often not possible
  - Ways of regularizing model to allow sacrificing some errors for better predictions on more examples
Perceptron Learning Algorithm

Training data: $T = \{(x_t, y_t)\}_{t=1}^{T}$

1. $\omega^{(0)} = 0; \ i = 0$
2. for $n: 1..N$
3. for $t: 1..T$
4. Let $y' = \arg\max_{y'} \omega^{(i)} \cdot \phi(x_t, y')$
5. if $y' \neq y_t$
6. $\omega^{(i+1)} = \omega^{(i)} + \phi(x_t, y_t) - \phi(x_t, y')$
7. $i = i + 1$
8. return $\omega^i$
Perceptron: Separability and Margin

Given an training instance \((x_t, y_t)\), define:

\[ \tilde{Y}_t = \mathcal{Y} - \{y_t\} \]

\[ \text{i.e., } \tilde{Y}_t \text{ is the set of incorrect labels for } x_t \]

A training set \(\mathcal{T}\) is separable with margin \(\gamma > 0\) if there exists a vector \(u\) with \(\|u\| = 1\) such that:

\[ u \cdot \phi(x_t, y_t) - u \cdot \phi(x_t, y') \geq \gamma \]

for all \(y' \in \tilde{Y}_t\) and \(\|u\| = \sqrt{\sum_j u_j^2}\)

**Assumption:** the training set is separable with margin \(\gamma\)
**Theorem**: For any training set separable with a margin of $\gamma$, the following holds for the perceptron algorithm:

\[
\text{mistakes made during training} \leq \frac{R^2}{\gamma^2}
\]

where $R \geq \|\phi(x_t, y_t) - \phi(x_t, y')\|$ for all $(x_t, y_t) \in T$ and $y' \in \bar{Y}_t$

Thus, after a finite number of training iterations, the error on the training set will converge to zero

**Let’s prove it!** (proof taken from Collins ’02)
Perceptron Learning Algorithm

Training data: \( \mathcal{T} = \{(x_t, y_t)\}_{t=1}^{|\mathcal{T}|} \)

1. \( \omega^{(0)} = 0; \ i = 0 \)
2. for \( n : 1..N \)
3. for \( t : 1..T \)
4. Let \( y' = \arg\max_{y'} \omega^{(i)} \cdot \phi(x_t, y') \)
5. if \( y' \neq y_t \)
6. \( \omega^{(i+1)} = \omega^{(i)} + \phi(x_t, y_t) - \phi(x_t, y') \)
7. \( i = i + 1 \)
8. return \( \omega^{i} \)

\( \omega^{(k-1)} \) are the weights before \( k^{th} \) mistake

Suppose \( k^{th} \) mistake made at the \( t^{th} \) example, \( (x_t, y_t) \)

\( y' = \arg\max_{y'} \omega^{(k-1)} \cdot \phi(x_t, y') \)

\( y' \neq y_t \)

\( \omega^{(k)} = \omega^{(k-1)} + \phi(x_t, y_t) - \phi(x_t, y') \)

Now:

\[ u \cdot \omega^{(k)} = u \cdot \omega^{(k-1)} + u \cdot (\phi(x_t, y_t) - \phi(x_t, y')) \geq u \cdot \omega^{(k-1)} + \gamma \]

Now: \( \omega^{(0)} = 0 \) and \( u \cdot \omega^{(0)} = 0 \), by induction on \( k \), \( u \cdot \omega^{(k)} \geq k \gamma \)

Now: since \( u \cdot \omega^{(k)} \leq ||u|| \times ||\omega^{(k)}|| \) and \( ||u|| = 1 \) then \( ||\omega^{(k)}|| \geq k \gamma \)

Now:

\[
||\omega^{(k)}||^2 = ||\omega^{(k-1)}||^2 + ||\phi(x_t, y_t) - \phi(x_t, y')||^2 + 2\omega^{(k-1)} \cdot (\phi(x_t, y_t) - \phi(x_t, y'))
\]

\[
||\omega^{(k)}||^2 \leq ||\omega^{(k-1)}||^2 + R^2
\]

(since \( R \geq ||\phi(x_t, y_t) - \phi(x_t, y')|| \)
and \( \omega^{(k-1)} \cdot \phi(x_t, y_t) - \omega^{(k-1)} \cdot \phi(x_t, y') \leq 0 \)
Perceptron Learning Algorithm

- We have just shown that $||\omega^{(k)}|| \geq k\gamma$ and $||\omega^{(k)}||^2 \leq ||\omega^{(k-1)}||^2 + R^2$

- By induction on $k$ and since $\omega^{(0)} = 0$ and $||\omega^{(0)}||^2 = 0$

\[
||\omega^{(k)}||^2 \leq kR^2
\]

- Therefore,

\[
k^2\gamma^2 \leq ||\omega^{(k)}||^2 \leq kR^2
\]

- and solving for $k$

\[
k \leq \frac{R^2}{\gamma^2}
\]

- Therefore the number of errors is bounded!
Perceptron Summary

- Learns a linear classifier that minimizes error
- Guaranteed to find a $\omega$ in a finite amount of time
- Perceptron is an example of an Online Learning Algorithm
  - $\omega$ is updated based on a single training instance in isolation

$$\omega^{(i+1)} = \omega^{(i)} + \phi(x_t, y_t) - \phi(x_t, y')$$
Averaged Perceptron

Training data: \( \mathcal{T} = \{(x_t, y_t)\}^{|\mathcal{T}|}_{t=1} \)

1. \( \omega^{(0)} = 0; \ i = 0 \)
2. for \( n : 1..N \)
3. for \( t : 1..T \)
4. Let \( y' = \arg \max_{y'} \omega^{(i)} \cdot \phi(x_t, y') \)
5. if \( y' \neq y_t \)
6. \( \omega^{(i+1)} = \omega^{(i)} + \phi(x_t, y_t) - \phi(x_t, y') \)
7. else
6. \( \omega^{(i+1)} = \omega^{(i)} \)
7. \( i = i + 1 \)
8. return \( \frac{\sum_i \omega^{(i)}}{N \times T} \)
Margin

Training

Testing

Denote the value of the margin by $\gamma$
Maximizing Margin

- For a training set $\mathcal{T}$
- Margin of a weight vector $\omega$ is smallest $\gamma$ such that
  \[ \omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq \gamma \]
- for every training instance $(x_t, y_t) \in \mathcal{T}$, $y' \in \bar{Y}_t$
Maximizing Margin

- Intuitively maximizing margin makes sense.
- More importantly, generalization error to unseen test data is proportional to the inverse of the margin:
  \[ \epsilon \propto \frac{R^2}{\gamma^2 \times |T|} \]

- **Perceptron**: we have shown that:
  - If a training set is separable by some margin, the perceptron will find a \( \omega \) that separates the data.
  - However, the perceptron does not pick \( \omega \) to maximize the margin!
Support Vector Machines (SVMs)
Maximizing Margin

Let $\gamma > 0$

$$\max_{||\omega|| \leq 1} \gamma$$

such that:

$$\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq \gamma$$

$$\forall (x_t, y_t) \in \mathcal{T}$$

and $y' \in \mathcal{Y}_t$

- Note: algorithm still minimizes error if data is separable
- $||\omega||$ is bound since scaling trivially produces larger margin

$$\beta(\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y')) \geq \beta \gamma,$$ for some $\beta \geq 1$
**Max Margin = Min Norm**

Let $\gamma > 0$

Max Margin:

$$\max_{\|\omega\| \leq 1} \gamma$$

such that:

$$\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq \gamma$$

$\forall (x_t, y_t) \in \mathcal{T}$

and $y' \in \check{Y}_t$

Min Norm:

$$\min_{\omega} \frac{1}{2}\|\omega\|^2$$

such that:

$$\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq 1$$

$\forall (x_t, y_t) \in \mathcal{T}$

and $y' \in \check{Y}_t$

▶ Instead of fixing $\|\omega\|$ we fix the margin $\gamma = 1$
Max Margin = Min Norm

Max Margin:

\[
\max_{||\omega||\leq 1} \gamma
\]

such that:

\[
\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq \gamma
\]

\forall (x_t, y_t) \in \mathcal{T}

and \( y' \in \mathcal{Y}_t \)

Min Norm:

\[
\min_{\omega} \frac{1}{2} ||\omega||^2
\]

such that:

\[
\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq 1
\]

\forall (x_t, y_t) \in \mathcal{T}

and \( y' \in \mathcal{Y}_t \)

- Let's say min norm solution \( ||\omega|| = \zeta \)
- Now say original objective is \( \max ||\omega|| \leq \zeta \gamma \)
- We know that \( \gamma \) must be 1
  - Or we would have found smaller \( ||\omega|| \) in min norm solution
- \( ||\omega|| \leq 1 \) in max margin formulation is an arbitrary scaling choice
Support Vector Machines

\[ \omega = \arg \min_{\omega} \frac{1}{2} \| \omega \|^2 \]

such that:

\[ \omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq 1 \]

\[ \forall (x_t, y_t) \in T \text{ and } y' \in \bar{Y}_t \]

- Quadratic programming problem – a well known convex optimization problem
- Can be solved with many techniques [Nocedal and Wright 1999]
Support Vector Machines

What if data is not separable?

\[
\omega = \arg \min_{\omega, \xi} \frac{1}{2} ||\omega||^2 + C \sum_{t=1}^{|T|} \xi_t
\]

such that:

\[
\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq 1 - \xi_t \quad \text{and} \quad \xi_t \geq 0
\]

\forall (x_t, y_t) \in T \quad \text{and} \quad y' \in \bar{Y}_t

\xi_t: \text{ trade-off between margin per example and } ||\omega||

Larger C = more examples correctly classified

If data is separable, optimal solution has \( \xi_i = 0, \forall i \)
Support Vector Machines

\[ \omega = \arg \min_{\omega,\xi} \frac{1}{2} \| \omega \|^2 + C \sum_{t=1}^{\tau} \xi_t \]

such that:

\[ \omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq 1 - \xi_t \]
Support Vector Machines

$$\omega = \arg \min_{\omega, \xi} \frac{1}{2}||\omega||^2 + C \sum_{t=1}^{|T|} \xi_t$$

such that:

$$\omega \cdot \phi(x_t, y_t) - \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') \geq 1 - \xi_t$$
Support Vector Machines

\[ \omega = \arg \min_{\omega, \xi} \frac{1}{2} \| \omega \|^2 + C \sum_{t=1}^{|T|} \xi_t \]

such that:

\[ \xi_t \geq 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t) \]
Support Vector Machines

\[ \omega = \arg\min_{\omega, \xi} \frac{\lambda}{2} ||\omega||^2 + \sum_{t=1}^{T} \xi_t \quad \lambda = \frac{1}{C} \]

such that:

\[ \xi_t \geq 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t) \]
Support Vector Machines

\[ \omega = \arg \min_{\omega, \xi} \frac{\lambda}{2} \|\omega\|^2 + \sum_{t=1}^{\mathcal{T}} \xi_t \quad \lambda = \frac{1}{C} \]

such that:

\[ \xi_t \geq 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t) \]

If \( \|\omega\| \) classifies \((x_t, y_t)\) with margin 1, penalty \( \xi_t = 0 \)

Otherwise penalty \( \xi_t = 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t) \)
Support Vector Machines

\[ \omega = \arg \min_{\omega, \xi} \frac{\lambda}{2} \| \omega \|^2 + \sum_{t=1}^{T} \xi_t \quad \lambda = \frac{1}{C} \]

such that:

\[ \xi_t \geq 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t) \]

If \( \| \omega \| \) classifies \((x_t, y_t)\) with margin 1, penalty \( \xi_t = 0 \)
Otherwise penalty \( \xi_t = 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t) \)

Hinge loss:

\[ \text{loss}((x_t, y_t); \omega) = \max (0, 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t)) \]
Support Vector Machines

\[ \omega = \arg \min_{\omega, \xi} \frac{\lambda}{2} ||\omega||^2 + \sum_{t=1}^{\mathcal{T}} \xi_t \]

such that:

\[ \xi_t \geq 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t) \]

Hinge loss equivalent

\[ \omega = \arg \min_{\omega} \mathcal{L}(\mathcal{T}; \omega) = \arg \min_{\omega} \sum_{t=1}^{\mathcal{T}} \text{loss}((x_t, y_t); \omega) + \frac{\lambda}{2} ||\omega||^2 \]

\[ = \arg \min_{\omega} \left( \sum_{t=1}^{\mathcal{T}} \max_{y' \neq y_t} (0, 1 + \max \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t)) \right) + \frac{\lambda}{2} ||\omega||^2 \]
Summary

What we have covered

▶ Linear Classifiers
  ▶ Naive Bayes
  ▶ Logistic Regression
  ▶ Perceptron
  ▶ Support Vector Machines

What is next

▶ Regularization
▶ Online learning
▶ Non-linear classifiers
Regularization
Overfitting

- Early in lecture we made assumption data was i.i.d.
- Rarely is this true
  - E.g., syntactic analyzers typically trained on 40,000 sentences from early 1990s WSJ news text

- Even more common: \( \mathcal{T} \) is very small
- This leads to **overfitting**

- E.g.: ‘fake’ is never a verb in WSJ treebank (only adjective)
  - High weight on ‘\( \phi(x, y) = 1 \) if \( x=\text{fake} \) and \( y=\text{adjective} \)’
  - Of course: leads to high log-likelihood / low error

- Other features might be more indicative
- Adjacent word identities: ‘He wants to X his death’ \( \rightarrow X=\text{verb} \)
Regularization

- In practice, we regularize models to prevent overfitting
  \[
  \arg\max_{\omega} \mathcal{L}(T; \omega) - \lambda \mathcal{R}(\omega)
  \]

- Where \( \mathcal{R}(\omega) \) is the regularization function

- \( \lambda \) controls how much to regularize

- Common functions
  - \( \text{L2: } \mathcal{R}(\omega) \propto \|\omega\|_2 = \|\omega\| = \sqrt{\sum_i \omega_i^2} \) – smaller weights desired
  - \( \text{L0: } \mathcal{R}(\omega) \propto \|\omega\|_0 = \sum_i [[\omega_i > 0]] \) – zero weights desired
    - Non-convex
    - Approximate with \( \text{L1: } \mathcal{R}(\omega) \propto \|\omega\|_1 = \sum_i |\omega_i| \)
Logistic Regression with L2 Regularization

- Perhaps most common classifier in NLP

\[ L(T; \omega) - \lambda R(\omega) = \sum_{t=1}^{|T|} \log \left( \frac{e^{\omega \cdot \phi(x_t, y_t) / Z_x}}{Z_x} \right) - \frac{\lambda}{2} \| \omega \|^2 \]

- What are the new partial derivatives?

\[ \frac{\partial}{\partial w_i} L(T; \omega) - \frac{\partial}{\partial w_i} \lambda R(\omega) \]

- We know \( \frac{\partial}{\partial w_i} L(T; \omega) \)

- Just need \( \frac{\partial}{\partial w_i} \frac{\lambda}{2} \| \omega \|^2 = \frac{\partial}{\partial w_i} \frac{\lambda}{2} \left( \sqrt{\sum_i \omega_i^2} \right)^2 = \frac{\partial}{\partial w_i} \frac{\lambda}{2} \sum_i \omega_i^2 = \lambda \omega_i \)
Support Vector Machines

Hinge-loss formulation: L2 regularization already happening!

\[
\omega = \arg \min_{\omega} \mathcal{L}(T; \omega) + \lambda \mathcal{R}(\omega)
\]

\[
= \arg \min_{\omega} \sum_{t=1}^{\lvert T \rvert} \text{loss}((x_t, y_t); \omega) + \lambda \mathcal{R}(\omega)
\]

\[
= \arg \min_{\omega} \sum_{t=1}^{\lvert T \rvert} \max (0, 1 + \max_{y \neq y_t} \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t)) + \lambda \mathcal{R}(\omega)
\]

\[
= \arg \min_{\omega} \sum_{t=1}^{\lvert T \rvert} \max (0, 1 + \max_{y \neq y_t} \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t)) + \frac{\lambda}{2} \|\omega\|^2
\]

\[\uparrow\text{ SVM optimization} \uparrow\]
SVMs vs. Logistic Regression

\[
\omega = \arg \min_{\omega} \mathcal{L}(\mathcal{T}; \omega) + \lambda \mathcal{R}(\omega)
\]

\[
= \arg \min_{\omega} \sum_{t=1}^{|\mathcal{T}|} \text{loss}(x_t, y_t; \omega) + \lambda \mathcal{R}(\omega)
\]
SVMs vs. Logistic Regression

\[
\omega = \arg \min_{\omega} \mathcal{L}(\mathcal{T}; \omega) + \lambda \mathcal{R}(\omega)
\]

\[
= \arg \min_{\omega} \sum_{t=1}^{\mathcal{T}} \text{loss}(x_t, y_t; \omega) + \lambda \mathcal{R}(\omega)
\]

SVMs/hinge-loss: \(\max (0, 1 + \max_{y \neq y_t} (\omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t)))\)

\[
\omega = \arg \min_{\omega} \sum_{t=1}^{\mathcal{T}} \max (0, 1 + \max_{y \neq y_t} \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t)) + \frac{\lambda}{2} \|\omega\|^2
\]
SVMs vs. Logistic Regression

\[ \omega = \arg\min_{\omega} L(T; \omega) + \lambda R(\omega) \]

\[ = \arg\min_{\omega} \sum_{t=1}^{\left| T \right|} \text{loss}((x_t, y_t); \omega) + \lambda R(\omega) \]

SVMs/hinge-loss: \( \max (0, 1 + \max_{y \neq y_t} (\omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t))) \)

\[ \omega = \arg\min_{\omega} \sum_{t=1}^{\left| T \right|} \max_{y \neq y_t} (0, 1 + \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t)) + \frac{\lambda}{2} ||\omega||^2 \]

Logistic Regression/log-loss: \( -\log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_x} \right) \)

\[ \omega = \arg\min_{\omega} \sum_{t=1}^{\left| T \right|} -\log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_x} \right) + \frac{\lambda}{2} ||\omega||^2 \]
Generalized Linear Classifiers

\[ \omega = \arg \min_{\omega} \mathcal{L}(T; \omega) + \lambda R(\omega) = \arg \min_{\omega} \sum_{t=1}^{\lvert T \rvert} \text{loss}((x_t, y_t); \omega) + \lambda R(\omega) \]
Online Learning
Online vs. Batch Learning

Batch($\mathcal{T}$);

▶ for 1 \ldots N

▶ $\omega \leftarrow \text{update}(\mathcal{T}; \omega)$

▶ return $\omega$

Online($\mathcal{T}$);

▶ for 1 \ldots N

▶ for $(x_t, y_t) \in \mathcal{T}$

▶ $\omega \leftarrow \text{update}((x_t, y_t); \omega)$

▶ end for

▶ end for

▶ return $\omega$

E.g., SVMs, logistic regression, NB

E.g., Perceptron

$\omega = \omega + \phi(x_t, y_t) - \phi(x_t, y)$
Online vs. Batch Learning

- Online algorithms
  - Tend to converge more quickly
  - Often easier to implement
  - Require more hyperparameter tuning (exception Perceptron)
  - More unstable convergence

- Batch algorithms
  - Tend to converge more slowly
  - Implementation more complex (quad prog, LBFGs)
  - Typically more robust to hyperparameters
  - More stable convergence
Gradient Descent Reminder

- Let $\mathcal{L}(\mathcal{T}; \omega) = \sum_{t=1}^{\mathcal{T}} \text{loss}((x_t, y_t); \omega)$
  - Set $\omega^0 = O^m$
  - Iterate until convergence

$$
\omega^i = \omega^{i-1} - \alpha \nabla \mathcal{L}(\mathcal{T}; \omega^{i-1}) = \omega^{i-1} - \sum_{t=1}^{\mathcal{T}} \alpha \nabla \text{loss}((x_t, y_t); \omega^{i-1})
$$

- $\alpha > 0$ and set so that $\mathcal{L}(\mathcal{T}; \omega^i) < \mathcal{L}(\mathcal{T}; \omega^{i-1})$
Gradient Descent Reminder

- Let $\mathcal{L}(\mathcal{T}; \omega) = \sum_{t=1}^{\lvert \mathcal{T} \rvert} \text{loss}((x_t, y_t); \omega)$
  - Set $\omega^0 = O_m$
  - Iterate until convergence

$$\omega^i = \omega^{i-1} - \alpha \nabla \mathcal{L}(\mathcal{T}; \omega^{i-1}) = \omega^{i-1} - \sum_{t=1}^{\lvert \mathcal{T} \rvert} \alpha \nabla \text{loss}((x_t, y_t); \omega^{i-1})$$

- $\alpha > 0$ and set so that $\mathcal{L}(\mathcal{T}; \omega^i) < \mathcal{L}(\mathcal{T}; \omega^{i-1})$

- **Stochastic Gradient Descent (SGD)**
  - Approximate $\nabla \mathcal{L}(\mathcal{T}; \omega)$ with single $\nabla \text{loss}((x_t, y_t); \omega)$
Stochastic Gradient Descent

- Let $\mathcal{L}(\mathcal{T}; \omega) = \sum_{t=1}^{\mathcal{|T|}} \text{loss}((x_t, y_t); \omega)$

- Set $\omega^0 = O^m$

- Iterate until convergence
  - sample $(x_t, y_t) \in \mathcal{T}$ // “stochastic”
  - $\omega^i = \omega^{i-1} - \alpha \nabla \text{loss}((x_t, y_t); \omega)$

- Return $\omega$
Stochastic Gradient Descent

- Let $\mathcal{L}(T; \omega) = \sum_{t=1}^{|T|} \text{loss}(x_t, y_t; \omega)$

- Set $\omega^0 = O^m$

- Iterate until convergence
  - Sample $(x_t, y_t) \in T$ // “stochastic”
  - $\omega^i = \omega^{i-1} - \alpha \nabla \text{loss}(x_t, y_t; \omega)$

- Return $\omega$

In practice

- Set $\omega^0 = O^m$

- For $1 \ldots N$
  - For $(x_t, y_t) \in T$
    - $\omega^i = \omega^{i-1} - \alpha \nabla \text{loss}(x_t, y_t; \omega)$

- Return $\omega$
Stochastic Gradient Descent

- Let $\mathcal{L}(\mathcal{T}; \omega) = \sum_{t=1}^{\vert \mathcal{T} \vert} \text{loss}((x_t, y_t); \omega)$

- Set $\omega^0 = O^m$

- Iterate until convergence
  - Sample $(x_t, y_t) \in \mathcal{T}$  
    - $\omega^i = \omega^{i-1} - \alpha \nabla \text{loss}((x_t, y_t); \omega)$

- Return $\omega$

In practice

Need to solve $\nabla \text{loss}((x_t, y_t); \omega)$

- Set $\omega^0 = O^m$

- For 1 \ldots N
  - For $(x_t, y_t) \in \mathcal{T}$
    - $\omega^i = \omega^{i-1} - \alpha \nabla \text{loss}((x_t, y_t); \omega)$

- Return $\omega$
Online Logistic Regression

- Stochastic Gradient Descent (SGD)
- \( \text{loss}(\{x_t, y_t\}; \omega) = \text{log-loss} \)
- \( \nabla \text{loss}(\{x_t, y_t\}; \omega) = \nabla \left( - \log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_{x_t}} \right) \right) \)
- From logistic regression section:
  \[
  \nabla \left( - \log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_{x_t}} \right) \right) = - \left( \phi(x_t, y_t) - \sum_{y} P(y|x) \phi(x_t, y) \right)
  \]
- Plus regularization term (if part of model)
Online SVMs

- Stochastic Gradient Descent (SGD)
- $\text{loss}((x_t, y_t); \omega) = \text{hinge-loss}$

$$\nabla \text{loss}((x_t, y_t); \omega) = \nabla \left( \max \left( 0, 1 + \max_{y \neq y_t} \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t) \right) \right)$$

- Subgradient is:

$$\begin{cases} 0, & \text{if } \omega \cdot \phi(x_t, y_t) - \max_y \omega \cdot \phi(x_t, y) \geq 1 \\ \phi(x_t, y) - \phi(x_t, y_t), & \text{otherwise, where } y = \max_y \omega \cdot \phi(x_t, y) \end{cases}$$

- Plus regularization term (required for SVMs)
Perceptron and Hinge-Loss

SVM subgradient update looks like perceptron update

$$\omega^i = \omega^{i-1} - \alpha \begin{cases} 
0, & \text{if } \omega \cdot \phi(x_t, y_t) - \max_y \omega \cdot \phi(x_t, y) \geq 1 \\
\phi(x_t, y) - \phi(x_t, y_t), & \text{otherwise, where } y = \max_y \omega \cdot \phi(x_t, y) 
\end{cases}$$

Perceptron

$$\omega^i = \omega^{i-1} - \alpha \begin{cases} 
0, & \text{if } \omega \cdot \phi(x_t, y_t) - \max_y \omega \cdot \phi(x_t, y) \geq 0 \\
\phi(x_t, y) - \phi(x_t, y_t), & \text{otherwise, where } y = \max_y \omega \cdot \phi(x_t, y) 
\end{cases}$$

where $\alpha = 1$, note $\phi(x_t, y) - \phi(x_t, y_t)$ not $\phi(x_t, y_t) - \phi(x_t, y)$ since ‘−’ (descent)

Perceptron = SGD with no-margin hinge-loss

$$\max \left( 0, 1 + \max_{y \neq y_t} \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t) \right)$$
Margin Infused Relaxed Algorithm (MIRA)

Batch (SVMs):

\[
\min \frac{1}{2} \| \omega \|^2
\]

such that:

\[
\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq 1
\]

\(\forall (x_t, y_t) \in \mathcal{T} \) and \(y' \in \mathcal{Y}_t\)

Online (MIRA):

Training data: \( \mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}} \)

1. \( \omega^{(0)} = 0; \ i = 0 \)
2. for \( n : 1..N \)
3. for \( t : 1..T \)
4. \( \omega^{(i+1)} = \arg \min_{\omega^*} \| \omega^* - \omega^{(i)} \| \)
   such that:
   \[
   \omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq 1
   \]
   \(\forall y' \in \mathcal{Y}_t\)
5. \( i = i + 1 \)
6. return \( \omega^i \)

- MIRA has much smaller optimizations with only \( |\mathcal{Y}_t| \) constraints
Quick Summary
Linear Classifiers

- Naive Bayes, Perceptron, Logistic Regression and SVMs
- Generative vs. Discriminative
- Objective functions and loss functions
  - Log-loss, min error and hinge loss
  - Generalized linear classifiers
- Regularization
- Online vs. Batch learning
Non-linear Classifiers
Non-Linear Classifiers

- Some data sets require more than a linear classifier to be correctly modeled
- A lot of models out there
  - K-Nearest Neighbours
  - Decision Trees
  - **Kernels**
  - Neural Networks
Kernels

- A kernel is a similarity function between two points that is symmetric and positive semi-definite, which we denote by:

\[ \varphi(x_t, x_r) \in \mathbb{R} \]

- Let \( M \) be a \( n \times n \) matrix such that ...

\[ M_{t,r} = \varphi(x_t, x_r) \]

- ... for any \( n \) points. Called the Gram matrix.

- Symmetric:

\[ \varphi(x_t, x_r) = \varphi(x_r, x_t) \]

- Positive definite: for all non-zero \( \mathbf{v} \)

\[ \mathbf{v}M\mathbf{v}^T \geq 0 \]
Kernels

- Mercer’s Theorem: for any kernel $\varphi$, there exists an $\phi$, such that:
  \[ \varphi(x_t, x_r) = \phi(x_t) \cdot \phi(x_r) \]

- Since our features are over pairs $(x, y)$, we will write kernels over pairs
  \[ \varphi((x_t, y_t), (x_r, y_r)) = \phi(x_t, y_t) \cdot \phi(x_r, y_r) \]
Kernel Trick – Perceptron Algorithm

Training data: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\left|\mathcal{T}\right|}$

1. $\omega^{(0)} = 0; \quad i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $y = \text{arg max}_y \omega^{(i)} \cdot \phi(x_t, y)$
5. if $y \neq y_t$
6. $\omega^{(i+1)} = \omega^{(i)} + \phi(x_t, y_t) - \phi(x_t, y)$
7. $i = i + 1$
8. return $\omega^i$

- Each feature function $\phi(x_t, y_t)$ is added and $\phi(x_t, y)$ is subtracted to $\omega$ say $\alpha_{y,t}$ times
  - $\alpha_{y,t}$ is the # of times during learning label $y$ is predicted for example $t$

- Thus,

$$\omega = \sum_{t,y} \alpha_{y,t} [\phi(x_t, y_t) - \phi(x_t, y)]$$
Kernel Trick – Perceptron Algorithm

► We can re-write the argmax function as:

\[ y^* = \arg \max_{y^*} \sum_{t,y} \alpha_{y,t} [\phi(x_t, y_t) - \phi(x_t, y)] \cdot \phi(x_t, y^*) \]

\[ = \arg \max_{y^*} \sum_{t,y} \alpha_{y,t} [\phi(x_t, y_t) \cdot \phi(x_t, y^*) - \phi(x_t, y) \cdot \phi(x_t, y^*)] \]

\[ = \arg \max_{y^*} \sum_{t,y} \alpha_{y,t} [\mathcal{K}(x_t, y_t), (x_t, y^*)) - \mathcal{K}(x_t, y), (x_t, y^*))] \]

► We can then re-write the perceptron algorithm strictly with kernels
Kernel Trick – Perceptron Algorithm

Training data: \( \mathcal{T} = \{ (x_t, y_t) \}_{t=1}^{\mathcal{T}} \)

1. \( \forall y, t \) set \( \alpha_{y,t} = 0 \)
2. for \( n : 1..N \)
3. for \( t : 1..T \)
4. Let \( y^* = \text{arg max}_{y^*} \sum_{t,y} \alpha_{y,t} [\varphi((x_t, y_t), (x_t, y^*)) - \varphi((x_t, y), (x_t, y^*))] \)
5. if \( y^* \neq y_t \)
6. \( \alpha_{y^*,t} = \alpha_{y^*,t} + 1 \)

- Given a new instance \( x \)

\[ y^* = \text{arg max}_{y^*} \sum_{t,y} \alpha_{y,t} [\varphi((x_t, y_t), (x, y^*)) - \varphi((x_t, y), (x, y^*))] \]

- But it seems like we have just complicated things???
Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space.
- Computing a non-linear kernel is often better computationally than calculating the corresponding dot product in the high dimension feature space.
- Thus, kernels allow us to efficiently learn non-linear classifiers.
Linear Classifiers in High Dimension

\[ \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]

\[ (x_1, x_2) \rightarrow (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \]
Example: Polynomial Kernel

$\phi(x) \in \mathbb{R}^M$, $d \geq 2$

$\varphi(x_t, x_s) = (\phi(x_t) \cdot \phi(x_s) + 1)^d$

$O(M)$ to calculate for any $d$!!

But in the original feature space (primal space)

Consider $d = 2$, $M = 2$, and $\phi(x_t) = [x_{t,1}, x_{t,2}]$

$$(\phi(x_t) \cdot \phi(x_s) + 1)^2 = ([x_{t,1}, x_{t,2}] \cdot [x_{s,1}, x_{s,2}] + 1)^2$$

$$= (x_{t,1}x_{s,1} + x_{t,2}x_{s,2} + 1)^2$$

$$= (x_{t,1}x_{s,1})^2 + (x_{t,2}x_{s,2})^2 + 2(x_{t,1}x_{s,1}) + 2(x_{t,2}x_{s,2})$$

$$+ 2(x_{t,1}x_{t,2}x_{s,1}x_{s,2}) + (1)^2$$

which equals:

$$[(x_{t,1})^2, (x_{t,2})^2, \sqrt{2}x_{t,1}, \sqrt{2}x_{t,2}, \sqrt{2}x_{t,1}x_{t,2}, 1] \cdot [(x_{s,1})^2, (x_{s,2})^2, \sqrt{2}x_{s,1}, \sqrt{2}x_{s,2}, \sqrt{2}x_{s,1}x_{s,2}, 1]$$
Popular Kernels

▶ Polynomial kernel

\[ \varphi(x_t, x_s) = (\phi(x_t) \cdot \phi(x_s) + 1)^d \]

▶ Gaussian radial basis kernel (infinite feature space representation!)

\[ \varphi(x_t, x_s) = \exp\left(\frac{-||\phi(x_t) - \phi(x_s)||^2}{2\sigma}\right) \]

▶ String kernels \cite{Lodhi et al. 2002, Collins and Duffy 2002}

▶ Tree kernels \cite{Collins and Duffy 2002}
Kernels Summary

- Can turn a linear classifier into a non-linear classifier
- Kernels project feature space to higher dimensions
  - Sometimes exponentially larger
  - Sometimes an infinite space!
- Can “kernalize” algorithms to make them non-linear
References and Further Reading


K. Crammer, O. Dekel, J. Keshat, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive aggressive algorithms. JMLR.


Maximum entropy Markov models for information extraction and segmentation. In Proc. ICML.


