Introduction to Machine Learning

Linear Classifiers

Lisbon Machine Learning School, 2015

Shay Cohen

School of Informatics, University of Edinburgh
E-mail: scohen@inf.ed.ac.uk

Slides heavily based on Ryan McDonald's slides from 2014

Linear Classifiers

- Go onto ACL Anthology
- Search for: "Naive Bayes", "Maximum Entropy", "Logistic Regression", "SVM", "Perceptron"
- Do the same on Google Scholar
  - "Maximum Entropy" & "NLP" 11,000 hits, 240 before 2000
  - "SVM" & "NLP" 15,000 hits, 556 before 2000
  - "Perceptron" & "NLP", 4,000 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
  - One the most important tools

Experiment

- Document 1 – label: 0; words: $\star \diamond \circ$
- Document 2 – label: 0; words: $\star \heartsuit \triangle$
- Document 3 – label: 1; words: $\star \triangle \spadesuit$
- Document 4 – label: 1; words: $\diamond \triangle \circ$

- New document – words: $\star \diamond \circ$; label ?
- New document – words: $\star \diamond \heartsuit$; label ?
- New document – words: $\star \diamond \spadesuit$; label ?
- New document – words: $\star \triangle \circ$; label ?

Why and how can we do this?

Experiment

- Document 1 – label: 0; words: $\star \diamond \circ$
- Document 2 – label: 0; words: $\star \heartsuit \triangle$
- Document 3 – label: 1; words: $\star \triangle \spadesuit$
- Document 4 – label: 1; words: $\diamond \triangle \circ$

- New document – words: $\star \triangle \circ$; label ?

Label 0 **Label 1**

$$P(0|\star) = \frac{\text{count}(\star \text{ and } 0)}{\text{count}(\star)} = \frac{2}{3} = 0.67 \text{ vs. } P(1|\star) = \frac{\text{count}(\star \text{ and } 1)}{\text{count}(\star)} = \frac{1}{3} = 0.33$$

$$P(0|\triangle) = \frac{\text{count}(\triangle \text{ and } 0)}{\text{count}(\triangle)} = \frac{1}{3} = 0.33 \text{ vs. } P(1|\triangle) = \frac{\text{count}(\triangle \text{ and } 1)}{\text{count}(\triangle)} = \frac{2}{3} = 0.67$$

$$P(0|\circ) = \frac{\text{count}(\circ \text{ and } 0)}{\text{count}(\circ)} = \frac{1}{2} = 0.5 \text{ vs. } P(1|\circ) = \frac{\text{count}(\circ \text{ and } 1)}{\text{count}(\circ)} = \frac{1}{2} = 0.5$$

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: $\boldsymbol{x} \in \mathcal{X}$
  - e.g., document or sentence with some words $\boldsymbol{x} = w_1 \ldots w_n$, or a series of previous actions
- Output: $\boldsymbol{y} \in \mathcal{Y}$
  - e.g., parse tree, document class, part-of-speech tags, word-sense
- Input/Output pair: $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$
  - e.g., a document $\boldsymbol{x}$ and its label $\boldsymbol{y}$
  - Sometimes $\boldsymbol{x}$ is explicit in $\boldsymbol{y}$, e.g., a parse tree $\boldsymbol{y}$ will contain the sentence $\boldsymbol{x}$

General Goal

When given a new input $\boldsymbol{x}$ predict the correct output $\boldsymbol{y}$

But we need to formulate this computationally!

Feature Representations

- We assume a mapping from input $\boldsymbol{x}$ to a high dimensional feature vector
  - $\phi(\boldsymbol{x}) : \mathcal{X} \rightarrow \mathbb{R}^m$
- For many cases, more convenient to have mapping from input-output pairs $(\boldsymbol{x}, \boldsymbol{y})$
  - $\phi(\boldsymbol{x}, \boldsymbol{y}) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- Under certain assumptions, these are equivalent
- Most papers in NLP use $\phi(\boldsymbol{x}, \boldsymbol{y})$
- (Was?) not so common in NLP: $\phi \in \mathbb{R}^m$ (but see word embeddings)
- 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 $\mathbf{v} \in \mathbb{R}^m$, let $\mathbf{v}_j$ be the j^{th} value

Examples

- $\boldsymbol{x}$ is a document and $\boldsymbol{y}$ is a label

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

We expect this feature to have a positive weight, "interest" is a positive indicator for the label "financial"

Examples

- $\boldsymbol{x}$ is a document and $\boldsymbol{y}$ is a label

$$\phi_j(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains the word ``president''} \\ & \text{and } \boldsymbol{y} = \text{``sports''} \\ 0 & \text{otherwise} \end{cases}$$

We expect this feature to have a negative weight?

Examples

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

Punctuation symbols - positive indicator or negative indicator for scientific articles?

Examples

- $\boldsymbol{x}$ is a word and $\boldsymbol{y}$ is a part-of-speech tag

$$\phi_j(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} = \text{``bank'' and } \boldsymbol{y} = \text{ Verb} \\ 0 & \text{otherwise} \end{cases}$$

What weight would it get?

Example 2

- $\boldsymbol{x}$ is a name, $\boldsymbol{y}$ is a label classifying the name

$$\phi_0(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``George''} \\ & \text{and } \boldsymbol{y} = \text{``Person''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_1(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``Washington''} \\ & \text{and } \boldsymbol{y} = \text{``Person''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_2(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``Bridge''} \\ & \text{and } \boldsymbol{y} = \text{``Person''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_3(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``General''} \\ & \text{and } \boldsymbol{y} = \text{``Person''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_4(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``George''} \\ & \text{and } \boldsymbol{y} = \text{``Object''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_5(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``Washington''} \\ & \text{and } \boldsymbol{y} = \text{``Object''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_6(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``Bridge''} \\ & \text{and } \boldsymbol{y} = \text{``Object''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_7(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``General''} \\ & \text{and } \boldsymbol{y} = \text{``Object''} \\ 0 & \text{otherwise} \end{cases}$$

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

Block Feature Vectors

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

- Instead of $\phi(\boldsymbol{x}, \boldsymbol{y}) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$ over input/outputs $(\boldsymbol{x}, \boldsymbol{y})$
- Let $\phi(\boldsymbol{x}) : \mathcal{X} \rightarrow \mathbb{R}^{m'}$ (e.g.,$m' = m/|\mathcal{Y}|$)
  - i.e., feature representation only over inputs $\boldsymbol{x}$
- Equivalent when $\phi(x, y)$ includes y as a non-decomposable object
- Disadvantages to $\phi(x)$ formulation: no complex features over properties of labels
- Advantages: can make math cleaner, especially with binary classification

Feature Representations - $\phi(x)$ vs. $\phi(x, y)$

- $\phi(\boldsymbol{x}, \boldsymbol{y})$
  - $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Person $\rightarrow \phi(\boldsymbol{x}, \boldsymbol{y}) =$ [1 1 0 1 0 0 0 0]
  - $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Object $\rightarrow \phi(\boldsymbol{x}, \boldsymbol{y}) =$ [0 0 0 0 1 1 0 1]
- $\phi(\boldsymbol{x})$
  - $\boldsymbol{x}$=General George Washington $\rightarrow \phi(\boldsymbol{x}) =$ [1 1 0 1]
- Different ways of representing same thing
- In this case, can deterministically map from $\phi(\boldsymbol{x})$ to $\phi(\boldsymbol{x}, \boldsymbol{y})$ given $\boldsymbol{y}$

Linear Classifiers

- **Linear classifier**: score (or probability) of a particular classification is based on a linear combination of features and their weights
- Let $\boldsymbol{\omega} \in \mathbb{R}^m$ be a high dimensional weight vector
- Assume that $\boldsymbol{\omega}$ is known
  - **Multiclass Classification**: $\mathcal{Y} = \{0, 1, \ldots, N\}$

$$\begin{aligned} \boldsymbol{y} &= \arg\max_{\boldsymbol{y}} \; \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \\ &= \arg\max_{\boldsymbol{y}} \; \sum_{j=0}^{m} \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}, \boldsymbol{y}) \end{aligned}$$

  - **Binary Classification** just a special case of multiclass

Linear Classifiers – $\phi(x)$

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

$$\boldsymbol{y} = \arg\max_{\boldsymbol{y}} \; \boldsymbol{\omega_y} \cdot \boldsymbol{\phi}(\boldsymbol{x})$$

- $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$
  - $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Person $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) = [1\ 1\ 0\ 1\ 0\ 0\ 0\ 0]$
  - $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Object $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) = [0\ 0\ 0\ 0\ 1\ 1\ 0\ 1]$
  - Single $\boldsymbol{\omega} \in \mathbb{R}^8$
- $\boldsymbol{\phi}(\boldsymbol{x})$
  - $\boldsymbol{x}$=General George Washington $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [1\ 1\ 0\ 1]$
  - Two parameter vectors $\boldsymbol{\omega}_0 \in \mathbb{R}^4$, $\boldsymbol{\omega}_1 \in \mathbb{R}^4$

Linear Classifiers - Bias Terms

- Often linear classifiers presented as

$$\boldsymbol{y} \quad = \quad \arg\max_{\boldsymbol{y}} \sum_{j=0}^{m} \omega_j \times \phi_j(\boldsymbol{x}, \boldsymbol{y}) + b_{\boldsymbol{y}}$$

- Where b is a bias or offset term
- Sometimes this is folded into ϕ

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

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

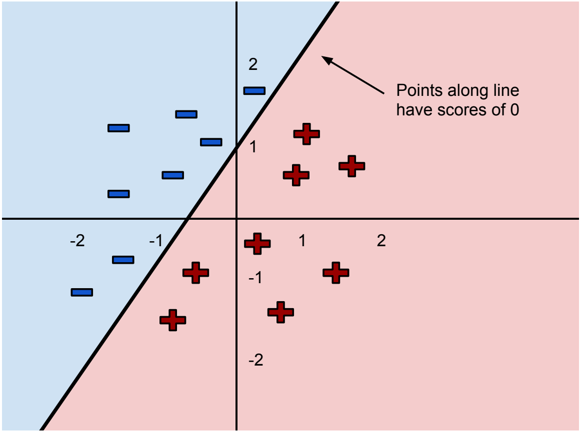
$$\phi_4(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \boldsymbol{y} = \text{“Person”} \\ 0 & \text{otherwise} \end{cases} \qquad \phi_9(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \boldsymbol{y} = \text{“Object”} \\ 0 & \text{otherwise} \end{cases}$$

- ω_4 and ω_9 are now the bias terms for the labels

Binary Linear Classifier

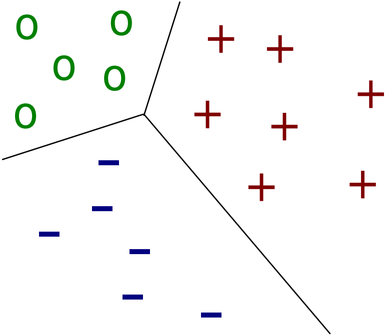
Let's say $\boldsymbol{\omega} = (1, -1)$ and $b_{\boldsymbol{y}} = 1$, $\forall \boldsymbol{y}$

Then $\boldsymbol{\omega}$ is a line (generally a hyperplane) that divides all points:



Multiclass Linear Classifier

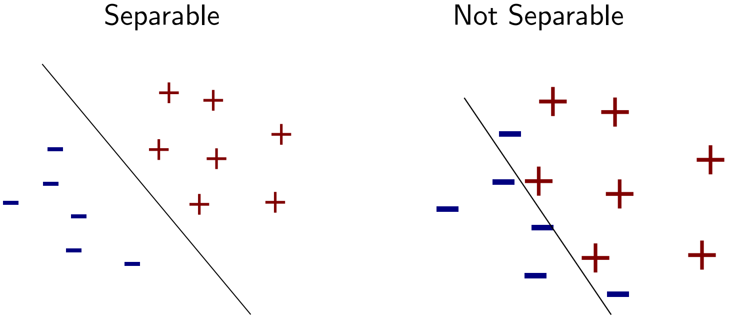
Defines regions of space. Visualization difficult.



- i.e., + are all points $(\boldsymbol{x}, \boldsymbol{y})$ where $+ = \arg\max_{\boldsymbol{y}}\ \boldsymbol{\omega} \cdot \phi(\boldsymbol{x}, \boldsymbol{y})$

Separability

- A set of points is separable, if there exists a ω such that classification is perfect



- This can also be defined mathematically (and we will do that shortly)

Machine Learning – finding ω

We now have a way to make dcisions... If we have a $\boldsymbol{\omega}$. But where do we get this $\boldsymbol{\omega}$?

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

Objective Functions

- $\mathcal{L}(\cdot)$ is called the objective function
- Usually we can decompose $\mathcal{L}$ by training pairs $(\boldsymbol{x}, \boldsymbol{y})$
  - $\mathcal{L}(\mathcal{T}; \boldsymbol{\omega}) \propto \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{T}} loss((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\omega})$
  - *loss* is a function that measures some value correlated with errors of parameters $\boldsymbol{\omega}$ on instance $(\boldsymbol{x}, \boldsymbol{y})$
- Defining $\mathcal{L}(\cdot)$ and *loss* is core of linear classifiers in machine learning
- Example: $y \in \{1, -1\}$, $f(x|w)$ is the prediction we make for x using w
- Loss is:

Supervised Learning – Assumptions

- Assumption: $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ 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

Naive Bayes

Probabilistic Models

- Let's put aside linear classifiers for a moment
- Here is another approach to decision making
  - Probabilistically model $P(\boldsymbol{y}|\boldsymbol{x})$
  - If we can define this distribution, then classification becomes
    - $\arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x})$

Bayes Rule

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

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

$$\arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) \propto \arg\max_{\boldsymbol{y}} P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y})$$

- Since $\boldsymbol{x}$ is fixed
- $P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y}) = P(\boldsymbol{x}, \boldsymbol{y})$: 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)

- We need to decide on the structure of $P(\boldsymbol{x}, \boldsymbol{y})$
- $P(\boldsymbol{x}|\boldsymbol{y}) = P(\phi(\boldsymbol{x})|\boldsymbol{y}) = P(\phi_1(\boldsymbol{x}), \ldots, \phi_m(\boldsymbol{x})|\boldsymbol{y})$

Naive Bayes Assumption

(conditional independence)

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

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

Naive Bayes – Learning

- Input: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$
- Let $\phi_i(\boldsymbol{x}) \in \{1, \ldots, F_i\}$ – categorical; common in NLP
- Parameters $\mathcal{P} = \{P(\boldsymbol{y}), P(\phi_i(\boldsymbol{x})|\boldsymbol{y})\}$
  - Both $P(\boldsymbol{y})$ and $P(\phi_i(\boldsymbol{x})|\boldsymbol{y})$ are multinomials

Maximum Likelihood Estimation

- What's left? Defining an objective $\mathcal{L}(\mathcal{T})$
- P plays the role of w
- What objective to use?
- Objective: Maximum Likelihood Estimation (MLE)

$$\mathcal{L}(\mathcal{T}) = \prod_{t=1}^{|\mathcal{T}|} P(\boldsymbol{x}_t, \boldsymbol{y}_t) = \prod_{t=1}^{|\mathcal{T}|} \left(P(\boldsymbol{y}_t) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right)$$

$$\mathcal{P} = \arg\max_{\mathcal{P}} \prod_{t=1}^{|\mathcal{T}|} \left(P(\boldsymbol{y}_t) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right)$$

Naive Bayes – Learning

MLE has closed form solution!! (more later) – count and normalize

$$\mathcal{P} = \arg\max_{\mathcal{P}} \prod_{t=1}^{|\mathcal{T}|} \left(P(\boldsymbol{y}_t) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right)$$

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

$$P(\phi_i(\boldsymbol{x})|\boldsymbol{y}) = \frac{\sum_{t=1}^{|\mathcal{T}|} [[\phi_i(\boldsymbol{x}_t) = \phi_i(\boldsymbol{x}) \text{ and } \boldsymbol{y}_t = \boldsymbol{y}]]}{\sum_{t=1}^{|\mathcal{T}|} [[\boldsymbol{y}_t = \boldsymbol{y}]]}$$

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

Thus, these are just normalized counts over events in $\mathcal{T}$

Intuitively makes sense!

Naive Bayes Example

- $\phi_i(\boldsymbol{x}) \in 0, 1, \forall i$
- doc 1: $\boldsymbol{y}_1 = 0$, $\phi_0(\boldsymbol{x}_1) = 1$, $\phi_1(\boldsymbol{x}_1) = 1$
- doc 2: $\boldsymbol{y}_2 = 0$, $\phi_0(\boldsymbol{x}_2) = 0$, $\phi_1(\boldsymbol{x}_2) = 1$
- doc 3: $\boldsymbol{y}_3 = 1$, $\phi_0(\boldsymbol{x}_3) = 1$, $\phi_1(\boldsymbol{x}_3) = 0$

- Two label parameters $P(\boldsymbol{y} = 0)$, $P(\boldsymbol{y} = 1)$
- Eight feature parameters
  - 2 (labels) * 2 (features) * 2 (feature values)
  - E.g., $\boldsymbol{y} = 0$ and $\phi_0(\boldsymbol{x}) = 1$: $P(\phi_0(\boldsymbol{x}) = 1|\boldsymbol{y} = 0)$
- We really have one label parameter and 2 * 2 * (2 - 1) feature parameters

- $P(\boldsymbol{y} = 0) = 2/3$, $P(\boldsymbol{y} = 1) = 1/3$
- $P(\phi_0(\boldsymbol{x}) = 1|\boldsymbol{y} = 0) = 1/2$, $P(\phi_1(\boldsymbol{x}) = 0|\boldsymbol{y} = 1) = 1/1$

Naive Bayes Document Classification

- doc 1: $\boldsymbol{y}_1 =$ sports, "hockey is fast"
- doc 2: $\boldsymbol{y}_2 =$ politics, "politicians talk fast"
- doc 3: $\boldsymbol{y}_3 =$ politics, "washington is sleazy"

- $\phi_0(\boldsymbol{x}) = 1$ iff doc has word 'hockey', 0 o.w.
- $\phi_1(\boldsymbol{x}) = 1$ iff doc has word 'is', 0 o.w.
- $\phi_2(\boldsymbol{x}) = 1$ iff doc has word 'fast', 0 o.w.
- $\phi_3(\boldsymbol{x}) = 1$ iff doc has word 'politicians', 0 o.w.
- $\phi_4(\boldsymbol{x}) = 1$ iff doc has word 'talk', 0 o.w.
- $\phi_5(\boldsymbol{x}) = 1$ iff doc has word 'washington', 0 o.w.
- $\phi_6(\boldsymbol{x}) = 1$ iff doc has word 'sleazy', 0 o.w.

Your turn? What is $P(\mathrm{sports})$? What is $P(\phi_0(0) = 1|\mathrm{politics})$?

Deriving MLE

$$\mathcal{P} \quad = \quad \arg\max_{\mathcal{P}} \prod_{t=1}^{|\mathcal{T}|} \left(P(\boldsymbol{y}_t) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right)$$

Deriving MLE (for handout)

$$
\begin{aligned}
\mathcal{P} &= \arg\max_{\mathcal{P}} \prod_{t=1}^{|\mathcal{T}|} \left(P(\boldsymbol{y}_t) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right) \\
&= \arg\max_{\mathcal{P}} \sum_{t=1}^{|\mathcal{T}|} \left(\log P(\boldsymbol{y}_t) + \sum_{i=1}^{m} \log P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right) \\
&= \arg\max_{P(\boldsymbol{y})} \sum_{t=1}^{|\mathcal{T}|} \log P(\boldsymbol{y}_t) + \arg\max_{P(\phi_i(\boldsymbol{x})|\boldsymbol{y})} \sum_{t=1}^{|\mathcal{T}|} \sum_{i=1}^{m} \log P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t)
\end{aligned}
$$

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

Deriving MLE

$$\mathcal{P} = \arg\max_{P(\boldsymbol{y})} \sum_{t=1}^{|\mathcal{T}|} \log P(\boldsymbol{y}_t) + \arg\max_{P(\phi_i(\boldsymbol{x})|\boldsymbol{y})} \sum_{t=1}^{|\mathcal{T}|} \sum_{i=1}^{m} \log P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t)$$

Both optimizations are of the form

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

For example:

$$\arg\max_{P(\boldsymbol{y})} \sum_{t=1}^{|\mathcal{T}|} \log P(\boldsymbol{y}_t) = \arg\max_{P(\boldsymbol{y})} \sum_{\boldsymbol{y}} \text{count}(\boldsymbol{y}, \mathcal{T}) \log P(\boldsymbol{y})$$

$$\text{such that } \textstyle\sum_{\boldsymbol{y}} P(\boldsymbol{y}) = 1,\ P(\boldsymbol{y}) \geq 0$$

Deriving MLE

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

Introduce Lagrangian multiplier λ, optimization becomes

$$\arg\max_{P,\lambda}\ \sum_v \text{count}(v) \log P(v) - \lambda \left(\sum_v P(v) - 1\right)$$

Derivative:

Set to zero:

Final solution:

Deriving MLE (for handout)

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

Introduce Lagrangian multiplier λ, optimization becomes

$$\arg\max_{P,\lambda}\ \sum_v \text{count}(v) \log P(v) - \lambda \left(\sum_v P(v) - 1\right)$$

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

$$\mathcal{P} = \underset{\mathcal{P}}{\arg\max} \prod_{t=1}^{|\mathcal{T}|} \left(P(\boldsymbol{y}_t) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right)$$

$$= \underset{P(\boldsymbol{y})}{\arg\max} \sum_{t=1}^{|\mathcal{T}|} \log P(\boldsymbol{y}_t) + \underset{P(\phi_i(\boldsymbol{x})|\boldsymbol{y})}{\arg\max} \sum_{t=1}^{|\mathcal{T}|} \sum_{i=1}^{m} \log P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t)$$

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

$$P(\phi_i(\boldsymbol{x})|\boldsymbol{y}) = \frac{\sum_{t=1}^{|\mathcal{T}|} [[\phi_i(\boldsymbol{x}_t) = \phi_i(\boldsymbol{x}) \text{ and } \boldsymbol{y}_t = \boldsymbol{y}]]}{\sum_{t=1}^{|\mathcal{T}|} [[\boldsymbol{y}_t = \boldsymbol{y}]]}$$

NB is a linear classifier

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

$$\arg\max_{\boldsymbol{y}} \; P(\boldsymbol{y}|\boldsymbol{\phi}(\boldsymbol{x})) \quad \propto \quad \arg\max_{\boldsymbol{y}} \; P(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{y}) = \arg\max_{\boldsymbol{y}} \; P(\boldsymbol{y}) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x})|\boldsymbol{y}) =$$

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

NB is a linear classifier (for handout)

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

$$
\begin{aligned}
\arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{\phi}(\boldsymbol{x})) \quad &\propto \quad \arg\max_{\boldsymbol{y}} P(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{y}) = \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x})|\boldsymbol{y}) \\
&= \quad \arg\max_{\boldsymbol{y}} \log P(\boldsymbol{y}) + \sum_{i=1}^{m} \log P(\phi_i(\boldsymbol{x})|\boldsymbol{y}) \\
&= \quad \arg\max_{\boldsymbol{y}} \boldsymbol{\omega}_{\boldsymbol{y}} + \sum_{i=1}^{m} \boldsymbol{\omega}_{\phi_i(\boldsymbol{x}),\boldsymbol{y}} \\
&= \quad \arg\max_{\boldsymbol{y}} \sum_{\boldsymbol{y}'} \boldsymbol{\omega}_{\boldsymbol{y}} \psi_{\boldsymbol{y}'}(\boldsymbol{y}) + \sum_{i=1}^{m} \sum_{j=1}^{F_i} \boldsymbol{\omega}_{\phi_i(\boldsymbol{x}),\boldsymbol{y}} \psi_{i,j}(\boldsymbol{x})
\end{aligned}
$$

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

Smoothing

- doc 1: $\boldsymbol{y}_1 =$ sports, "hockey is fast"
- doc 2: $\boldsymbol{y}_2 =$ politics, "politicians talk fast"
- doc 3: $\boldsymbol{y}_3 =$ 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 $P(\boldsymbol{x}, \boldsymbol{y})$
  - Statistical model must explain generation of input

- Occam's Razor: why model input?
- Discriminative models
  - Use $\mathcal{L}$ that directly optimizes $P(\boldsymbol{y}|\boldsymbol{x})$ (or something related)
  - Logistic Regression – MLE of $P(\boldsymbol{y}|\boldsymbol{x})$
  - Perceptron and SVMs – minimize classification error

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

Logistic Regression

Logistic Regression

Define a conditional probability:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}}, \qquad \text{where } Z_{\boldsymbol{x}} = \sum_{\boldsymbol{y}'\in\mathcal{Y}} e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}')}$$

Note: still a linear classifier

$$\begin{aligned}
\arg\max_{\boldsymbol{y}} \; P(\boldsymbol{y}|\boldsymbol{x}) &= \arg\max_{\boldsymbol{y}} \; \frac{e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}} \\
&= \arg\max_{\boldsymbol{y}} \; e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})} \\
&= \arg\max_{\boldsymbol{y}} \; \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})
\end{aligned}$$

Logistic Regression

$$P(y|x) = \frac{e^{\boldsymbol{\omega}\cdot\phi(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}}$$

- Q: How do we learn weights $\boldsymbol{\omega}$
- A: Set weights to maximize log-likelihood of training data:

$$\begin{aligned} \boldsymbol{\omega} &= \arg\max_{\boldsymbol{\omega}} \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) \\ &= \arg\max_{\boldsymbol{\omega}} \prod_{t=1}^{|\mathcal{T}|} P(\boldsymbol{y}_t|\boldsymbol{x}_t) = \arg\max_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \log P(\boldsymbol{y}_t|\boldsymbol{x}_t) \end{aligned}$$

- In a nutshell we set the weights $\boldsymbol{\omega}$ so that we assign as much probability to the correct label $\boldsymbol{y}$ for each $\boldsymbol{x}$ in the training set

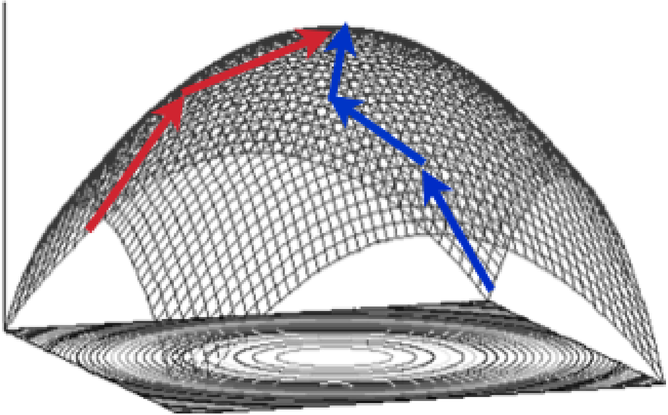
Logistic Regression

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{e^{\boldsymbol{\omega}\cdot\phi(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}}, \qquad \text{where } Z_{\boldsymbol{x}} = \sum_{\boldsymbol{y}'\in\mathcal{Y}} e^{\boldsymbol{\omega}\cdot\phi(\boldsymbol{x},\boldsymbol{y}')}$$

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \log P(\boldsymbol{y}_t|\boldsymbol{x}_t) \ (*)$$

- 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



Gradient Ascent

- Let $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \sum_{t=1}^{|\mathcal{T}|} \log\left(e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)}/Z_{\boldsymbol{x}}\right)$
- Want to find $\arg\max_{\boldsymbol{\omega}} \mathcal{L}(\mathcal{T};\boldsymbol{\omega})$
  - Set $\boldsymbol{\omega}^0 = O^m$
  - Iterate until convergence

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

- $\alpha > 0$ and set so that $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}^i) > \mathcal{L}(\mathcal{T};\boldsymbol{\omega}^{i-1})$
- $\nabla \mathcal{L}(\mathcal{T};\boldsymbol{\omega})$ is gradient of $\mathcal{L}$ w.r.t. $\boldsymbol{\omega}$
  - A gradient is all partial derivatives over variables w_i
  - i.e., $\nabla \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \left(\frac{\partial}{\partial \boldsymbol{\omega}_0}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \frac{\partial}{\partial \boldsymbol{\omega}_1}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \ldots, \frac{\partial}{\partial \boldsymbol{\omega}_m}\mathcal{L}(\mathcal{T};\boldsymbol{\omega})\right)$
- Gradient ascent will always find $\boldsymbol{\omega}$ to maximize $\mathcal{L}$

Gradient Descent

- Let $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = - \sum_{t=1}^{|\mathcal{T}|} \log \left(e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)} / Z_{\boldsymbol{x}} \right)$
- Want to find $\arg\min_{\boldsymbol{\omega}} \mathcal{L}(\mathcal{T};\boldsymbol{\omega})$
  - Set $\boldsymbol{\omega}^0 = O^m$
  - Iterate until convergence

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

- $\alpha > 0$ and set so that $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}^i) < \mathcal{L}(\mathcal{T};\boldsymbol{\omega}^{i-1})$
- $\nabla \mathcal{L}(\mathcal{T};\boldsymbol{\omega})$ is gradient of $\mathcal{L}$ w.r.t. $\boldsymbol{\omega}$
  - A gradient is all partial derivatives over variables w_i
  - i.e., $\nabla \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = (\frac{\partial}{\partial \boldsymbol{\omega}_0}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \frac{\partial}{\partial \boldsymbol{\omega}_1}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \ldots, \frac{\partial}{\partial \boldsymbol{\omega}_m}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}))$
- Gradient descent will always find $\boldsymbol{\omega}$ to minimize $\mathcal{L}$

The partial derivatives

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

$$\begin{aligned}
\mathcal{L}(\mathcal{T}; \boldsymbol{\omega}) &= \sum_t \log P(\boldsymbol{y}_t | \boldsymbol{x}_t) \\
&= \sum_t \log \frac{e^{\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')}} \\
&= \sum_t \log \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}}
\end{aligned}$$

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

$\frac{\partial}{\partial \boldsymbol{\omega}_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) =$

The partial derivatives 1 (for handout)

$$
\begin{aligned}
\frac{\partial}{\partial \omega_i} \mathcal{L}(\mathcal{T}; \omega) &= \frac{\partial}{\partial \omega_i} \sum_t \log \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} \\
&= \sum_t \frac{\partial}{\partial \omega_i} \log \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} \\
&= \sum_t \left(\frac{Z_{\boldsymbol{x}_t}}{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}\right)\left(\frac{\partial}{\partial \omega_i} \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}}\right)
\end{aligned}
$$

The partial derivatives

Now, $\frac{\partial}{\partial \boldsymbol{\omega}_i} \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} =$

The partial derivatives 2 (for handout)

Now,

$$
\begin{aligned}
\frac{\partial}{\partial \omega_i} \frac{e^{\sum_j \omega_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} &= \frac{Z_{\boldsymbol{x}_t} \frac{\partial}{\partial \omega_i} e^{\sum_j \omega_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}_t)} - e^{\sum_j \omega_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}_t)} \frac{\partial}{\partial \omega_i} Z_{\boldsymbol{x}_t}}{Z^2_{\boldsymbol{x}_t}} \\
&= \frac{Z_{\boldsymbol{x}_t} e^{\sum_j \omega_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}_t)} \phi_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - e^{\sum_j \omega_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}_t)} \frac{\partial}{\partial \omega_i} Z_{\boldsymbol{x}_t}}{Z^2_{\boldsymbol{x}_t}} \\
&= \frac{e^{\sum_j \omega_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z^2_{\boldsymbol{x}_t}} \left(Z_{\boldsymbol{x}_t} \phi_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \frac{\partial}{\partial \omega_i} Z_{\boldsymbol{x}_t}\right) \\
&= \frac{e^{\sum_j \omega_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z^2_{\boldsymbol{x}_t}} \Big(Z_{\boldsymbol{x}_t} \phi_i(\boldsymbol{x}_t, \boldsymbol{y}_t) \\
&\qquad - \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \omega_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}')} \phi_i(\boldsymbol{x}_t, \boldsymbol{y}')\Big)
\end{aligned}
$$

because

$$
\frac{\partial}{\partial \omega_i} Z_{\boldsymbol{x}_t} = \frac{\partial}{\partial \omega_i} \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \omega_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}')} = \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \omega_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}')} \phi_i(\boldsymbol{x}_t, \boldsymbol{y}')
$$

The partial derivatives

The partial derivatives 3 (for handout)

From before,

$$\frac{\partial}{\partial \boldsymbol{\omega}_i} \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} = \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z^2_{\boldsymbol{x}_t}} (Z_{\boldsymbol{x}_t} \boldsymbol{\phi}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}')} \boldsymbol{\phi}_i(\boldsymbol{x}_t, \boldsymbol{y}'))$$

Sub this in,

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\omega}_i} \mathcal{L}(\mathcal{T}; \boldsymbol{\omega}) &= \sum_t \Big(\frac{Z_{\boldsymbol{x}_t}}{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}\Big)\Big(\frac{\partial}{\partial \boldsymbol{\omega}_i} \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}}\Big) \\
&= \sum_t \frac{1}{Z_{x_t}} (Z_{\boldsymbol{x}_t} \boldsymbol{\phi}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}')} \boldsymbol{\phi}_i(\boldsymbol{x}_t, \boldsymbol{y}'))) \\
&= \sum_t \boldsymbol{\phi}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t, \boldsymbol{y}')}}{Z_{x_t}} \boldsymbol{\phi}_i(\boldsymbol{x}_t, \boldsymbol{y}') \\
&= \sum_t \boldsymbol{\phi}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}' | \boldsymbol{x}_t) \boldsymbol{\phi}_i(\boldsymbol{x}_t, \boldsymbol{y}')
\end{aligned}$$

FINALLY!!!

- After all that,

$$\frac{\partial}{\partial \omega_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) \;=\; \sum_t \phi_i(\boldsymbol{x}_t,\boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}'\in\mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\phi_i(\boldsymbol{x}_t,\boldsymbol{y}')$$

- And the gradient is:

$$\nabla\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = (\frac{\partial}{\partial \omega_0}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \frac{\partial}{\partial \omega_1}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \ldots, \frac{\partial}{\partial \omega_m}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}))$$

- So we can now use gradient ascent to find $\boldsymbol{\omega}$!!

Logistic Regression Summary

- Define conditional probability

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}}$$

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

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} \sum_t \log P(\boldsymbol{y}_t|\boldsymbol{x}_t)$$

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

$$\frac{\partial}{\partial \omega_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \sum_t \phi_i(\boldsymbol{x}_t,\boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}'\in\mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\phi_i(\boldsymbol{x}_t,\boldsymbol{y}')$$

Logistic Regression = Maximum Entropy

- Well-known equivalence
- Max Ent: maximize entropy subject to constraints on features: $P = \arg\max_P H(P)$ under constraints
  - Empirical feature counts must equal expected counts
- Quick intuition
  - Partial derivative in logistic regression

$$\frac{\partial}{\partial \boldsymbol{\omega}_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \sum_t \phi_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\phi_i(\boldsymbol{x}_t, \boldsymbol{y}')$$

  - 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 $\boldsymbol{\omega}$ that minimizes error

$$\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \sum_{t=1}^{|\mathcal{T}|} 1 - [[\boldsymbol{y}_t = \arg\max_{\boldsymbol{y}} \ \boldsymbol{\omega} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y})]]$$

$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} 1 - [[\boldsymbol{y}_t = \arg\max_{\boldsymbol{y}} \ \boldsymbol{\omega} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y})]]$$

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

- 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

$$1000 \times (\boldsymbol{x}_i, \boldsymbol{y} = 0) = [-1, 1, 0, 0] \quad \text{for} \quad i = 1 \ldots 1000$$
$$1 \times (\boldsymbol{x}_{1001}, \boldsymbol{y} = 1) = [0, 0, 3, 1]$$

- Now consider $\boldsymbol{\omega} = [-1, 0, 1, 0]$
- Error in this case is 0 – so $\boldsymbol{\omega}$ 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

$$1000 \times (\boldsymbol{x}_i, \boldsymbol{y} = 0) = [-1, 1, 0, 0] \quad \text{for} \quad i = 1 \ldots 1000$$
$$1 \times (\boldsymbol{x}_{1001}, \boldsymbol{y} = 1) = [0, 0, 3, 1]$$

- Now consider $\boldsymbol{\omega} = [-1, 7, 1, 0]$
- Error in this case is 1 – so $\boldsymbol{\omega}$ does not minimize 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: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

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

Perceptron: Separability and Margin

- Given an training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t)$, define:
  - $\bar{\mathcal{Y}}_t = \mathcal{Y} - \{\boldsymbol{y}_t\}$
  - i.e., $\bar{\mathcal{Y}}_t$ is the set of incorrect labels for $\boldsymbol{x}_t$
- A training set $\mathcal{T}$ is separable with margin $\gamma > 0$ if there exists a vector $\mathbf{u}$ with $\|\mathbf{u}\| = 1$ such that:

$$\mathbf{u} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{u} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

  for all $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$ and $||\mathbf{u}|| = \sqrt{\sum_j \mathbf{u}_j^2}$
- **Assumption**: the training set is separable with margin γ

Perceptron: Main Theorem

- **Theorem**: For any training set separable with a margin of γ, the following holds for the perceptron algorithm:

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

  where $R \geq ||\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')||$ for all $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$ and $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$
- 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: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

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

- $\boldsymbol{\omega}^{(k-1)}$ are the weights before k^{th} mistake
- Suppose k^{th} mistake made at the t^{th} example, $(\boldsymbol{x}_t, \boldsymbol{y}_t)$
- $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \boldsymbol{\omega}^{(k-1)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$
- $\boldsymbol{y}' \neq \boldsymbol{y}_t$
- $\boldsymbol{\omega}^{(k)} = \boldsymbol{\omega}^{(k-1)} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$

Perceptron Learning Algorithm (for handout)

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

$\boldsymbol{\omega}^{(0)} = 0;\ i = 0$
for $n : 1..N$
  for $t : 1..T$
    Let $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \boldsymbol{\omega}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$
    if $\boldsymbol{y}' \neq \boldsymbol{y}_t$
      $\boldsymbol{\omega}^{(i+1)} = \boldsymbol{\omega}^{(i)} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$
      $i = i + 1$
return $\boldsymbol{\omega}^i$

- $\boldsymbol{\omega}^{(k-1)}$ are the weights before k^{th} mistake
- Suppose k^{th} mistake made at the t^{th} example, $(\boldsymbol{x}_t, \boldsymbol{y}_t)$
- $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \boldsymbol{\omega}^{(k-1)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$
- $\boldsymbol{y}' \neq \boldsymbol{y}_t$
- $\boldsymbol{\omega}^{(k)} = \boldsymbol{\omega}^{(k-1)} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$

- Now: $\mathbf{u} \cdot \boldsymbol{\omega}^{(k)} = \mathbf{u} \cdot \boldsymbol{\omega}^{(k-1)} + \mathbf{u} \cdot (\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')) \geq \mathbf{u} \cdot \boldsymbol{\omega}^{(k-1)} + \gamma$
- Now: $\boldsymbol{\omega}^{(0)} = 0$ and $\mathbf{u} \cdot \boldsymbol{\omega}^{(0)} = 0$, by induction on k, $\mathbf{u} \cdot \boldsymbol{\omega}^{(k)} \geq k\gamma$
- Now: since $\mathbf{u} \cdot \boldsymbol{\omega}^{(k)} \leq ||\mathbf{u}|| \times ||\boldsymbol{\omega}^{(k)}||$ and $||\mathbf{u}|| = 1$ then $||\boldsymbol{\omega}^{(k)}|| \geq k\gamma$
- Now:

$$
\begin{aligned}
||\boldsymbol{\omega}^{(k)}||^2 &= ||\boldsymbol{\omega}^{(k-1)}||^2 + ||\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')||^2 + 2\boldsymbol{\omega}^{(k-1)} \cdot (\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')) \\
||\boldsymbol{\omega}^{(k)}||^2 &\leq ||\boldsymbol{\omega}^{(k-1)}||^2 + R^2 \\
&\quad \text{(since } R \geq ||\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')|| \\
&\quad \text{and } \boldsymbol{\omega}^{(k-1)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega}^{(k-1)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \leq 0)
\end{aligned}
$$

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 $||\omega^{(0)}||^2 = 0$
- Therefore,
- and solving for k
- Therefore the number of errors is bounded!

Perceptron Learning Algorithm (for handout)

- 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 $\boldsymbol{\omega}$ in a finite amount of time
- Perceptron is an example of an Online Learning Algorithm
  - $\boldsymbol{\omega}$ is updated based on a single training instance in isolation

$$\boldsymbol{\omega}^{(i+1)} = \boldsymbol{\omega}^{(i)} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$$

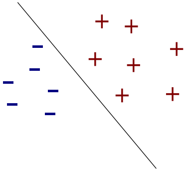
Averaged Perceptron

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

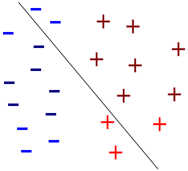
$\boldsymbol{\omega}^{(0)} = 0$; $i = 0$
for $n : 1..N$
    for $t : 1..T$
        Let $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \boldsymbol{\omega}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$
        if $\boldsymbol{y}' \neq \boldsymbol{y}_t$
            $\boldsymbol{\omega}^{(i+1)} = \boldsymbol{\omega}^{(i)} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$
        else
            $\boldsymbol{\omega}^{(i+1)} = \boldsymbol{\omega}^{(i)}$
        $i = i + 1$
return $\left(\sum_i \boldsymbol{\omega}^{(i)}\right) / (N \times T)$

Margin

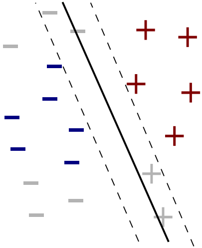
Training

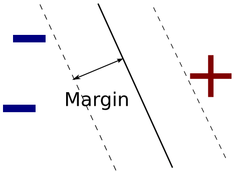


Testing



Denote the value of the margin by γ





Maximizing Margin

- For a training set $\mathcal{T}$
- Margin of a weight vector $\boldsymbol{\omega}$ is smallest γ such that

$$\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

- for every training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$, $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$

Maximizing Margin

- Intuitively maximizing margin makes sense
- More importantly, generalization error to unseen test data is proportional to the inverse of the margin

$$\epsilon \propto \frac{R^2}{\gamma^2 \times |\mathcal{T}|}$$

- **Perceptron**: we have shown that:
  - If a training set is separable by some margin, the perceptron will find a $\boldsymbol{\omega}$ that separates the data
  - However, the perceptron does not pick $\boldsymbol{\omega}$ to maximize the margin!

Support Vector Machines (SVMs)

Maximizing Margin

Let $\gamma > 0$

$$\max_{||\boldsymbol{\omega}||=1} \quad \gamma$$

such that:

$$\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

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

$$\beta(\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')) \geq \beta\gamma, \text{ for some } \beta \geq 1$$

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||\boldsymbol{\omega}||=1} \quad \gamma$$

such that:

$$\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

$$\forall(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

Change of variable: $u = \frac{w}{\gamma}$?
$||\boldsymbol{\omega}|| = 1$ iff $||\mathbf{u}|| = 1/\gamma$

Min Norm (step 1):

$$\max_{||\mathbf{u}||=1/\gamma} \quad \gamma$$

such that:

$$\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

$$\forall(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||\boldsymbol{\omega}||=1} \quad \gamma$$

such that:

$$\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

Change variables: $u = \dfrac{w}{\gamma}$?

$||\boldsymbol{\omega}|| = 1$ iff $||\mathbf{u}|| = \gamma$

Min Norm (step 2):

$$\max_{||\mathbf{u}||=1/\gamma} \quad \gamma$$

such that:

$$\gamma\mathbf{u}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \gamma\mathbf{u}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||\boldsymbol{\omega}||=1} \quad \gamma$$

such that:

$$\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

Change variables: $u = \dfrac{w}{\gamma}$?

$||\boldsymbol{\omega}|| = 1$ iff $||\mathbf{u}|| = \gamma$

Min Norm (step 3):

$$\max_{||\mathbf{u}||=1/\gamma} \quad \gamma$$

such that:

$$\mathbf{u}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{u}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

But γ is really not constrained!

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||\boldsymbol{\omega}||=1} \quad \gamma$$

such that:

$$\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

$$\forall(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

Change variables: $u = \dfrac{w}{\gamma}$?

$||\boldsymbol{\omega}|| = 1$ iff $||\mathbf{u}|| = \gamma$

Min Norm (step 4):

$$\max_{\mathbf{u}} \quad \frac{1}{||\mathbf{u}||} = \min_{\mathbf{u}} ||\mathbf{u}||$$

such that:

$$\mathbf{u}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{u}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1$$

$$\forall(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

But γ is really not constrained!

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||\boldsymbol{\omega}||=1} \quad \gamma$$

such that:

$$\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

$$\forall(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

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

Min Norm:

$$\min_{\mathbf{u}} \quad \frac{1}{2}||\mathbf{u}||^2$$

such that:

$$\mathbf{u}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{u}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1$$

$$\forall(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

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

- Intuition: Instead of fixing $||\boldsymbol{\omega}||$ we fix the margin $\gamma = 1$

Support Vector Machines

$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}} \ \frac{1}{2}||\boldsymbol{\omega}||^2$$

such that:

$$\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T} \text{ and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

- 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? (Original problem: will not satisfy the constraints!)

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

such that:

$$\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}') \geq 1-\xi_t \text{ and } \xi_t \geq 0$$

$$\forall(\boldsymbol{x}_t,\boldsymbol{y}_t)\in\mathcal{T} \text{ and } \boldsymbol{y}'\in\bar{\mathcal{Y}}_t$$

ξ_t: trade-off between margin per example and $\|\boldsymbol{\omega}\|$
Larger C = more examples correctly classified
If data is separable, optimal solution has $\xi_i = 0$, $\forall i$

Support Vector Machines

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

such that:

$$\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1 - \xi_t$$

Can we have a more compact representation of this objective function?

$$\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1 - \xi_t$$

$$\xi_t \geq 1 + \underbrace{\max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)}_{\textit{negated margin for example}}$$

Support Vector Machines

$$\xi_t \geq 1 + \underbrace{\max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)}_{\textit{negated margin for example}}$$

- If $\|\boldsymbol{\omega}\|$ classifies $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ with margin 1, penalty $\xi_t = 0$
- (Objective wants to keep ξ_t small and $\xi_t = 0$ satisfies the constraint)
- Otherwise: $\xi_t = 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)$
- (Again, because that's the minimal ξ_t that satisfies the constraint, and we want ξ_t smallest as possible)
- That means that in the end ξ_t will be:

$$\xi_t = \max\{0, 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)\}$$

(If an example is classified correctly, $\xi_t = 0$ and the second term in the max is negative.)

Support Vector Machines

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

such that:

$$\xi_t \geq 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \; \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)$$

Hinge loss equivalent

$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}} \; \mathcal{L}(\mathcal{T}; \boldsymbol{\omega}) = \arg\min_{\boldsymbol{\omega}} \; \sum_{t=1}^{|\mathcal{T}|} loss((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}) \; + \; \frac{\lambda}{2}||\boldsymbol{\omega}||^2$$

$$= \arg\min_{\boldsymbol{\omega}} \; \left(\sum_{t=1}^{|\mathcal{T}|} \max\left(0, 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \; \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)\right) \right) + \frac{\lambda}{2}||\boldsymbol{\omega}||^2$$

Summary

What we have covered

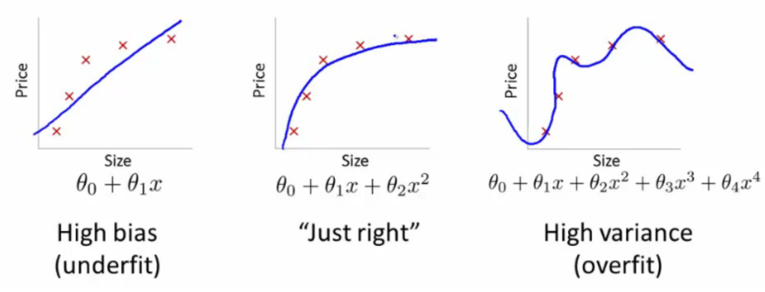
- Linear Classifiers
  - Naive Bayes
  - Logistic Regression
  - Perceptron
  - Support Vector Machines

What is next

- Regularization
- Online learning
- Non-linear classifiers

Regularization

Fit of a Model



- Two sources of error:
  - Bias error, measures how well the hypothesis class fits the space we are trying to model
  - Variance error, measures sensitivity to training set selection
  - Want to balance these two things

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 "$\phi(\boldsymbol{x}, \boldsymbol{y}) = 1$ if $\boldsymbol{x}$=fake and $\boldsymbol{y}$=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=verb

Regularization

- In practice, we regularize models to prevent overfitting

$$\arg\max_{\boldsymbol{\omega}} \; \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) - \lambda\mathcal{R}(\boldsymbol{\omega})$$

- Where $\mathcal{R}(\boldsymbol{\omega})$ is the regularization function
- λ controls how much to regularize
- Common functions
  - L2: $\mathcal{R}(\boldsymbol{\omega}) \propto \|\boldsymbol{\omega}\|_2 = \|\boldsymbol{\omega}\| = \sqrt{\sum_i \omega_i^2}$ – smaller weights desired
  - L0: $\mathcal{R}(\boldsymbol{\omega}) \propto \|\boldsymbol{\omega}\|_0 = \sum_i [[\omega_i > 0]]$ – zero weights desired
    - Non-convex
    - Approximate with L1: $\mathcal{R}(\boldsymbol{\omega}) \propto \|\boldsymbol{\omega}\|_1 = \sum_i |\omega_i|$

Logistic Regression with L2 Regularization

- Perhaps most common classifier in NLP

$$\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) - \lambda\mathcal{R}(\boldsymbol{\omega}) = \sum_{t=1}^{|\mathcal{T}|} \log\left(e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)}/Z_{\boldsymbol{x}}\right) - \frac{\lambda}{2}\|\boldsymbol{\omega}\|^2$$

- What are the new partial derivatives?

$$\frac{\partial}{\partial w_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) - \frac{\partial}{\partial w_i}\lambda\mathcal{R}(\boldsymbol{\omega})$$

- We know $\frac{\partial}{\partial w_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega})$
- Just need $\frac{\partial}{\partial w_i}\frac{\lambda}{2}\|\boldsymbol{\omega}\|^2 = \frac{\partial}{\partial w_i}\frac{\lambda}{2}\left(\sqrt{\sum_i \boldsymbol{\omega}_i^2}\right)^2 = \frac{\partial}{\partial w_i}\frac{\lambda}{2}\sum_i \boldsymbol{\omega}_i^2 = \lambda\boldsymbol{\omega}_i$

Support Vector Machines

Hinge-loss formulation: L2 regularization already happening!

$$
\begin{aligned}
\boldsymbol{\omega} &= \arg\min_{\boldsymbol{\omega}} \; \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) + \lambda\mathcal{R}(\boldsymbol{\omega}) \\
&= \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \mathit{loss}((\boldsymbol{x}_t,\boldsymbol{y}_t);\boldsymbol{\omega}) + \lambda\mathcal{R}(\boldsymbol{\omega}) \\
&= \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \max\left(0, 1 + \max_{\boldsymbol{y}\neq\boldsymbol{y}_t} \; \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)\right) + \lambda\mathcal{R}(\boldsymbol{\omega}) \\
&= \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \max\left(0, 1 + \max_{\boldsymbol{y}\neq\boldsymbol{y}_t} \; \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)\right) + \frac{\lambda}{2}\|\boldsymbol{\omega}\|^2
\end{aligned}
$$

↑ SVM optimization ↑

SVMs vs. Logistic Regression

$$\begin{aligned}\boldsymbol{\omega} &= \arg\min_{\boldsymbol{\omega}} \; \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) + \lambda\mathcal{R}(\boldsymbol{\omega}) \\ &= \arg\min_{\boldsymbol{\omega}} \; \sum_{t=1}^{|\mathcal{T}|} \mathit{loss}((\boldsymbol{x}_t, \boldsymbol{y}_t);\boldsymbol{\omega}) + \lambda\mathcal{R}(\boldsymbol{\omega})\end{aligned}$$

SVMs/hinge-loss: $\max\left(0, 1 + \max_{\boldsymbol{y}\neq\boldsymbol{y}_t}\left(\boldsymbol{\omega}\cdot\phi(\boldsymbol{x}_t,\boldsymbol{y}) - \boldsymbol{\omega}\cdot\phi(\boldsymbol{x}_t,\boldsymbol{y}_t)\right)\right)$

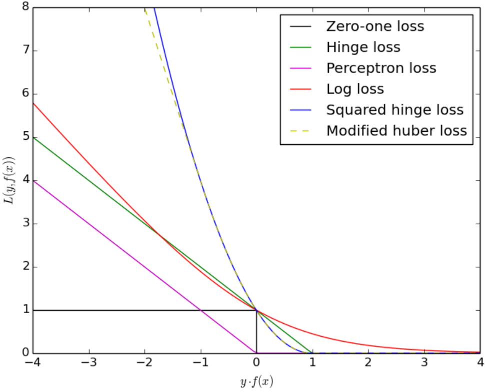
$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \max\left(0, 1 + \max_{\boldsymbol{y}\neq\boldsymbol{y}_t} \boldsymbol{\omega}\cdot\phi(\boldsymbol{x}_t,\boldsymbol{y}) - \boldsymbol{\omega}\cdot\phi(\boldsymbol{x}_t,\boldsymbol{y}_t)\right) + \frac{\lambda}{2}\|\boldsymbol{\omega}\|^2$$

Logistic Regression/log-loss: $-\log\left(e^{\boldsymbol{\omega}\cdot\phi(\boldsymbol{x}_t,\boldsymbol{y}_t)}/Z_{\boldsymbol{x}}\right)$

$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} -\log\left(e^{\boldsymbol{\omega}\cdot\phi(\boldsymbol{x}_t,\boldsymbol{y}_t)}/Z_{\boldsymbol{x}}\right) + \frac{\lambda}{2}\|\boldsymbol{\omega}\|^2$$

Generalized Linear Classifiers

$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}} \ \mathcal{L}(\mathcal{T}; \boldsymbol{\omega}) + \lambda \mathcal{R}(\boldsymbol{\omega}) = \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \mathit{loss}((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}) + \lambda \mathcal{R}(\boldsymbol{\omega})$$



Which Classifier to Use?

- Trial and error
- Training time available
- Choice of features is often more important

Online Learning

Online vs. Batch Learning

Batch($\mathcal{T}$);

- for 1 ... N
  - $\boldsymbol{\omega} \leftarrow \text{update}(\mathcal{T}; \boldsymbol{\omega})$
- return $\boldsymbol{\omega}$

E.g., SVMs, logistic regression, NB

Online($\mathcal{T}$);

- for 1 ... N
  - for $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$
    - $\boldsymbol{\omega} \leftarrow \text{update}((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega})$
  - end for
- end for
- return $\boldsymbol{\omega}$

E.g., Perceptron

$$\boldsymbol{\omega} = \boldsymbol{\omega} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})$$

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 $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \sum_{t=1}^{|\mathcal{T}|} loss((\boldsymbol{x}_t, \boldsymbol{y}_t);\boldsymbol{\omega})$
  - Set $\boldsymbol{\omega}^0 = O^m$
  - Iterate until convergence

$$\boldsymbol{\omega}^i = \boldsymbol{\omega}^{i-1} - \alpha \nabla \mathcal{L}(\mathcal{T};\boldsymbol{\omega}^{i-1}) = \boldsymbol{\omega}^{i-1} - \sum_{t=1}^{|\mathcal{T}|} \alpha \nabla loss((\boldsymbol{x}_t, \boldsymbol{y}_t);\boldsymbol{\omega}^{i-1})$$

- $\alpha > 0$ and set so that $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}^i) < \mathcal{L}(\mathcal{T};\boldsymbol{\omega}^{i-1})$
- Stochastic Gradient Descent (SGD)
  - Approximate $\nabla \mathcal{L}(\mathcal{T};\boldsymbol{\omega})$ with single $\nabla loss((\boldsymbol{x}_t, \boldsymbol{y}_t);\boldsymbol{\omega})$

Stochastic Gradient Descent

- Let $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \sum_{t=1}^{|\mathcal{T}|} loss((\boldsymbol{x}_t, \boldsymbol{y}_t);\boldsymbol{\omega})$
- Set $\boldsymbol{\omega}^0 = O^m$
- iterate until convergence
  - sample $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$ // "stochastic"
    - $\boldsymbol{\omega}^i = \boldsymbol{\omega}^{i-1} - \alpha \nabla loss((\boldsymbol{x}_t, \boldsymbol{y}_t);\boldsymbol{\omega})$
- return $\boldsymbol{\omega}$

In practice

Need to solve $\nabla loss((\boldsymbol{x}_t, \boldsymbol{y}_t);\boldsymbol{\omega})$

- Set $\boldsymbol{\omega}^0 = O^m$
- for $1 \ldots N$
  - for $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$
    - $\boldsymbol{\omega}^i = \boldsymbol{\omega}^{i-1} - \alpha \nabla loss((\boldsymbol{x}_t, \boldsymbol{y}_t);\boldsymbol{\omega})$
- return $\boldsymbol{\omega}$

Online Logistic Regression

- Stochastic Gradient Descent (SGD)
- $loss((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}) = \text{log-loss}$
- $\nabla loss((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}) = \nabla \left(-\log \left(e^{\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)} / Z_{\boldsymbol{x}_t} \right) \right)$
- From logistic regression section:

$$\nabla \left(-\log \left(e^{\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)} / Z_{\boldsymbol{x}_t} \right) \right) = -\left(\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \right)$$

- Plus regularization term (if part of model)

Online SVMs

- Stochastic Gradient Descent (SGD)
- *loss*$((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega})$ = hinge-loss

$$\nabla \textit{loss}((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}) = \nabla \left(\max\,(0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t}\ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)) \right)$$

- Subgradient is:

$$\nabla \left(\max\,(0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t}\ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)) \right)$$

$$= \begin{cases} 0, & \text{if } \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \max_{\boldsymbol{y}} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \geq 1 \\ \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t), & \text{otherwise, where } \boldsymbol{y} = \max_{\boldsymbol{y}} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \end{cases}$$

- Plus regularization term (required for SVMs)

Perceptron and Hinge-Loss

SVM subgradient update looks like perceptron update

$$\boldsymbol{\omega}^i = \boldsymbol{\omega}^{i-1} - \alpha \begin{cases} 0, & \text{if } \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \max_{\boldsymbol{y}} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \geq 1 \\ \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t), & \text{otherwise, where } \boldsymbol{y} = \max_{\boldsymbol{y}} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \end{cases}$$

Perceptron

$$\boldsymbol{\omega}^i = \boldsymbol{\omega}^{i-1} - \alpha \begin{cases} 0, & \text{if } \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \max_{\boldsymbol{y}} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \geq 0 \\ \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t), & \text{otherwise, where } \boldsymbol{y} = \max_{\boldsymbol{y}} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \end{cases}$$

where $\alpha = 1$, note $\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)$ not $\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})$ since '$-$' (descent)

Perceptron = SGD with no-margin hinge-loss

$$\max\left(0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)\right)$$

Margin Infused Relaxed Algorithm (MIRA)

Batch (SVMs):

$$\min \frac{1}{2}||\boldsymbol{\omega}||^2$$

such that:

$$\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T} \text{ and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

Online (MIRA):

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

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

- MIRA has much smaller optimizations with only $|\bar{\mathcal{Y}}_t|$ constraints

Quick Summary

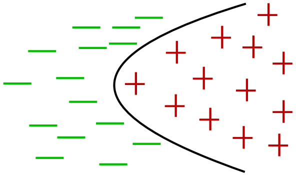
Linear Classifiers

- 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
- Decision boundary is no longer a hyperplane in the feature space
- A lot of models out there
  - K-Nearest Neighbours
  - Decision Trees
  - Neural Networks
  - **Kernels**



Kernels

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

$$K(\boldsymbol{x}_t, \boldsymbol{x}_r) \in \mathbb{R}$$

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

$$M_{t,r} = K(\boldsymbol{x}_t, \boldsymbol{x}_r)$$

- ... for any n points. Called the Gram matrix.
- Symmetric:

$$K(\boldsymbol{x}_t, \boldsymbol{x}_r) = K(\boldsymbol{x}_r, \boldsymbol{x}_t)$$

- Positive definite: for all non-zero **v** and any set of xs that define a Gram matrix:

$$\mathbf{v} M \mathbf{v}^T \geq 0$$

Kernels

- **Mercer's Theorem**: for any kernel K, there exists an ϕ, in some $\mathbb{R}^d$, such that:

$$K(\boldsymbol{x}_t, \boldsymbol{x}_r) = \boldsymbol{\phi}(\boldsymbol{x}_t) \cdot \boldsymbol{\phi}(\boldsymbol{x}_r)$$

- Since our features are over pairs $(\boldsymbol{x}, \boldsymbol{y})$, we will write kernels over pairs

$$K((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}_r, \boldsymbol{y}_r)) = \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) \cdot \boldsymbol{\phi}(\boldsymbol{x}_r, \boldsymbol{y}_r)$$

Kernel Trick: General Overview

- Define a kernel, and do not explicitly use dot product between vectors, only kernel calculations
- In some high-dimensional space, this corresponds to dot product
- In that space, the decision boundary is linear, but in the original space, we now have a non-linear decision boundary
- Let's do it for the Perceptron!

Kernel Trick – Perceptron Algorithm

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

$\boldsymbol{\omega}^{(0)} = 0$; $i = 0$
for $n : 1..N$
    for $t : 1..T$
        Let $\boldsymbol{y} = \arg\max_{\boldsymbol{y}} \boldsymbol{\omega}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})$
        if $\boldsymbol{y} \neq \boldsymbol{y}_t$
            $\boldsymbol{\omega}^{(i+1)} = \boldsymbol{\omega}^{(i)} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})$
            $i = i + 1$
return $\boldsymbol{\omega}^{i}$

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

$$\boldsymbol{\omega} = \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})]$$

Kernel Trick – Perceptron Algorithm

- We can re-write the argmax function as:
  $\boldsymbol{y}* = \arg\max_{\boldsymbol{y}^*} \boldsymbol{\omega}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}^*)$

  $=$

  $=$

  $=$

- We can then re-write the perceptron algorithm strictly with kernels

Kernel Trick – Perceptron Algorithm (for handout)

- We can re-write the argmax function as:

$$
\begin{aligned}
\boldsymbol{y}* &= \arg\max_{\boldsymbol{y}^*} \boldsymbol{\omega}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}^*) \\
&= \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})] \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}^*) \\
&= \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}^*) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}^*)] \\
&= \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [K((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}_t, \boldsymbol{y}^*)) - K((\boldsymbol{x}_t, \boldsymbol{y}), (\boldsymbol{x}, \boldsymbol{y}^*))]
\end{aligned}
$$

- We can then re-write the perceptron algorithm strictly with kernels

Kernel Trick – Perceptron Algorithm

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

1. $\forall \boldsymbol{y}, t$ set $\alpha_{\boldsymbol{y},t} = 0$
2. for n : 1..N
3. for t : 1..T
4. Let $\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t}[K((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}_t, \boldsymbol{y}^*)) - K((\boldsymbol{x}_t, \boldsymbol{y}), (\boldsymbol{x}_t, \boldsymbol{y}^*))]$
5. if $\boldsymbol{y}^* \neq \boldsymbol{y}_t$
6. $\alpha_{\boldsymbol{y}^*,t} = \alpha_{\boldsymbol{y}^*,t} + 1$

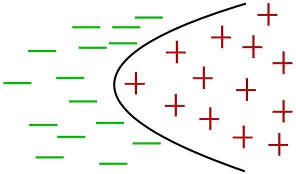
- Given a new instance $\boldsymbol{x}$

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t}[K((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}, \boldsymbol{y}^*)) - K((\boldsymbol{x}_t, \boldsymbol{y}), (\boldsymbol{x}, \boldsymbol{y}^*))]$$

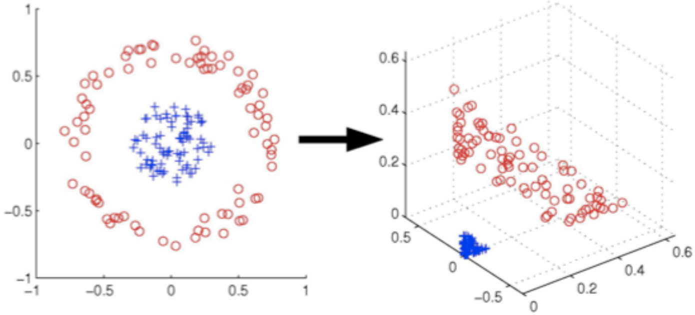
- But it seems like we have just complicated things???

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



$$\begin{array}{ccc} \Re^2 & \longrightarrow & \Re^3 \\ (x_1, x_2) & \longmapsto & (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \end{array}$$

Example: Polynomial Kernel

- $\phi(\boldsymbol{x}) \in \mathbb{R}^M$, $d \geq 2$
- $K(\boldsymbol{x}_t, \boldsymbol{x}_s) = (\phi(\boldsymbol{x}_t) \cdot \phi(\boldsymbol{x}_s) + 1)^d$
  - $O(M)$ to calculate for any d!!
- But in the original feature space (primal space)
  - Consider $d = 2$, $M = 2$, and $\phi(\boldsymbol{x}_t) = [x_{t,1}, x_{t,2}]$

$$
\begin{aligned}
(\phi(\boldsymbol{x}_t) \cdot \phi(\boldsymbol{x}_s) + 1)^2 &= ([x_{t,1}, x_{t,2}] \cdot [x_{s,1}, x_{s,2}] + 1)^2 \\
&= (x_{t,1}x_{s,1} + x_{t,2}x_{s,2} + 1)^2 \\
&= (x_{t,1}x_{s,1})^2 + (x_{t,2}x_{s,2})^2 + 2(x_{t,1}x_{s,1}) + 2(x_{t,2}x_{s,2}) \\
&\quad +2(x_{t,1}x_{t,2}x_{s,1}x_{s,2}) + (1)^2
\end{aligned}
$$

which equals:

$$
\underbrace{[(x_{t,1})^2, (x_{t,2})^2, \sqrt{2}x_{t,1}, \sqrt{2}x_{t,2}, \sqrt{2}x_{t,1}x_{t,2}, 1]}_{\textit{feature vector in high-dimensional space}} \cdot \underbrace{[(x_{s,1})^2, (x_{s,2})^2, \sqrt{2}x_{s,1}, \sqrt{2}x_{s,2}, \sqrt{2}x_{s,1}x_{s,2}, 1]}_{\textit{feature vector in high-dimensional space}}
$$

Popular Kernels

- Polynomial kernel

$$K(\boldsymbol{x}_t, \boldsymbol{x}_s) = (\phi(\boldsymbol{x}_t) \cdot \phi(\boldsymbol{x}_s) + 1)^d$$

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

$$K(\boldsymbol{x}_t, \boldsymbol{x}_s) = exp(\frac{-||\phi(\boldsymbol{x}_t) - \phi(\boldsymbol{x}_s)||^2}{2\sigma})$$

- String kernels [Lodhi et al. 2002, Collins and Duffy 2002]
- Tree kernels [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 “kernelize” algorithms to make them non-linear
- (e.g. support vector machines)

Wrap up and time for questions

Summary

Basic principles of machine learning:

- To do learning, we set up an objective function that tells the fit of the model to the data
- We optimize with respect to the model (weights, probability model, etc.)
- Can do it in a batch or online fashion

What model to use?

- One example of a model: linear classifiers
- Can kernelize these models to get non-linear classification

References and Further Reading

- A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. 1996.
  A maximum entropy approach to natural language processing. Computational Linguistics, 22(1).

- C.T. Chu, S.K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun. 2007.
  Map-Reduce for machine learning on multicore. In Advances in Neural Information Processing Systems.

- M. Collins and N. Duffy. 2002.
  New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In Proc. ACL.

- M. Collins. 2002.
  Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In Proc. EMNLP.

- K. Crammer and Y. Singer. 2001.
  On the algorithmic implementation of multiclass kernel based vector machines. JMLR.

- K. Crammer and Y. Singer. 2003.
  Ultraconservative online algorithms for multiclass problems. JMLR.

- K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. 2003.
  Online passive aggressive algorithms. In Proc. NIPS.
- K. Crammer, O. Dekel, J. Keshat, S. Shalev-Shwartz, and Y. Singer. 2006.
  Online passive aggressive algorithms. JMLR.
- Y. Freund and R.E. Schapire. 1999.
  Large margin classification using the perceptron algorithm. Machine Learning, 37(3):277–296.
- T. Joachims. 2002.
  Learning to Classify Text using Support Vector Machines. Kluwer.
- J. Lafferty, A. McCallum, and F. Pereira. 2001.
  Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proc. ICML.
- H. Lodhi, C. Saunders, J. Shawe-Taylor, and N. Cristianini. 2002.
  Classification with string kernels. Journal of Machine Learning Research.
- G. Mann, R. McDonald, M. Mohri, N. Silberman, and D. Walker. 2009.
  Efficient large-scale distributed training of conditional maximum entropy models. In Advances in Neural Information Processing Systems.
- A. McCallum, D. Freitag, and F. Pereira. 2000.

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

- R. McDonald, K. Crammer, and F. Pereira. 2005.
  Online large-margin training of dependency parsers. In Proc. ACL.
- K.R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. 2001.
  An introduction to kernel-based learning algorithms. IEEE Neural Networks, 12(2):181–201.
- J Nocedal and SJ Wright. 1999.
  Numerical optimization, volume 2. Springer New York.
- F. Sha and F. Pereira. 2003.
  Shallow parsing with conditional random fields. In Proc. HLT/NAACL, pages 213–220.
- C. Sutton and A. McCallum. 2006.
  An introduction to conditional random fields for relational learning. In L. Getoor and B. Taskar, editors, Introduction to Statistical Relational Learning. MIT Press.
- B. Taskar, C. Guestrin, and D. Koller. 2003.
  Max-margin Markov networks. In Proc. NIPS.
- B. Taskar. 2004.

Learning Structured Prediction Models: A Large Margin Approach. Ph.D. thesis, Stanford.

- I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. 2004. Support vector learning for interdependent and structured output spaces. In Proc. ICML.
- T. Zhang. 2004. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the twenty-first international conference on Machine learning.