Lecture 2: Linear Classifiers

André Martins & Vlad Niculae

Unbabel · instituto de telecomunicações · TÉCNICO LISBOA

Deep Structured Learning Course, Fall 2019

Announcements

Homework 1 is out!

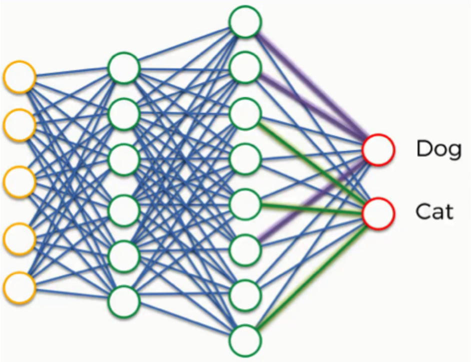
- Deadline: Friday, October 11
- Start early!!!

Why Linear Classifiers?

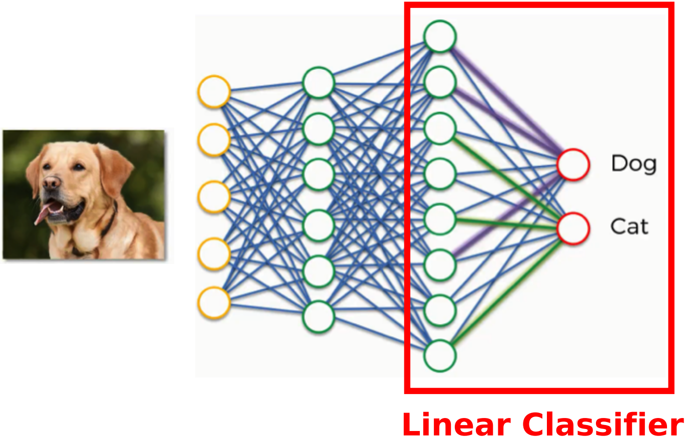
I know the course title promised "deep", but...

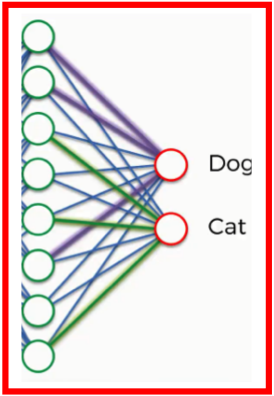
- Some underlying concepts are the same;
- The theory is much better understood;
- Linear classifiers are still widely used, fast, effective;
- Linear classifiers are **a component of neural networks**.

Linear Classifiers and Neural Networks



Linear Classifiers and Neural Networks

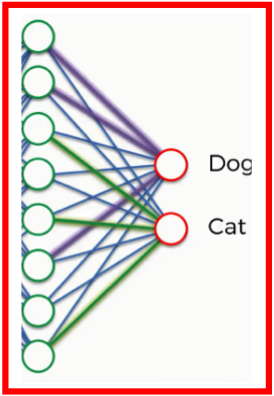




Linear Classifier

Linear Classifiers and Neural Networks

Handcrafted Features



Linear Classifier

This Unit's Roadmap

Part I.

- Binary and multi-class classification
- Linear classifiers: perceptron.

Part II.

- Naïve Bayes, logistic regression, SVMs
- Regularization and optimization, stochastic gradient descent
- Similarity-based classifiers and kernels.

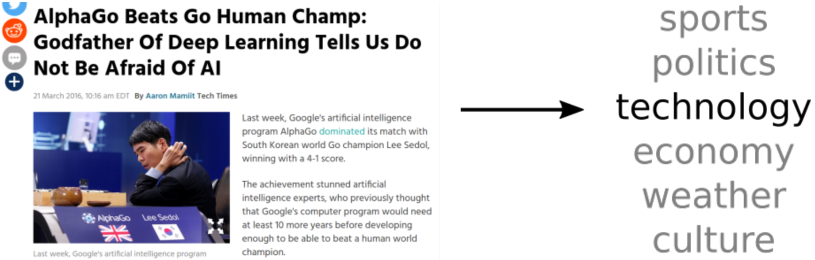
Example Tasks

Task: given an e-mail: is it SPAM or NOT-SPAM?
(binary)

Example Tasks

Task: given an e-mail: is it SPAM or NOT-SPAM? (binary)

Task: given a news article, determine its topic (politics, sports, etc.) (multi-class)



Outline

1. **Data and Feature Representation**
2. Perceptron
3. Naive Bayes
4. Logistic Regression
5. Support Vector Machines
6. Regularization
7. Non-Linear Classifiers

Disclaimer

Many of the following slides are adapted from Ryan McDonald.

Let's Start Simple

- Example 1 – sequence: $\star \diamond \circ$; label: -1
- Example 2 – sequence: $\star \heartsuit \triangle$; label: -1
- Example 3 – sequence: $\star \triangle \spadesuit$; label: $+1$
- Example 4 – sequence: $\diamond \triangle \circ$; label: $+1$

Let's Start Simple

- Example 1 – sequence: $\star \diamond \circ$; label: -1
- Example 2 – sequence: $\star \heartsuit \triangle$; label: -1
- Example 3 – sequence: $\star \triangle \spadesuit$; label: $+1$
- Example 4 – sequence: $\diamond \triangle \circ$; label: $+1$

- New sequence: $\star \diamond \circ$; label ?

Let's Start Simple

- Example 1 – sequence: $\star \diamond \circ$; label: -1
- Example 2 – sequence: $\star \heartsuit \triangle$; label: -1
- Example 3 – sequence: $\star \triangle \spadesuit$; label: $+1$
- Example 4 – sequence: $\diamond \triangle \circ$; label: $+1$

- New sequence: $\star \diamond \circ$; label -1
- New sequence: $\star \diamond \heartsuit$; label ?

Let's Start Simple

- Example 1 – sequence: $\star \diamond \circ$; label: -1
- Example 2 – sequence: $\star \heartsuit \triangle$; label: -1
- Example 3 – sequence: $\star \triangle \spadesuit$; label: $+1$
- Example 4 – sequence: $\diamond \triangle \circ$; label: $+1$

- New sequence: $\star \diamond \circ$; label -1
- New sequence: $\star \diamond \heartsuit$; label -1
- New sequence: $\star \triangle \circ$; label ?

Let's Start Simple

- Example 1 – sequence: $\star \diamond \circ$; label: -1
- Example 2 – sequence: $\star \heartsuit \triangle$; label: -1
- Example 3 – sequence: $\star \triangle \spadesuit$; label: $+1$
- Example 4 – sequence: $\diamond \triangle \circ$; label: $+1$

- New sequence: $\star \diamond \circ$; label -1
- New sequence: $\star \diamond \heartsuit$; label -1
- New sequence: $\star \triangle \circ$; label ?

Why can we do this?

Let's Start Simple: Machine Learning

- Example 1 – sequence: $\star \diamond \circ$; label: -1
- Example 2 – sequence: $\star \heartsuit \triangle$; label: -1
- Example 3 – sequence: $\star \triangle \spadesuit$; label: $+1$
- Example 4 – sequence: $\diamond \triangle \circ$; label: $+1$

- New sequence: $\star \diamond \heartsuit$; label -1

Label -1 **Label** $+1$

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

$$P(-1|\diamond) = \frac{\text{count}(\diamond \text{ and } -1)}{\text{count}(\diamond)} = \frac{1}{2} = 0.5 \text{ vs. } P(+1|\diamond) = \frac{\text{count}(\diamond \text{ and } +1)}{\text{count}(\diamond)} = \frac{1}{2} = 0.5$$

$$P(-1|\heartsuit) = \frac{\text{count}(\heartsuit \text{ and } -1)}{\text{count}(\heartsuit)} = \frac{1}{1} = 1.0 \text{ vs. } P(+1|\heartsuit) = \frac{\text{count}(\heartsuit \text{ and } +1)}{\text{count}(\heartsuit)} = \frac{0}{1} = 0.0$$

Let's Start Simple: Machine Learning

- Example 1 – sequence: $\star \diamond \circ$; label: -1
- Example 2 – sequence: $\star \heartsuit \triangle$; label: -1
- Example 3 – sequence: $\star \triangle \spadesuit$; label: $+1$
- Example 4 – sequence: $\diamond \triangle \circ$; label: $+1$

- New sequence: $\star \triangle \circ$; label ?

Label -1 **Label $+1$**

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

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

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

Machine Learning

1. Define a model/distribution of interest
2. Make some assumptions if needed
3. Fit the model to the data

Machine Learning

1. Define a model/distribution of interest
2. Make some assumptions if needed
3. Fit the model to the data

- Model: $P(\text{label}|\text{sequence}) = P(\text{label}|\text{symbol}_1, \ldots \text{symbol}_n)$
  - Prediction for new sequence $= \arg\max_{\text{label}} P(\text{label}|\text{sequence})$

- Assumption (naive Bayes—more later):

$$P(\text{symbol}_1, \ldots, \text{symbol}_n|\text{label}) = \prod_{i=1}^{n} P(\text{symbol}_i|\text{label})$$

- Fit the model to the data: count!! (simple probabilistic modeling)

Some Notation: Inputs and Outputs

- Input $\boldsymbol{x} \in \mathcal{X}$
  - e.g., a news article, a sentence, an image, ...
- Output $\boldsymbol{y} \in \mathcal{Y}$
  - e.g., fake/not fake, a topic, a parse tree, an image segmentation

- Input/Output pair: $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$
  - e.g., a **news article** together with a **topic**
  - e.g., a **sentence** together with a **parse tree**
  - e.g., an **image** partitioned into **segmentation regions**

Supervised Machine Learning

- We are given a **labeled dataset** of input/output pairs:

$$\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N} \subseteq \mathcal{X} \times \mathcal{Y}$$

- **Goal:** use it to learn a **classifier** $h : \mathcal{X} \rightarrow \mathcal{Y}$ that generalizes well to arbitrary inputs.
- At test time, given $\boldsymbol{x} \in \mathcal{X}$, we predict

$$\widehat{\boldsymbol{y}} = h(\boldsymbol{x}).$$

- Hopefully, $\widehat{\boldsymbol{y}} \approx \boldsymbol{y}$ most of the time.

Things can go by different names depending on what $\mathcal{Y}$ is...

Regression

Deals with **continuous** output variables:

- **Regression:** $\mathcal{Y} = \mathbb{R}$
  - e.g., given a news article, how much time a user will spend reading it?
- **Multivariate regression:** $\mathcal{Y} = \mathbb{R}^K$
  - e.g., predict the X-Y coordinates in an image where the user will click

Classification

Deals with **discrete** output variables:

- **Binary classification:** $\mathcal{Y} = \{\pm 1\}$
  - e.g., fake news detection

- **Multi-class classification:** $\mathcal{Y} = \{1, 2, \ldots, K\}$
  - e.g., topic classification

- **Structured classification:** $\mathcal{Y}$ exponentially large and structured
  - e.g., machine translation, caption generation, image segmentation

Later in this course, we'll cover **structured classification**

... but first, binary and multi-class classification.

Feature Representations

Feature engineering is an important step in linear classifiers:

- Bag-of-words features for text, also lemmas, parts-of-speech, ...
- SIFT features and wavelet representations in computer vision
- Other categorical, Boolean, and continuous features

Feature Representations

We need to represent information about $\boldsymbol{x}$

Typical approach: define a feature map $\boldsymbol{\psi} : \mathcal{X} \rightarrow \mathbb{R}^D$

- $\boldsymbol{\psi}(\boldsymbol{x})$ is a feature vector representing object $\boldsymbol{x}$.

Example: $\boldsymbol{x} =$"Buy a time sh4re t0day!"

$$\boldsymbol{\psi}(\boldsymbol{x}) = [\qquad\qquad\qquad\qquad\qquad]$$

Feature Representations

We need to represent information about $\boldsymbol{x}$

Typical approach: define a feature map $\boldsymbol{\psi} : \mathcal{X} \rightarrow \mathbb{R}^D$

- $\boldsymbol{\psi}(\boldsymbol{x})$ is a feature vector representing object $\boldsymbol{x}$.

Example: $\boldsymbol{x} =$"Buy a time sh4re t0day!"

$$\boldsymbol{\psi}(\boldsymbol{x}) = [5, 23, \qquad\qquad\qquad]$$

- Counts (e.g.: number of words, number of characters)

Feature Representations

We need to represent information about $\boldsymbol{x}$

Typical approach: define a feature map $\boldsymbol{\psi} : \mathcal{X} \to \mathbb{R}^D$

- $\boldsymbol{\psi}(\boldsymbol{x})$ is a feature vector representing object $\boldsymbol{x}$.

Example: $\boldsymbol{x} =$"Buy a time sh4re t0day!"

$$\boldsymbol{\psi}(\boldsymbol{x}) = [5, 23, 4.6, \qquad\qquad]$$

- Counts (e.g.: number of words, number of characters)
- Continuous (e.g.: average word length)

Feature Representations

We need to represent information about $\boldsymbol{x}$

Typical approach: define a feature map $\boldsymbol{\psi} : \mathcal{X} \to \mathbb{R}^D$

- $\boldsymbol{\psi}(\boldsymbol{x})$ is a feature vector representing object $\boldsymbol{x}$.

Example: $\boldsymbol{x} =$"Buy a time sh4re t0day!"

$$\boldsymbol{\psi}(\boldsymbol{x}) = [5, 23, 4.6, 1, \qquad\qquad]$$

- Counts (e.g.: number of words, number of characters)
- Continuous (e.g.: average word length)
- Binary (e.g.: presence of digits inside words)

Feature Representations

We need to represent information about $\boldsymbol{x}$

Typical approach: define a feature map $\boldsymbol{\psi} : \mathcal{X} \rightarrow \mathbb{R}^D$

- $\boldsymbol{\psi}(\boldsymbol{x})$ is a feature vector representing object $\boldsymbol{x}$.

Example: $\boldsymbol{x} =$"Buy a time sh4re t0day!"

$$\boldsymbol{\psi}(\boldsymbol{x}) = [5, 23, 4.6, 1, 0, 1, 0, 0]$$

- Counts (e.g.: number of words, number of characters)
- Continuous (e.g.: average word length)
- Binary (e.g.: presence of digits inside words)
- (Coded) categorical (e.g., question/exclamation/statement/none)

Binary Classification Teaser

$$\psi(x) = [5, 23, 4.6, 1, 0, 1, 0, 0]$$

Is it spam? $\mathcal{Y} = \{-1, +1\}$. We want a prediction rule $\widehat{y} = h(x)$.

In this example the true $y =$

Binary Classification Teaser

$$\psi(x) = [5, 23, 4.6, 1, 0, 1, 0, 0]$$

Is it spam? $\mathcal{Y} = \{-1, +1\}$. We want a prediction rule $\widehat{y} = h(x)$.

In this example the true $y = +1$.

Linear classifier: $h_w(x) = \text{sign}(w \cdot \psi(x))$.

Binary Classification Teaser

$$\psi(x) = [5, 23, 4.6, 1, 0, 1, 0, 0]$$

Is it spam? $\mathcal{Y} = \{-1, +1\}$. We want a prediction rule $\widehat{y} = h(x)$.

In this example the true $y = +1$.

Linear classifier: $h_w(x) = \text{sign}(w \cdot \psi(x))$.

For example:

$$w = [0, 0, -0.5, 10, 0, 2, 0, -1]$$

w is a vector in $\mathbb{R}^D$, w_i is the weight of feature i.

Binary Classification Teaser

$$\boldsymbol{\psi}(\boldsymbol{x}) = [5, 23, 4.6, 1, 0, 1, 0, 0]$$

Is it spam? $\mathcal{Y} = \{-1, +1\}$. We want a prediction rule $\widehat{\boldsymbol{y}} = h(\boldsymbol{x})$.

In this example the true $\boldsymbol{y} = +1$.

Linear classifier: $h_{\boldsymbol{w}}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x}))$.

For example:

$$\boldsymbol{w} = [0, 0, -0.5, 10, 0, 2, 0, -1]$$

$\boldsymbol{w}$ is a vector in $\mathbb{R}^D$, w_i is the weight of feature i.

$$z = \boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x}) = \sum_j w_j \psi_j(\boldsymbol{x}) = 5 \cdot 0 + 23 \cdot 0 + 4.6 \cdot -0.5 + \cdots = 9.7 > 0$$

A positive weight for feature i means the higher the feature, the more the object looks like it should be labeled $+1$.

Think of z as the *score* of the positive class.

How do we learn the weights? (teaser)

Think of the score z.

A few example criteria we will revisit later.

- Make $z > 0$ if $\boldsymbol{y} = +1$, $z < 0$ otherwise. (perceptron)

How do we learn the weights? (teaser)

Think of the score z.

A few example criteria we will revisit later.

- Make $z > 0$ if $\boldsymbol{y} = +1$, $z < 0$ otherwise. (perceptron)
- Make $z > 1$ if $\boldsymbol{y} = +1$, $z < -1$ otherwise. (SVM)

How do we learn the weights? (teaser)

Think of the score z.

A few example criteria we will revisit later.

- Make $z > 0$ if $\boldsymbol{y} = +1$, $z < 0$ otherwise. (perceptron)
- Make $z > 1$ if $\boldsymbol{y} = +1$, $z < -1$ otherwise. (SVM)
- $P(\hat{\boldsymbol{y}} = +1 | \boldsymbol{x}) \propto \exp(z)$; maximize $P(\hat{\boldsymbol{y}} = \boldsymbol{y} | \boldsymbol{x})$. (logistic regression)

From binary to multiclass

Linear binary classifier: $h_{\boldsymbol{w}}(\boldsymbol{x}) = \text{sign}\left(\underbrace{\boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x})}_{z \in \mathbb{R}}\right); \quad \boldsymbol{w} \in \mathbb{R}^D$.

What to do when we have K classes, $\mathcal{Y} = \{1, 2, \ldots, K\}$?

From binary to multiclass

Linear binary classifier: $h_{\boldsymbol{w}}(\boldsymbol{x}) = \text{sign}\,(\underbrace{\boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x})}_{z \in \mathbb{R}}); \quad \boldsymbol{w} \in \mathbb{R}^D$.

What to do when we have K classes, $\mathcal{Y} = \{1, 2, \ldots, K\}$?

Idea: Compute a score z_k for each class, select the winner!

From binary to multiclass

Linear binary classifier: $h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathsf{sign}\left(\underbrace{\boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x})}_{z \in \mathbb{R}}\right); \quad \boldsymbol{w} \in \mathbb{R}^D.$

What to do when we have K classes, $\mathcal{Y} = \{1, 2, \ldots, K\}$?

Idea: Compute a score z_k for each class, select the winner!

$$\boldsymbol{z} \in \mathbb{R}^K : \quad h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} z_{\boldsymbol{y}}$$

From binary to multiclass

Linear binary classifier: $h_{\boldsymbol{w}}(\boldsymbol{x}) = \text{sign}\left(\underbrace{\boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x})}_{z \in \mathbb{R}}\right); \quad \boldsymbol{w} \in \mathbb{R}^D.$

What to do when we have K classes, $\mathcal{Y} = \{1, 2, \ldots, K\}$?

Idea: Compute a score z_k for each class, select the winner!

$$\boldsymbol{z} \in \mathbb{R}^K : \quad h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} z_{\boldsymbol{y}}$$

A different linear model for each class:

$$\mathbf{W} = \begin{bmatrix} -\boldsymbol{w}_1- \\ \ldots \\ -\boldsymbol{w}_K- \end{bmatrix} \in \mathbb{R}^{K \times D}, \quad \boldsymbol{z} = \mathbf{W}\boldsymbol{\psi}(\boldsymbol{x}) = [\boldsymbol{w}_1 \cdot \boldsymbol{\psi}(\boldsymbol{x}), \ldots, \boldsymbol{w}_K \cdot \boldsymbol{\psi}(\boldsymbol{x})].$$

From binary to multiclass

Linear binary classifier: $h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathsf{sign}\left(\underbrace{\boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x})}_{z \in \mathbb{R}}\right); \quad \boldsymbol{w} \in \mathbb{R}^D.$

What to do when we have K classes, $\mathcal{Y} = \{1, 2, \ldots, K\}$?

Idea: Compute a score z_k for each class, select the winner!

$$\boldsymbol{z} \in \mathbb{R}^K : \quad h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} z_{\boldsymbol{y}}$$

A different linear model for each class:

$$\mathbf{W} = \begin{bmatrix} -\boldsymbol{w}_1- \\ \ldots \\ -\boldsymbol{w}_K- \end{bmatrix} \in \mathbb{R}^{K \times D}, \quad \boldsymbol{z} = \mathbf{W}\boldsymbol{\psi}(\boldsymbol{x}) = [\boldsymbol{w}_1 \cdot \boldsymbol{\psi}(\boldsymbol{x}), \ldots, \boldsymbol{w}_K \cdot \boldsymbol{\psi}(\boldsymbol{x})].$$

The binary classifier before is a special case:

$$\mathbf{W} = \begin{bmatrix} -\mathbf{0}- \\ -\boldsymbol{w}- \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.5 & 10 & 0 & 2 & 0 & -1 \end{bmatrix} \quad \boldsymbol{z} = [0, z]$$

What about the bias?

You may be used to seeing classifiers (or neural network layers) written as

$$\boldsymbol{z} = \mathbf{W}\psi(x) + \boldsymbol{b}.$$

Adding a "constant feature" of 1 allows the bias to be "absorbed" into $\mathbf{W}$.

What about the bias?

You may be used to seeing classifiers (or neural network layers) written as

$$\boldsymbol{z} = \mathbf{W}\psi(x) + \boldsymbol{b}.$$

Adding a "constant feature" of 1 allows the bias to be "absorbed" into $\mathbf{W}$.
Define $\tilde{\psi}(x) = [1, \psi(x)]$ and $\widetilde{\mathbf{W}} = [\boldsymbol{b}, \mathbf{W}]$. Multiplication reveals

$$\widetilde{\mathbf{W}}\tilde{\psi}(x) = \mathbf{W}\psi(x) + \boldsymbol{b}$$

How do we learn the weights? (teaser)

Think of the score vector $\boldsymbol{z}$.

A few example criteria we will revisit later.

- Make $z_{\boldsymbol{y}} > z_{\boldsymbol{y}'}$ (perceptron)

How do we learn the weights? (teaser)

Think of the score vector $\boldsymbol{z}$.

A few example criteria we will revisit later.

- Make $z_{\boldsymbol{y}} > z_{\boldsymbol{y}'}$ (perceptron)
- Make $z_{\boldsymbol{y}} > 1 + z_{\boldsymbol{y}'}$ (SVM)

How do we learn the weights? (teaser)

Think of the score vector $\boldsymbol{z}$.

A few example criteria we will revisit later.

- Make $z_{\boldsymbol{y}} > z_{\boldsymbol{y}'}$ (perceptron)
- Make $z_{\boldsymbol{y}} > 1 + z_{\boldsymbol{y}'}$ (SVM)
- $P(\hat{\boldsymbol{y}} = \boldsymbol{y}|\boldsymbol{x}) \propto \exp(z_{\boldsymbol{y}})$; maximize $P(\hat{\boldsymbol{y}} = \boldsymbol{y}|\boldsymbol{x})$ (logistic regression)

Feature Representations: Joint Feature Mappings

For multi-class/structured classification, a joint feature map $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^D$ is sometimes more convenient

- $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ instead of $\boldsymbol{\psi}(\boldsymbol{x})$

Each feature now represents a joint property of the input $\boldsymbol{x}$ and the candidate output $\boldsymbol{y}$.

We'll use this notation from now on.

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

To recover multi-class classifier from before:

$$h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} [\mathbf{W}\psi(\boldsymbol{x})]_{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \psi(x)$$

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

To recover multi-class classifier from before:

$$h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} [\mathbf{W}\psi(\boldsymbol{x})]_{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \psi(x)$$

Consider one-hot label representations $\boldsymbol{e}_{\boldsymbol{y}} := [0, \dots, 0, 1, 0, \dots, 0]$

Outer product $\boldsymbol{e}_{\boldsymbol{y}} \otimes \boldsymbol{\psi}(\boldsymbol{x}) = \begin{bmatrix} -\mathbf{0}- \\ \dots \\ -\psi(\boldsymbol{x})- \\ \dots \\ -\mathbf{0}- \end{bmatrix} \in \mathbb{R}^{K \times D}$ (same shape as $\mathbf{W}$!)

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

To recover multi-class classifier from before:

$$h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} [\mathbf{W}\psi(\boldsymbol{x})]_{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \psi(x)$$

Consider one-hot label representations $\boldsymbol{e}_{\boldsymbol{y}} := [0, \ldots, 0, 1, 0, \ldots, 0]$

Outer product $\boldsymbol{e}_{\boldsymbol{y}} \otimes \psi(\boldsymbol{x}) = \begin{bmatrix} -\mathbf{0}- \\ \ldots \\ -\psi(\boldsymbol{x})- \\ \ldots \\ -\mathbf{0}- \end{bmatrix} \in \mathbb{R}^{K \times D}$ (same shape as $\mathbf{W}$!)

Let $\phi(\boldsymbol{x}, \boldsymbol{y}) = \text{vec}\left(\boldsymbol{e}_{\boldsymbol{y}} \otimes \psi(\boldsymbol{x})\right)$, and $\boldsymbol{w} = \text{vec}(\mathbf{W})$.

Then, $\boldsymbol{w} \cdot \phi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{w}_{\boldsymbol{y}} \cdot \psi(\boldsymbol{x}) = z_{\boldsymbol{y}}$!

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

To recover multi-class classifier from before:

$$h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} [\mathbf{W}\psi(\boldsymbol{x})]_{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \psi(x)$$

Consider one-hot label representations $\boldsymbol{e}_{\boldsymbol{y}} := [0, \ldots, 0, 1, 0, \ldots, 0]$

Outer product $\boldsymbol{e}_{\boldsymbol{y}} \otimes \psi(\boldsymbol{x}) = \begin{bmatrix} -\mathbf{0}- \\ \ldots \\ -\psi(x)- \\ \ldots \\ -\mathbf{0}- \end{bmatrix} \in \mathbb{R}^{K \times D}$ (same shape as **W**!)

Let $\phi(\boldsymbol{x}, \boldsymbol{y}) = \text{vec}\left(\boldsymbol{e}_{\boldsymbol{y}} \otimes \psi(\boldsymbol{x})\right)$, and $\boldsymbol{w} = \text{vec}(\mathbf{W})$.

Then, $\boldsymbol{w} \cdot \phi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{w}_{\boldsymbol{y}} \cdot \psi(\boldsymbol{x}) = z_{\boldsymbol{y}}$!

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

$\phi(\boldsymbol{x}, \boldsymbol{y})$ is more expressive (allows complex features of $\boldsymbol{y}$, allows pruning!)

Examples

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

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

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

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

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

More Examples

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

$$\phi_0(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "George"} \\ & \text{and } \boldsymbol{y} = \text{"Person"} \\ 0 & \text{otherwise} \end{cases} \qquad \phi_4(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "George"} \\ & \text{and } \boldsymbol{y} = \text{"Location"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_1(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Washington"} \\ & \text{and } \boldsymbol{y} = \text{"Person"} \\ 0 & \text{otherwise} \end{cases} \qquad \phi_5(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Washington"} \\ & \text{and } \boldsymbol{y} = \text{"Location"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_2(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Bridge"} \\ & \text{and } \boldsymbol{y} = \text{"Person"} \\ 0 & \text{otherwise} \end{cases} \qquad \phi_6(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Bridge"} \\ & \text{and } \boldsymbol{y} = \text{"Location"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_3(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "General"} \\ & \text{and } \boldsymbol{y} = \text{"Person"} \\ 0 & \text{otherwise} \end{cases} \qquad \phi_7(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "General"} \\ & \text{and } \boldsymbol{y} = \text{"Location"} \\ 0 & \text{otherwise} \end{cases}$$

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

Feature Engineering and NLP Pipelines

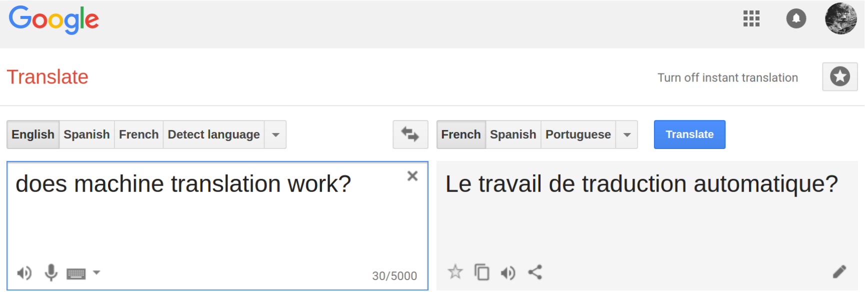
Classical NLP pipelines consist of stacking together several linear classifiers

Each classifier's predictions are used to handcraft features for other classifiers

Examples of features:

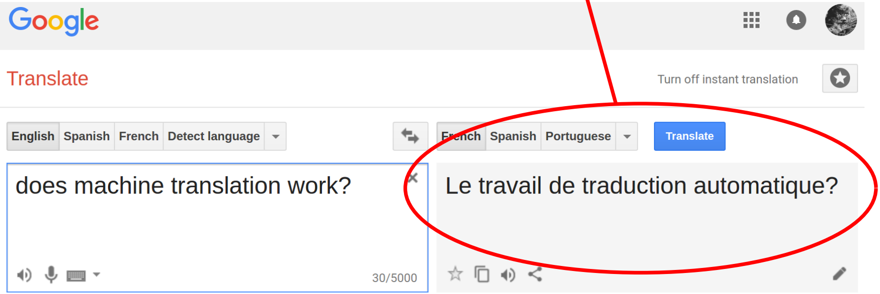
- POS tags: adjective counts for sentiment analysis
- Spell checker: misspellings counts for spam detection
- Parsing: depth of tree for readability assessment.

Example: Translation Quality Estimation

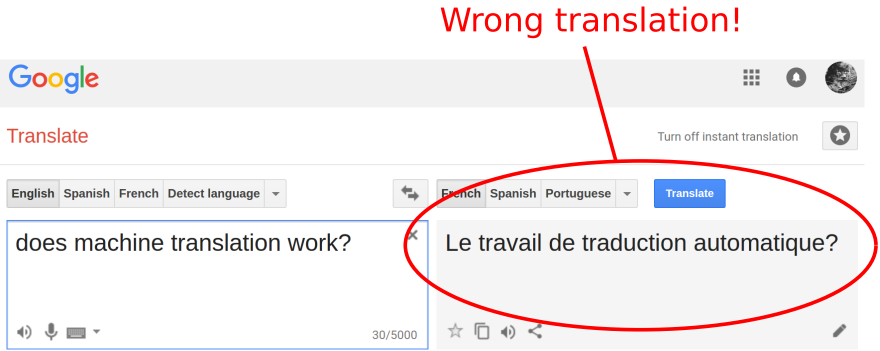


Example: Translation Quality Estimation

Wrong translation!



Example: Translation Quality Estimation



Goal: estimate the quality of a translation on the fly (without a reference)!

Example: Translation Quality Estimation

Hand-crafted features:

- no of tokens in the source/target segment
- LM probability of source/target segment and their ratio
- % of source 1–3-grams observed in 4 frequency quartiles of source corpus
- average no of translations per source word
- ratio of brackets and punctuation symbols in source & target segments
- ratio of numbers, content/non-content words in source & target segments
- ratio of nouns/verbs/etc in the source & target segments
- % of dependency relations b/w constituents in source & target segments
- diff in depth of the syntactic trees of source & target segments
- diff in no of PP/NP/VP/ADJP/ADVP/CONJP in source & target
- diff in no of person/location/organization entities in source & target
- features and global score of the SMT system
- number of distinct hypotheses in the n-best list
- 1–3-gram LM probabilities using translations in the n-best to train the LM
- average size of the target phrases
- proportion of pruned search graph nodes;
- proportion of recombined graph nodes.

Representation Learning

Feature engineering is a black art and can be very time-consuming

But it's a good way of encoding prior knowledge, and it is still widely used in practice (in particular with "small data")

Neural networks will alleviate this!

Our Setup

Let's assume a multi-class classification problem, with $|\mathcal{Y}|$ labels (classes).

Linear Classifiers

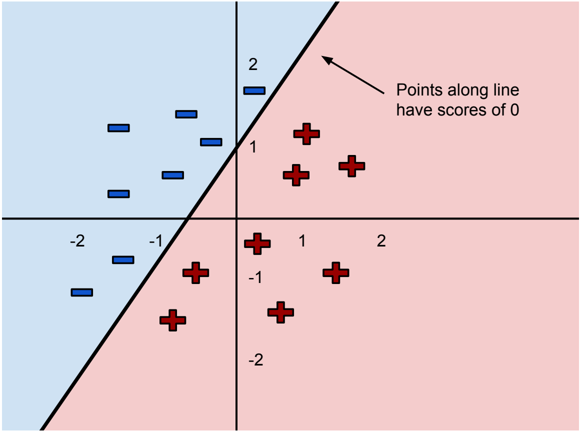
- Parametrized by a weight vector $\boldsymbol{w} \in \mathbb{R}^D$ (one weight per feature)
- The score (or probability) of a particular label is based on a linear combination of features and their weights
- At test time (known $\boldsymbol{w}$), predict the class $\widehat{\boldsymbol{y}}$ with highest score:

$$\widehat{\boldsymbol{y}} = h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$$

- At training time, different strategies to learn $\boldsymbol{w}$ yield different linear classifiers: perceptron, naïve Bayes, logistic regression, SVMs, ...

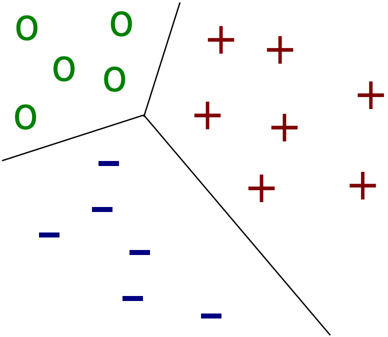
Binary Linear Classifier

A binary linear classifier $\boldsymbol{w}$ can be visualized as a line (hyperplane) separating positive and negative data points:



Multiclass Linear Classifier

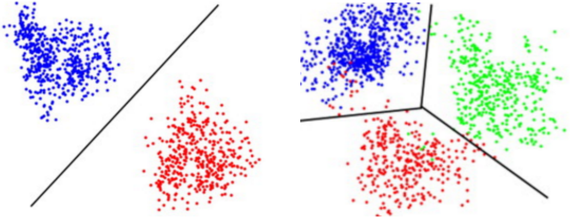
Defines regions of space.



- Prediction rule:

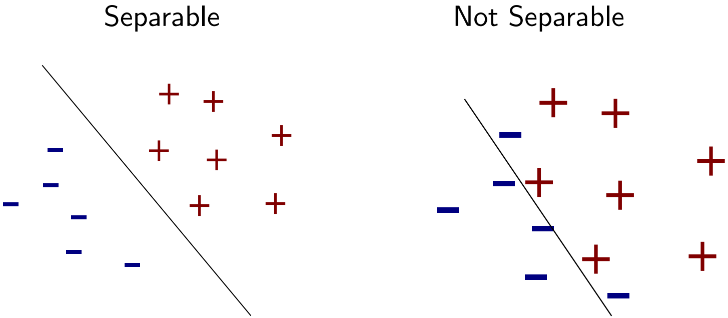
$$\widehat{\boldsymbol{y}} = h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \overbrace{\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})}^{\text{linear in } \boldsymbol{w}}$$

- The decision boundary is defined by the intersection of half spaces
- In the binary case ($|\mathcal{Y}| = 2$) this corresponds to a hyperplane classifier



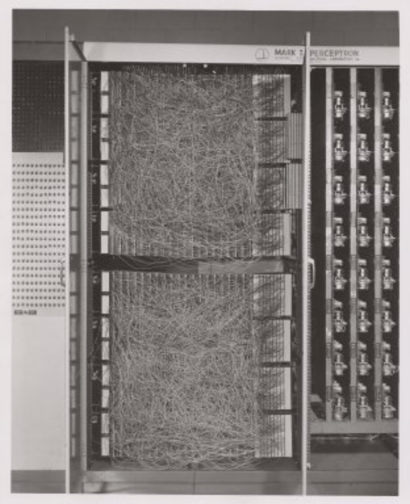
Linear Separability

- A set of points is linearly separable if there exists a $\boldsymbol{w}$ such that classification is perfect



Outline

Perceptron (Rosenblatt, 1958)



(Extracted from Wikipedia)

- Invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt
- Implemented in custom-built hardware as the "Mark 1 perceptron," designed for image recognition
- 400 photocells, randomly connected to the "neurons." Weights were encoded in potentiometers
- Weight updates during learning were performed by electric motors.

Perceptron in the News...

1958 New York Times...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

Perceptron in the News...

1958 New York Times...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.,

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

Perceptron Algorithm

- Online algorithm: process one data point at each round
  - Take $\boldsymbol{x}_i$; apply the current model to make a prediction for it
  - If prediction is correct, proceed
  - Else, correct model: add feature vector w.r.t. correct output & subtract feature vector w.r.t. predicted (wrong) output

Perceptron Algorithm

input: labeled data $\mathcal{D}$
initialize $\boldsymbol{w}^{(0)} = \mathbf{0}$
initialize $k = 0$ (number of mistakes)
repeat
  get new training example $(\boldsymbol{x}_i, \boldsymbol{y}_i)$
  predict $\widehat{\boldsymbol{y}}_i = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{w}^{(k)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y})$
  if $\widehat{\boldsymbol{y}}_i \neq \boldsymbol{y}_i$ **then**
    update $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)$
    increment k
  end if
until maximum number of epochs
output: model weights $\boldsymbol{w}$

Perceptron's Mistake Bound

A couple definitions:

- the training data is linearly separable with margin $\gamma > 0$ iff there is a weight vector $\boldsymbol{u}$ with $\|\boldsymbol{u}\| = 1$ such that

$$\boldsymbol{u} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) \geq \boldsymbol{u} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}'_i) + \gamma, \quad \forall i, \ \forall \boldsymbol{y}'_i \neq \boldsymbol{y}_i.$$

- radius of the data: $R = \max_{i, \ \boldsymbol{y}'_i \neq \boldsymbol{y}_i} \|\boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}'_i)\|$.

Perceptron's Mistake Bound

A couple definitions:

- the training data is linearly separable with margin $\gamma > 0$ iff there is a weight vector $\boldsymbol{u}$ with $\|\boldsymbol{u}\| = 1$ such that

$$\boldsymbol{u} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) \geq \boldsymbol{u} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}'_i) + \gamma, \quad \forall i, \ \forall \boldsymbol{y}'_i \neq \boldsymbol{y}_i.$$

- radius of the data: $R = \max_{i,\ \boldsymbol{y}'_i \neq \boldsymbol{y}_i} \|\boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}'_i)\|$.

Then we have the following bound of the number of mistakes:

Theorem (Novikoff (1962))

The perceptron algorithm is guaranteed to find a separating hyperplane after at most $\frac{R^2}{\gamma^2}$ mistakes.

One-Slide Proof

- **Lower bound on $\|\boldsymbol{w}^{(k+1)}\|$:**

$$\begin{aligned}\boldsymbol{u} \cdot \boldsymbol{w}^{(k+1)} &= \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + \boldsymbol{u} \cdot (\boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)) \\ &\geq \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + \gamma \\ &\geq k\gamma.\end{aligned}$$

Hence $\|\boldsymbol{w}^{(k+1)}\| = \|\boldsymbol{u}\| \cdot \|\boldsymbol{w}^{(k+1)}\| \geq \boldsymbol{u} \cdot \boldsymbol{w}^{(k+1)} \geq k\gamma$ (from CSI).

One-Slide Proof

- **Lower bound on $\|w^{(k+1)}\|$:**

$$\begin{aligned}\boldsymbol{u} \cdot \boldsymbol{w}^{(k+1)} &= \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + \boldsymbol{u} \cdot (\phi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \phi(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)) \\ &\geq \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + \gamma \\ &\geq k\gamma.\end{aligned}$$

  Hence $\|\boldsymbol{w}^{(k+1)}\| = \|\boldsymbol{u}\| \cdot \|\boldsymbol{w}^{(k+1)}\| \geq \boldsymbol{u} \cdot \boldsymbol{w}^{(k+1)} \geq k\gamma$ (from CSI).

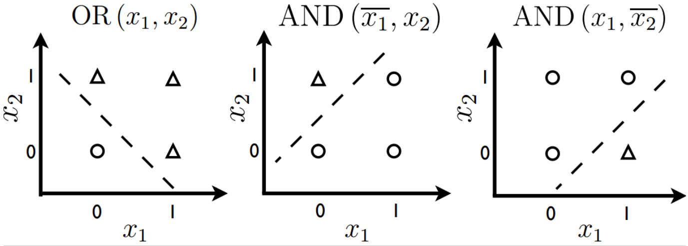
- **Upper bound on $\|w^{(k+1)}\|$:**

$$\begin{aligned}\|\boldsymbol{w}^{(k+1)}\|^2 &= \|\boldsymbol{w}^{(k)}\|^2 + \|\phi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \phi(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)\|^2 \\ &\quad + 2\boldsymbol{w}^{(k)} \cdot (\phi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \phi(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)) \\ &\leq \|\boldsymbol{w}^{(k)}\|^2 + R^2 \\ &\leq kR^2.\end{aligned}$$

Equating both sides, we get $(k\gamma)^2 \leq kR^2 \;\Rightarrow\; k \leq R^2/\gamma^2$ (QED).

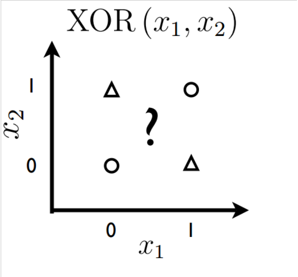
What a Simple Perceptron Can and Can't Do

- Remember: the decision boundary is linear (linear classifier)
- It **can** solve linearly separable problems (OR, AND)



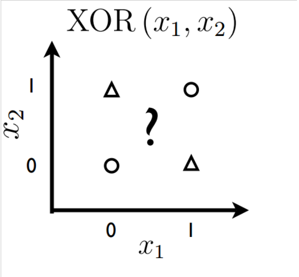
What a Simple Perceptron Can and Can't Do

- The logical XOR mapping: $h(x_1, x_2) = \text{XOR}(x_1, x_2)$.



What a Simple Perceptron Can and Can't Do

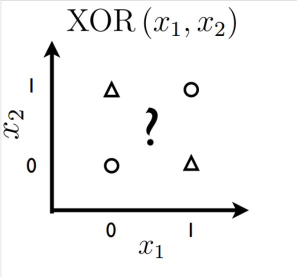
- The logical XOR mapping: $h(x_1, x_2) = \text{XOR}(x_1, x_2)$.



- Not linearly separable! The perceptron fails.
- Result attributed to Minsky and Papert (1969) but known well before.

What a Simple Perceptron Can and Can't Do

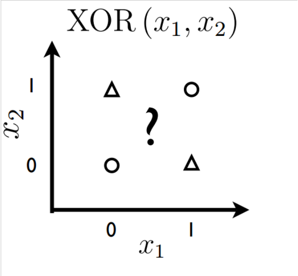
- The logical XOR mapping: $h(x_1, x_2) = \text{XOR}(x_1, x_2)$.



- Not linearly separable! The perceptron fails.
- Result attributed to Minsky and Papert (1969) but known well before.
- quiz: Our "objects" $\boldsymbol{x}$ here are pairs of bits. What is $\boldsymbol{\psi}(\boldsymbol{x})$?

What a Simple Perceptron Can and Can't Do

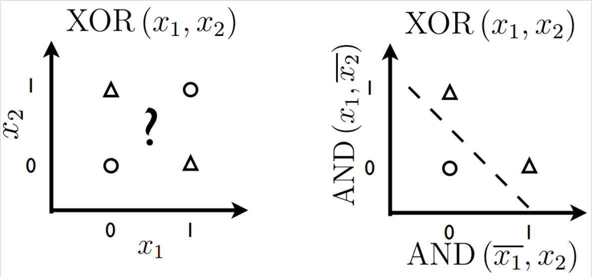
- The logical XOR mapping: $h(x_1, x_2) = \text{XOR}(x_1, x_2)$.



- Not linearly separable! The perceptron fails.
- Result attributed to Minsky and Papert (1969) but known well before.
- quiz: Our "objects" $\boldsymbol{x}$ here are pairs of bits. $\psi(\boldsymbol{x}) = [x_1, x_2]$.
  now, is there some other ψ that could "help" the perceptron?

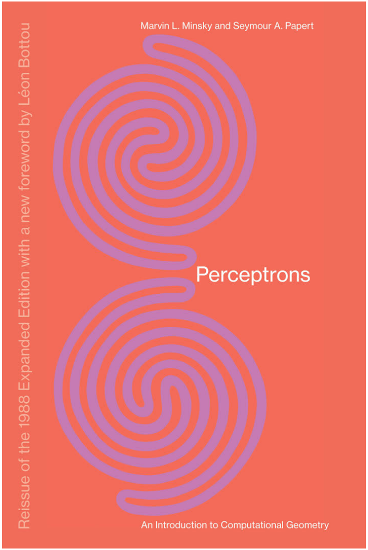
What a Simple Perceptron Can and Can't Do

- The logical XOR mapping: $h(x_1, x_2) = \mathrm{XOR}(x_1, x_2)$.



- Not linearly separable! The perceptron fails.
- Result attributed to Minsky and Papert (1969) but known well before.
- quiz: Our "objects" $\boldsymbol{x}$ here are pairs of bits. $\psi(\boldsymbol{x}) = [x_1, x_2]$.
  now, is there some other ψ that could "help" the perceptron?

Limitations of the Perceptron



Minsky and Papert (1969):

- Shows limitations of multi-layer perceptrons and fostered an "AI winter" period.

This Unit's Roadmap

Part I.

- Binary and multi-class classification
- Linear classifiers: perceptron.

Part II.

- Naïve Bayes, logistic regression, SVMs
- Regularization and optimization, stochastic gradient descent
- Similarity-based classifiers and kernels.

Outline

Probabilistic Models

- For a moment, forget linear classifiers and parameter vectors $\boldsymbol{w}$
- A *probabilistic* classifier models the conditional probability of labels $\boldsymbol{y}$ given inputs $\boldsymbol{x}$, i.e. $P(\boldsymbol{y}|\boldsymbol{x})$.

Probabilistic Models

- For a moment, forget linear classifiers and parameter vectors $\boldsymbol{w}$
- A *probabilistic* classifier models the conditional probability of labels $\boldsymbol{y}$ given inputs $\boldsymbol{x}$, i.e. $P(\boldsymbol{y}|\boldsymbol{x})$.
- Possible implementation: a function $f(\boldsymbol{x}) := [p_1, \ldots, p_K]$, where $p_c := P(\boldsymbol{y} = c|\boldsymbol{x})$.

Probabilistic Models

- For a moment, forget linear classifiers and parameter vectors $\boldsymbol{w}$
- A *probabilistic* classifier models the conditional probability of labels $\boldsymbol{y}$ given inputs $\boldsymbol{x}$, i.e. $P(\boldsymbol{y}|\boldsymbol{x})$.
- Possible implementation: a function $f(\boldsymbol{x}) := [p_1, \ldots, p_K]$, where $p_c := P(\boldsymbol{y} = c|\boldsymbol{x})$.
- If we can construct this distribution, then classification becomes:

$$\widehat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} P(\boldsymbol{y}|\boldsymbol{x})$$

But modelling $P(\boldsymbol{y}|\boldsymbol{x})$ directly is hard (or else we wouldn't need ML)!

Bayes Rule

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

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

Bayes Rule

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

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

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

Bayes Rule

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

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

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

- $P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y}) = P(\boldsymbol{x}, \boldsymbol{y})$: a joint probability

Bayes Rule

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

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

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

- $P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y}) = P(\boldsymbol{x}, \boldsymbol{y})$: a joint probability
- Above is a "generative story": Pick $\boldsymbol{y}$; then pick $\boldsymbol{x}$ given $\boldsymbol{y}$."

Bayes Rule

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

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

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

- $P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y}) = P(\boldsymbol{x}, \boldsymbol{y})$: a joint probability
- Above is a "generative story": Pick $\boldsymbol{y}$; then pick $\boldsymbol{x}$ given $\boldsymbol{y}$."
- Models that consider $P(\boldsymbol{x}, \boldsymbol{y})$ are called "generative models", because they come with a generative story.

Naive Bayes

Why is $P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y})$ better than $P(\boldsymbol{y}|\boldsymbol{x})$? Let's consider a special case.

Say input $\boldsymbol{x}$ is partitioned as $v_1, \ldots, v_L$, where $v_k \in \mathcal{V}$

Example:

- $\boldsymbol{x}$ is a document of length L
- v_k is the k^{th} token (a word)
- The set $\mathcal{V}$ is the vocabulary, e.g. $\mathcal{V} = \{$dog, cat, the, platypus, ...$\}$

$$P(\underbrace{v_1, \ldots, v_L}_{\boldsymbol{x}} | \boldsymbol{y})$$

(quiz: What data structure? How many parameters?)

Naive Bayes

Why is $P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y})$ better than $P(\boldsymbol{y}|\boldsymbol{x})$? Let's consider a special case.

Say input $\boldsymbol{x}$ is partitioned as $v_1, \ldots, v_L$, where $v_k \in \mathcal{V}$

Example:

- $\boldsymbol{x}$ is a document of length L
- v_k is the k^{th} token (a word)
- The set $\mathcal{V}$ is the vocabulary, e.g. $\mathcal{V} = \{\text{dog, cat, the, platypus, ...}\}$

Naive Bayes Assumption

(conditional independence)

$$P(\underbrace{v_1, \ldots, v_L}_{\boldsymbol{x}} | \boldsymbol{y}) = \prod_{k=1}^{L} P(v_k | \boldsymbol{y})$$

(quiz: What data structure? How many parameters?)

Multinomial Naive Bayes

$$P(\boldsymbol{x}, \boldsymbol{y}) = P(\boldsymbol{y})P(\underbrace{v_1, \ldots, v_L}_{\boldsymbol{x}}|\boldsymbol{y}) = P(\boldsymbol{y}) \prod_{k=1}^{L} P(v_k|\boldsymbol{y})$$

- All tokens are conditionally independent, given the label
- The word order doesn't matter ("bag-of-words")

Classifier that we can now implement:

$$h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}) \prod P(v_k|\boldsymbol{y})$$

Small caveat: we assumed that the document has a fixed length L.

Multinomial Naive Bayes – Arbitrary Length

Solution: introduce a distribution over document length $P(|\boldsymbol{x}|)$

- e.g. a Poisson distribution.

We get:

$$P(\boldsymbol{x}, \boldsymbol{y}) = P(\boldsymbol{y}) \underbrace{{\color{red}P(|\boldsymbol{x}|)} \prod_{k=1}^{|\boldsymbol{x}|} P(v_k | \boldsymbol{y})}_{P(\boldsymbol{x}|\boldsymbol{y})}$$

$P(|\boldsymbol{x}|)$ is constant (independent of $\boldsymbol{y}$), so nothing really changes

- the posterior $P(\boldsymbol{y}|\boldsymbol{x})$ is the same as before.

What Does This Buy Us?

$$P(\underbrace{v_1, \ldots, v_L}_{\boldsymbol{x}} |\boldsymbol{y}) = \prod_{k=1}^{L} P(v_k|\boldsymbol{y})$$

What do we gain with the Naive Bayes assumption?

What Does This Buy Us?

$$P(\underbrace{v_1, \ldots, v_L}_{\boldsymbol{x}} | \boldsymbol{y}) = \prod_{k=1}^{L} P(v_k | \boldsymbol{y})$$

What do we gain with the Naive Bayes assumption?

- A huge reduction in the number of parameters!
- Without factorization assumptions, $P(v_1, \ldots, v_L | \boldsymbol{y})$: $O(|\mathcal{V}|^L)$ parameters.
- With Naive Bayes, $O(|\mathcal{V}|)$ parameters.

What Does This Buy Us?

$$P(\underbrace{v_1, \ldots, v_L}_{\boldsymbol{x}} | \boldsymbol{y}) = \prod_{k=1}^{L} P(v_k | \boldsymbol{y})$$

What do we gain with the Naive Bayes assumption?

- A huge reduction in the number of parameters!
- Without factorization assumptions, $P(v_1, \ldots, v_L | \boldsymbol{y})$: $O(|\mathcal{V}|^L)$ parameters.
- With Naive Bayes, $O(|\mathcal{V}|)$ parameters.

Fewer parameters reduce computation, increace generalization power.
Generally: reduce overfitting but might underfit.

Naive Bayes – Learning

$$P(\boldsymbol{y})P(\underbrace{v_1, \ldots, v_L}_{\boldsymbol{x}}|\boldsymbol{y}) = P(\boldsymbol{y}) \prod_{k=1}^{L} P(v_k|\boldsymbol{y})$$

- Input: dataset $\mathcal{D} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{N}$ (examples assumed i.i.d.)
- Parameters $\boldsymbol{\Theta} = \{P(\boldsymbol{y}), P(v|\boldsymbol{y})\}$
- Objective: Maximum Likelihood Estimation (MLE): choose parameters that maximize the likelihood of observed data

$$\mathcal{L}(\boldsymbol{\Theta}; \mathcal{D}) = \prod_{t=1}^{N} P(\boldsymbol{x}_t, \boldsymbol{y}_t) = \prod_{t=1}^{N} \left(P(\boldsymbol{y}_t) \prod_{k=1}^{L} P(v_k(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right)$$

$$\widehat{\boldsymbol{\Theta}} = \arg\max_{\boldsymbol{\Theta}} \prod_{t=1}^{N} \left(P(\boldsymbol{y}_t) \prod_{k=1}^{L} P(v_k(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right)$$

Naive Bayes – Learning via MLE

For the multinomial Naive Bayes model, MLE has a closed form solution!!

It all boils down to counting and normalizing!!

(The proof is left as an exercise...)

Naive Bayes – Learning via MLE

$$\widehat{\boldsymbol{\Theta}} = \arg\max_{\boldsymbol{\Theta}} \prod_{t=1}^{N} \left(P(\boldsymbol{y}_t) \prod_{k=1}^{L} P(v_k(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right)$$

$$\widehat{P}(\boldsymbol{y}) = \frac{\sum_{t=1}^{N} [[\boldsymbol{y}_t = \boldsymbol{y}]]}{N}$$

$$\widehat{P}(v|\boldsymbol{y}) = \frac{\sum_{t=1}^{N} \sum_{k=1}^{L} [[v_k(\boldsymbol{x}_t) = v \text{ and } \boldsymbol{y}_t = \boldsymbol{y}]]}{L \sum_{t=1}^{N} [[\boldsymbol{y}_t = \boldsymbol{y}]]}$$

$[[X]]$ is 1 if property X holds, 0 otherwise (Iverson notation)
Fraction of times a feature appears in training cases of a given label

Naive Bayes Example

- Corpus of movie reviews: 7 examples for **training**

Doc	Words	Class
1	Great movie, excellent plot, renown actors	Positive
2	I had not seen a fantastic plot like this in good 5 years. Amazing!!!	Positive
3	Lovely plot, amazing cast, somehow I am in love with the bad guy	Positive
4	Bad movie with great cast, but very poor plot and unimaginative ending	Negative
5	I hate this film, it has nothing original	Negative
6	Great movie, but not...	Negative
7	Very bad movie, I have no words to express how I dislike it	Negative

Naive Bayes Example

- **Features**: adjectives (bag-of-words)

Doc	Words	Class
1	Great movie, excellent plot, renowned actors	Positive
2	I had not seen a fantastic plot like this in good 5 years. amazing !!!	Positive
3	Lovely plot, amazing cast, somehow I am in love with the bad guy	Positive
4	Bad movie with great cast, but very poor plot and unimaginative ending	Negative
5	I hate this film, it has nothing original. Really bad	Negative
6	Great movie, but not...	Negative
7	Very bad movie, I have no words to express how I dislike it	Negative

Naive Bayes Example

Relative frequency:

Priors:

$$P(\text{positive}) = \frac{\sum_{t=1}^{N}[[\boldsymbol{y}_t = \text{positive}]]}{N} = 3/7 = 0.43$$

$$P(\text{negative}) = \frac{\sum_{t=1}^{N}[[\boldsymbol{y}_t = \text{negative}]]}{N} = 4/7 = 0.57$$

Assume standard pre-processing: tokenization, lowercasing, punctuation removal (except special punctuation like !!!)

Naive Bayes Example

Likelihoods: Count adjective v in class $\boldsymbol{y}$ / adjectives in $\boldsymbol{y}$

$$\widehat{P}(v|\boldsymbol{y}) = \frac{\sum_{t=1}^{N}\sum_{k=1}^{L}[[v_k(\boldsymbol{x}_t) = v \text{ and } \boldsymbol{y}_t = \boldsymbol{y}]]}{L\sum_{t=1}^{N}[[\boldsymbol{y}_t = \boldsymbol{y}]]}$$

$$\begin{array}{ll|ll}
P(amazing|positive) & = 2/10 & P(amazing|negative) & = 0/8 \\
P(bad|positive) & = 1/10 & P(bad|negative) & = 3/8 \\
P(excellent|positive) & = 1/10 & P(excellent|negative) & = 0/8 \\
P(fantastic|positive) & = 1/10 & P(fantastic|negative) & = 0/8 \\
P(good|positive) & = 1/10 & P(good|negative) & = 0/8 \\
P(great|positive) & = 1/10 & P(great|negative) & = 2/8 \\
P(lovely|positive) & = 1/10 & P(lovely|negative) & = 0/8 \\
P(original|positive) & = 0/10 & P(original|negative) & = 1/8 \\
P(poor|positive) & = 0/10 & P(poor|negative) & = 1/8 \\
P(renowned|positive) & = 1/10 & P(renowned|negative) & = 0/8 \\
P(unimaginative|positive) & = 0/10 & P(unimaginative|negative) & = 1/8
\end{array}$$

Naive Bayes Example: Test Time

$$h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}) \prod P(v_k|\boldsymbol{y})$$

Doc	Words	Class
8	This was a fantastic story, good, lovely	???

Naive Bayes Example: Test Time

$$h(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}) \prod P(v_k|\boldsymbol{y})$$

Doc	Words	Class
8	This was a fantastic story, good, lovely	???

Final decision

$$P(positive) * P(fantastic|positive) * P(good|positive) * P(lovely|positive)$$

$$3/7 * 1/10 * 1/10 * 1/10 = 0.00043$$

$$P(negative) * P(fantastic|negative) * P(good|negative) * P(lovely|negative)$$

$$4/7 * 0/8 * 0/8 * 0/8 = 0$$

So: *sentiment = positive*

Naive Bayes Example: Test Time

Doc	Words	Class
10	Boring movie, annoying plot, unimaginative ending	???

Final decision

$$P(positive) * P(boring|positive) * P(annoying|positive) * P(unimaginative|positive)$$

$$3/7 * 0/10 * 0/10 * 0/10 = 0$$

$$P(negative) * P(boring|negative) * P(annoying|negative) * P(unimaginative|negative)$$

$$4/7 * 0/8 * 0/8 * 1/8 = 0$$

So: $sentiment = ???$

Laplace Smoothing

Add smoothing to feature counts (add 1 to every count):

$$\widehat{P}(v|\boldsymbol{y}) = \frac{\sum_{t=1}^{N} \sum_{k=1}^{L} [[v_k(\boldsymbol{x}_t) = v \text{ and } \boldsymbol{y}_t = \boldsymbol{y}]] + 1}{L \sum_{t=1}^{N} [[\boldsymbol{y}_t = \boldsymbol{y}]] + |\mathcal{V}|}$$

where $|\mathcal{V}|$ = number of distinct adjectives in training (all classes) = **12**

Interpretation: as if we inserted a dummy document containing a single word: One for each known word, one for each class label.

Doc	Words	Class
11	Boring movie, annoying plot, unimaginative ending	???

Final decision

$$P(positive) * P(boring|positive) * P(annoying|positive) * P(unimaginative|positive)$$

$$3/7 * ((0+1)/(10+12)) * ((0+1)/(10+12)) * ((0+1)/(10+12)) = 0.000040$$

$$P(negative) * P(boring|negative) * P(annoying|negative) * P(unimaginative|negative)$$

$$4/7 * ((0+1)/(8+12)) * ((0+1)/(8+12)) * ((1+1)/(8+12)) = 0.000143$$

So: *sentiment* = *negative*

Finally...

Multinomial Naive Bayes is a Linear Classifier!

One Slide Proof

- Let $b_{\boldsymbol{y}} = \log P(\boldsymbol{y})$, $\forall \boldsymbol{y} \in \mathcal{Y}$
- Let $[\boldsymbol{w}_{\boldsymbol{y}}]_v = \log P(v|\boldsymbol{y})$, $\forall \boldsymbol{y} \in \mathcal{Y}, v \in \mathcal{V}$
- Let $[\boldsymbol{\psi}(\boldsymbol{x})]_v = \sum_{k=1}^{L} [[v_k(\boldsymbol{x}) = v]]$, $\forall v \in \mathcal{V}$ (# times v occurs in $\boldsymbol{x}$)

$$\begin{aligned}
\arg\max_{\boldsymbol{y}} \; P(\boldsymbol{y}|\boldsymbol{x}) &\propto \arg\max_{\boldsymbol{y}} \left(P(\boldsymbol{y}) \prod_{k=1}^{L} P(v_k(\boldsymbol{x})|\boldsymbol{y}) \right) \\
&= \arg\max_{\boldsymbol{y}} \left(\log P(\boldsymbol{y}) + \sum_{k=1}^{L} \log P(v_k(\boldsymbol{x})|\boldsymbol{y}) \right) \\
&= \arg\max_{\boldsymbol{y}} \left(\underbrace{\log P(\boldsymbol{y})}_{b_{\boldsymbol{y}}} + \sum_{v \in \mathcal{V}} [\boldsymbol{\psi}(\boldsymbol{x})]_v \underbrace{\log P(v|\boldsymbol{y})}_{[\boldsymbol{w}_{\boldsymbol{y}}]_v} \right) \\
&= \arg\max_{\boldsymbol{y}} \left(\boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\psi}(\boldsymbol{x}) + b_{\boldsymbol{y}} \right).
\end{aligned}$$

Discriminative versus Generative

- Generative models attempt to model inputs and outputs
  - e.g., Naive Bayes = MLE of joint distribution $P(\boldsymbol{x}, \boldsymbol{y})$
  - Statistical model must explain generation of input
  - Can we sample a document from the multinomial Naive Bayes model? How?

Discriminative versus Generative

- Generative models attempt to model inputs and outputs
  - e.g., Naive Bayes = MLE of joint distribution $P(\boldsymbol{x}, \boldsymbol{y})$
  - Statistical model must explain generation of input
  - Can we sample a document from the multinomial Naive Bayes model? How?
- Occam's Razor: why model input?
- Discriminative models
  - Use loss function 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

Discriminative versus Generative

- Generative models attempt to model inputs and outputs
  - e.g., Naive Bayes = MLE of joint distribution $P(\boldsymbol{x}, \boldsymbol{y})$
  - Statistical model must explain generation of input
  - Can we sample a document from the multinomial Naive Bayes model? How?

- Occam's Razor: why model input?
- Discriminative models
  - Use loss function 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
  - They differ only on what distribution they use to set $\boldsymbol{w}$

So far

We have covered:

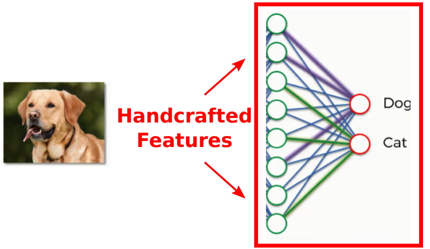
- The perceptron algorithm
- (Multinomial) Naive Bayes.

We saw that both are instances of linear classifiers.

Perceptron finds a separating hyperplane (if it exists), Naive Bayes is a generative probabilistic model

Next: a discriminative probabilistic model.

Reminder



$$\widehat{\boldsymbol{y}} = \arg\max \left(\mathbf{W}\psi(\boldsymbol{x}) + \boldsymbol{b}\right), \quad \mathbf{W} = \begin{bmatrix} \vdots \\ -\boldsymbol{w}_{\boldsymbol{y}}- \\ \vdots \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} \vdots \\ b_{\boldsymbol{y}} \\ \vdots \end{bmatrix}.$$

equivalent to

$$\widehat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \boldsymbol{w} \cdot \phi(\boldsymbol{x}, \boldsymbol{y}) \quad \text{where} \quad \boldsymbol{w} = \mathrm{vec}([\boldsymbol{b}, \mathbf{W}])$$

Outline

Logistic Regression

A linear model gives us a score for each class, $\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$.

Logistic Regression

A linear model gives us a score for each class, $\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$.

Define a conditional probability:

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

Logistic Regression

A linear model gives us a score for each class, $\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$.

Define a conditional probability:

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

Note: still a linear classifier

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

Binary Logistic Regression

Binary labels ($\mathcal{Y} = \{\pm 1\}$)

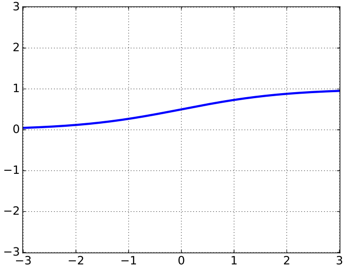
Scores: $\boldsymbol{z} = [0, \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x})]$

$$
\begin{aligned}
P(\boldsymbol{y} = +1 | \boldsymbol{x}) &= \frac{\exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}))}{1 + \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}))} \\
&= \frac{1}{1 + \exp(-\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}))} \\
&= \sigma(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x})).
\end{aligned}
$$

This is called a sigmoid transformation (more later!)

Sigmoid Transformation

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



- Widely used in neural networks
- Can be regarded as a 2D softmax
- "Squashes" a real number between 0 and 1
- The output can be interpreted as a probability
- Positive, bounded, strictly increasing

Multinomial Logistic Regression

$$P_{\boldsymbol{w}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}))}{Z_{\boldsymbol{x}}}$$

- How do we learn weights $\boldsymbol{w}$?
- Set $\boldsymbol{w}$ to maximize the conditional log-likelihood of training data:

$$\begin{aligned}\widehat{\boldsymbol{w}} &= \arg\max_{\boldsymbol{w}\in\mathbb{R}^D} \log\left(\prod_{t=1}^{N} P_{\boldsymbol{w}}(\boldsymbol{y}_t|\boldsymbol{x}_t)\right) = \arg\min_{\boldsymbol{w}\in\mathbb{R}^D} -\sum_{t=1}^{N} \log P_{\boldsymbol{w}}(\boldsymbol{y}_t|\boldsymbol{x}_t) = \\ &= \arg\min_{\boldsymbol{w}\in\mathbb{R}^D} \sum_{t=1}^{N} \left(\log \sum_{\boldsymbol{y}'_t} \exp(\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}'_t)) - \boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)\right),\end{aligned}$$

i.e., set $\boldsymbol{w}$ to assign as much probability mass as possible to the correct labels!

Logistic Regression

- This objective function is convex
- Therefore any local minimum is a global minimum
- No closed form solution, but lots of numerical techniques
  - Gradient methods (gradient descent, conjugate gradient)
  - Quasi-Newton methods (L-BFGS, ...)

Recap: Convex functions

Pro: Guarantee of a global minima ✓

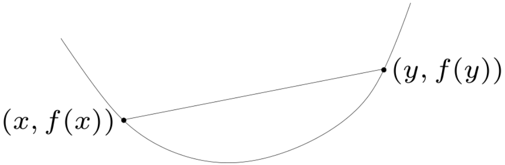


Figure: Illustration of a convex function. The line segment between any two points on the graph lies entirely above the curve.

Recap: Iterative Descent Methods

Goal: find the minimum/minimizer of $f : \mathbb{R}^d \to \mathbb{R}$

- Proceed in **small steps** in the **optimal direction** till a **stopping criterion** is met.
- **Gradient descent**: updates of the form: $x^{(k+1)} \leftarrow x^{(k)} - \eta_k \nabla f(x^{(k)})$

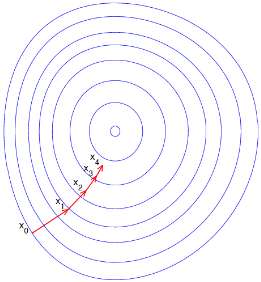


Figure: Illustration of gradient descent. The red lines correspond to steps taken in the negative gradient direction.

Gradient Descent

- Let $L(\boldsymbol{w};(\boldsymbol{x},\boldsymbol{y})) = \log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}')) - \boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})$
- This is our loss function!
- We want to find $\arg\min_{\boldsymbol{w}} \sum_{t=1}^{N} L(\boldsymbol{w};(\boldsymbol{x}_t,\boldsymbol{y}_t))$
  - Set $\boldsymbol{w}^0 = \mathbf{0}$
  - Iterate until convergence (for suitable stepsize η_k):

$$\begin{aligned} \boldsymbol{w}^{k+1} &= \boldsymbol{w}^k - \eta_k \nabla_{\boldsymbol{w}} \left(\sum_{t=1}^{N} L(\boldsymbol{w};(\boldsymbol{x}_t,\boldsymbol{y}_t))\right) \\ &= \boldsymbol{w}^k - \eta_k \sum_{t=1}^{N} \nabla_{\boldsymbol{w}} L(\boldsymbol{w};(\boldsymbol{x}_t,\boldsymbol{y}_t)) \end{aligned}$$

- $\nabla_{\boldsymbol{w}} L(\boldsymbol{w})$ is gradient of L w.r.t. $\boldsymbol{w}$
- For convex L, with minor assumptions on η_k, gradient descent will always find the optimal $\boldsymbol{w}$!

Stochastic Gradient Optimization

It turns out this works with a Monte Carlo approximation of the gradient:

- Set $\boldsymbol{w}^0 = \mathbf{0}$
- Iterate until convergence
  - Pick $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ randomly
  - Update $\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta_k \nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$
- i.e. we approximate the true gradient with a noisy, unbiased, gradient, based on a single sample
- Variants exist in-between (mini-batches)
- All guaranteed to find the optimal $\boldsymbol{w}$ (for suitable step sizes)

Computing the Gradient

- For this to work, we need to be able to compute $\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$, where

$$L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) = \log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$$

Some reminders:

1. $\nabla_{\boldsymbol{w}} \log F(\boldsymbol{w}) = \frac{1}{F(\boldsymbol{w})} \nabla_{\boldsymbol{w}} F(\boldsymbol{w})$
2. $\nabla_{\boldsymbol{w}} \exp F(\boldsymbol{w}) = \exp(F(\boldsymbol{w})) \nabla_{\boldsymbol{w}} F(\boldsymbol{w})$

Computing the Gradient

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) \quad = \quad \nabla_{\boldsymbol{w}} \left(\log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \right)$$

Computing the Gradient

$$\begin{aligned}
\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) &= \nabla_{\boldsymbol{w}} \left(\log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \right) \\
&= \nabla_{\boldsymbol{w}} \log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \nabla_{\boldsymbol{w}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}$$

Computing the Gradient

$$
\begin{aligned}
\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) &= \nabla_{\boldsymbol{w}} \left(\log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \right) \\
&= \nabla_{\boldsymbol{w}} \log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \nabla_{\boldsymbol{w}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \frac{1}{\sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}'))} \sum_{\boldsymbol{y}'} \nabla_{\boldsymbol{w}} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

Computing the Gradient

$$\begin{aligned}
\nabla_{\boldsymbol{w}} L(\boldsymbol{w};(\boldsymbol{x},\boldsymbol{y})) &= \nabla_{\boldsymbol{w}} \left(\log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}')) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}) \right) \\
&= \nabla_{\boldsymbol{w}} \log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}')) - \nabla_{\boldsymbol{w}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}) \\
&= \frac{1}{\sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}'))} \sum_{\boldsymbol{y}'} \nabla_{\boldsymbol{w}} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}')) - \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}) \\
&= \frac{1}{Z_{\boldsymbol{x}}} \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}')) \nabla_{\boldsymbol{w}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})
\end{aligned}$$

Computing the Gradient

$$\begin{aligned}
\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) &= \nabla_{\boldsymbol{w}} \left(\log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \right) \\
&= \nabla_{\boldsymbol{w}} \log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \nabla_{\boldsymbol{w}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \frac{1}{\sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}'))} \sum_{\boldsymbol{y}'} \nabla_{\boldsymbol{w}} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \frac{1}{Z_{\boldsymbol{x}}} \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) \nabla_{\boldsymbol{w}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \sum_{\boldsymbol{y}'} \frac{\exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}'))}{Z_{\boldsymbol{x}}} \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}$$

Computing the Gradient

$$\begin{aligned}
\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) &= \nabla_{\boldsymbol{w}} \left(\log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \right) \\
&= \nabla_{\boldsymbol{w}} \log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \nabla_{\boldsymbol{w}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \frac{1}{\sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}'))} \sum_{\boldsymbol{y}'} \nabla_{\boldsymbol{w}} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \frac{1}{Z_{\boldsymbol{x}}} \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')) \nabla_{\boldsymbol{w}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \sum_{\boldsymbol{y}'} \frac{\exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}'))}{Z_{\boldsymbol{x}}} \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \sum_{\boldsymbol{y}'} P_{\boldsymbol{w}}(\boldsymbol{y}' | \boldsymbol{x}) \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}$$

Computing the Gradient

$$
\begin{aligned}
\nabla_{\boldsymbol{w}} L(\boldsymbol{w};(\boldsymbol{x},\boldsymbol{y})) &= \nabla_{\boldsymbol{w}}\left(\log\sum_{\boldsymbol{y}'}\exp(\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}')) - \boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})\right) \\
&= \nabla_{\boldsymbol{w}}\log\sum_{\boldsymbol{y}'}\exp(\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}')) - \nabla_{\boldsymbol{w}}\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}) \\
&= \frac{1}{\sum_{\boldsymbol{y}'}\exp(\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}'))}\sum_{\boldsymbol{y}'}\nabla_{\boldsymbol{w}}\exp(\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}')) - \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}) \\
&= \frac{1}{Z_{\boldsymbol{x}}}\sum_{\boldsymbol{y}'}\exp(\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}'))\nabla_{\boldsymbol{w}}\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}) \\
&= \sum_{\boldsymbol{y}'}\frac{\exp(\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}'))}{Z_{\boldsymbol{x}}}\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}) \\
&= \sum_{\boldsymbol{y}'}P_{\boldsymbol{w}}(\boldsymbol{y}'|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}) \\
&= \mathbb{E}_{Y}[\boldsymbol{\phi}(\boldsymbol{x},Y)] - \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}).
\end{aligned}
$$

Logistic Regression Summary

- Define conditional probability

$$P_{\boldsymbol{w}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}))}{Z_{\boldsymbol{x}}}$$

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

$$\boldsymbol{w} = \arg\max_{\boldsymbol{w}} \sum_t \log P_{\boldsymbol{w}}(\boldsymbol{y}_t|\boldsymbol{x}_t) = \arg\min_{\boldsymbol{w}} \sum_t L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$$

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

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) = \mathbb{E}_Y[\boldsymbol{\phi}(\boldsymbol{x}, Y)] - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$$

The Story So Far

- Naive Bayes: generative, maximizes joint likelihood $P_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y})$
  - closed form solution (boils down to counting and normalizing)
- Logistic regression: discriminative, max. conditional likelihood $P_{\boldsymbol{w}}(\boldsymbol{y}|\boldsymbol{x})$
  - also called log-linear model and max-entropy classifier
  - no closed form solution
  - stochastic gradient updates look like

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^{k} + \eta\big(\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) - \mathbb{E}_{Y}[\boldsymbol{\phi}(\boldsymbol{x}, Y)]\big)$$

The Story So Far

- Naive Bayes: generative, maximizes joint likelihood $P_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y})$
  - closed form solution (boils down to counting and normalizing)
- Logistic regression: discriminative, max. conditional likelihood $P_{\boldsymbol{w}}(\boldsymbol{y}|\boldsymbol{x})$
  - also called log-linear model and max-entropy classifier
  - no closed form solution
  - stochastic gradient updates look like

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^{k} + \eta\big(\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) - \mathbb{E}_{Y}[\boldsymbol{\phi}(\boldsymbol{x}, Y)]\big)$$

- The Perceptron: discriminative, non-probabilistic classifier

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^{k} + \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{x}, \widehat{\boldsymbol{y}})$$

The Story So Far

- Naive Bayes: generative, maximizes joint likelihood $P_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y})$
  - closed form solution (boils down to counting and normalizing)
- Logistic regression: discriminative, max. conditional likelihood $P_{\boldsymbol{w}}(\boldsymbol{y}|\boldsymbol{x})$
  - also called log-linear model and max-entropy classifier
  - no closed form solution
  - stochastic gradient updates look like

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^{k} + \eta\big(\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) - \mathbb{E}_{Y}[\boldsymbol{\phi}(\boldsymbol{x}, Y)]\big)$$

- The Perceptron: discriminative, non-probabilistic classifier

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^{k} + \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{x}, \widehat{\boldsymbol{y}})$$

- **Relationship:** LR/Perceptron differ in how they interact with the current state of the model during training:
  the **prediction** $\boldsymbol{\phi}(\boldsymbol{x}, \widehat{\boldsymbol{y}})$ vs. the **expectation** $\mathbb{E}_{Y}[\boldsymbol{\phi}(\boldsymbol{x}, Y)]$.

Maximizing Margin

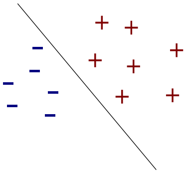
- For a training set $\mathcal{D}$
- Margin of a weight vector $\boldsymbol{w}$ is smallest γ such that

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

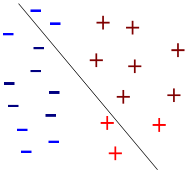
- for every training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$, $\boldsymbol{y}' \in \mathcal{Y}$

Margin

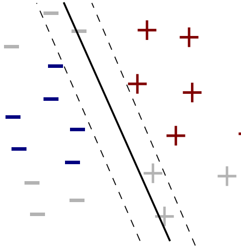
Training

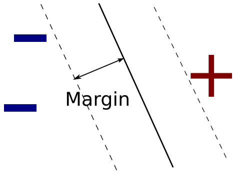


Testing



Denote the value of the margin by γ





Maximizing Margin

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

$$\epsilon \propto \frac{R^2}{\gamma^2 \times N}$$

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

Outline

Maximizing Margin

Let $\gamma > 0$

$$\max_{||\boldsymbol{w}|| \leq 1} \quad \gamma$$

subject to

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

$$\text{for all } (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$$

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

- Note: algorithm still minimizes error (0!) if data is separable

Maximizing Margin

Let $\gamma > 0$

$$\max_{||\boldsymbol{w}|| \leq 1} \quad \gamma$$

subject to

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

$$\text{for all } (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$$

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

- Note: algorithm still minimizes error (0!) if data is separable
- $||\boldsymbol{w}||$ is bound since scaling trivially produces larger margin

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||\boldsymbol{w}|| \leq 1} \quad \gamma$$

subject to

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

for all $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$

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

=

Min Norm:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}||\boldsymbol{w}||^2$$

subject to

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

for all $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$

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

- Instead of fixing $||\boldsymbol{w}||$ we fix the margin $\gamma = 1$
- Make substitution $\boldsymbol{w}' = \boldsymbol{w}/\gamma$; then we have $\gamma = \frac{\|\boldsymbol{w}\|}{\|\boldsymbol{w}'\|} = \frac{1}{\|\boldsymbol{w}'\|}$.

Support Vector Machines

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

subject to

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

$$\text{for all } (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D} \text{ and } \boldsymbol{y}' \in \mathcal{Y}$$

- Quadratic programming problem with many constraints
- Can be solved with many techniques.

Support Vector Machines

What if data is not separable?

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w},\xi} \; \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{t=1}^{N}\xi_t$$

subject to

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

$$\text{for all } (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D} \text{ and } \boldsymbol{y}' \in \mathcal{Y}$$

ξ_t: trade-off between margin per example and $\|\boldsymbol{w}\|$
Larger C = more examples correctly classified

Support Vector Machines

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

such that:

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

Support Vector Machines

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w},\xi} \; \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{t=1}^{N}\xi_t$$

such that:

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

Support Vector Machines

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w},\xi} \; \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{t=1}^{N}\xi_t$$

such that:

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

Support Vector Machines

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w},\xi} \; \frac{\lambda}{2}||\boldsymbol{w}||^2 + \sum_{t=1}^{N} \xi_t \qquad \lambda = \frac{1}{C}$$

such that:

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

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w},\xi} \; \frac{\lambda}{2}||\boldsymbol{w}||^2 + \sum_{t=1}^{N} \xi_t \qquad \lambda = \frac{1}{C}$$

such that:

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

If $\|\boldsymbol{w}\|$ classifies $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ with margin 1, penalty $\xi_t = 0$
Otherwise penalty $\xi_t = 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \; \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)$

Support Vector Machines

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w},\xi} \; \frac{\lambda}{2}||\boldsymbol{w}||^2 + \sum_{t=1}^{N} \xi_t \qquad \lambda = \frac{1}{C}$$

such that:

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

If $\|\boldsymbol{w}\|$ classifies $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ with margin 1, penalty $\xi_t = 0$
Otherwise penalty $\xi_t = 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \; \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)$

Hinge loss:

$$L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) = \max\left(0, 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \; \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)\right)$$

Support Vector Machines

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

such that:

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

Hinge loss equivalent

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

$$\text{where } L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) = \max(0, 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}} \; \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}))$$

$$= \left(\max_{\boldsymbol{y}' \in \mathcal{Y}} \; \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') + [[\boldsymbol{y}' \neq \boldsymbol{y}]] \right) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$$

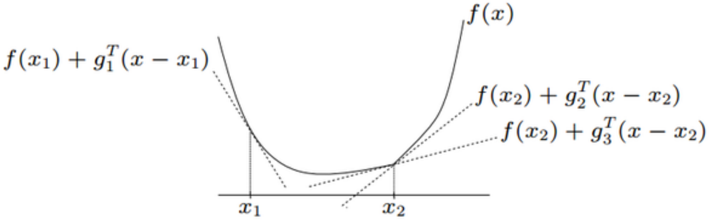
From Gradient to Subgradient

The hinge loss is a piecewise linear function—not differentiable everywhere

Cannot use gradient descent, we must turn to subgradient descent.

Implementation is identical, but convergence properties can be worse.

Recap: Subgradient



- Defined for convex functions $f : \mathbb{R}^D \to \mathbb{R}$
- Generalizes the notion of gradient—in points where f is differentiable, there is a single subgradient which equals the gradient
- Other points may have multiple subgradients

Subgradient Descent

$$L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) \;=\; \left(\max_{\boldsymbol{y}' \in \mathcal{Y}} \; \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') + [[\boldsymbol{y}' \neq \boldsymbol{y}]]\right) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$$

A subgradient of the hinge loss is

$$\partial_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) \;\ni\; \boldsymbol{\phi}(\boldsymbol{x}, \widehat{\boldsymbol{y}}) - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$$

where

$$\widehat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}' \in \mathcal{Y}} \; \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') + [[\boldsymbol{y}' \neq \boldsymbol{y}]]$$

This gives us a way to train SVMs with (stochastic) sub-gradients!

Perceptron and Hinge-Loss

SVM update:

$$\widehat{\boldsymbol{y}} := \arg\max_{\boldsymbol{y}' \in \mathcal{Y}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') + [[\boldsymbol{y}' \neq \boldsymbol{y}_t]];$$

$$\boldsymbol{w}^{k+1} \leftarrow \boldsymbol{w}^k - \eta \begin{cases} 0, & \text{if } \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) \geq \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \widehat{\boldsymbol{y}}) \\ \boldsymbol{\phi}(\boldsymbol{x}_t, \widehat{\boldsymbol{y}}) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t), & \text{otherwise} \end{cases}$$

Perceptron update: (with $\eta = 1$)

$$\widehat{\boldsymbol{y}} := \arg\max_{\boldsymbol{y}' \in \mathcal{Y}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$$

$$\boldsymbol{w}^{k+1} \leftarrow \boldsymbol{w}^k - \eta \begin{cases} 0, & \text{if } \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) \geq \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \widehat{\boldsymbol{y}}) \\ \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t), & \text{otherwise} \end{cases}$$

Perceptron = Stochastic subgradient updates on the marginless hinge

$$L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) = \max_{\boldsymbol{y}'} \boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{y}') + [[y' \neq y]] - \boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{y})$$

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

Loss Functions

Perceptron:

$$L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) = \max_{\boldsymbol{y}' \in \mathcal{Y}} \left(\boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{y}') \right) \qquad - \boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{y})$$

SVM (a.k.a. Hinge, Max-Margin)

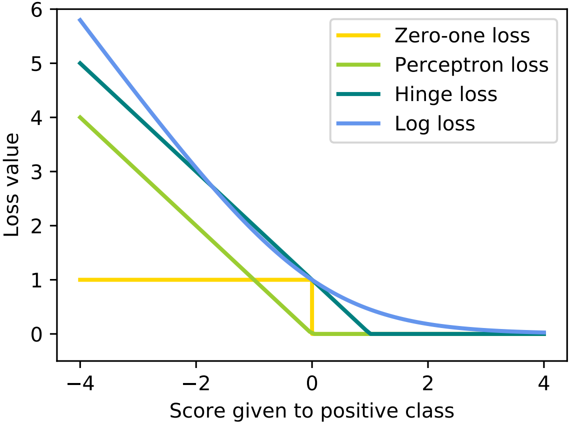
$$L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) = \max_{\boldsymbol{y}' \in \mathcal{Y}} \left(\boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{y}') + [[\boldsymbol{y}' \neq \boldsymbol{y}]] \right) \qquad - \boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{y})$$

Multinomial Logistic Regression (a.k.a. Cross-Entropy, MaxEnt)

$$L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) = \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \left(\boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{y}') \right) \qquad - \boldsymbol{w} \cdot \boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{y})$$

- Clearly, they are very similar!
- Tractable surrogates for the misclassification error rate.

Loss Functions



Summary

What we have covered

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

What is next

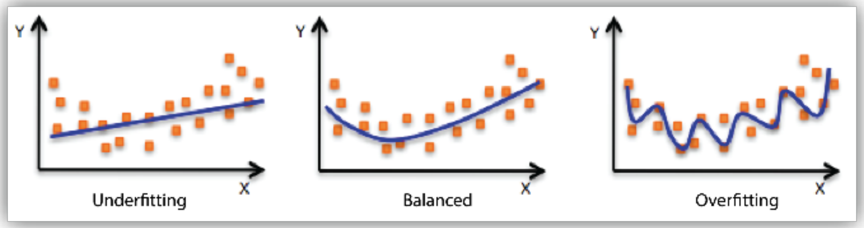
- Regularization
- Non-linear classifiers

Outline

Regularization

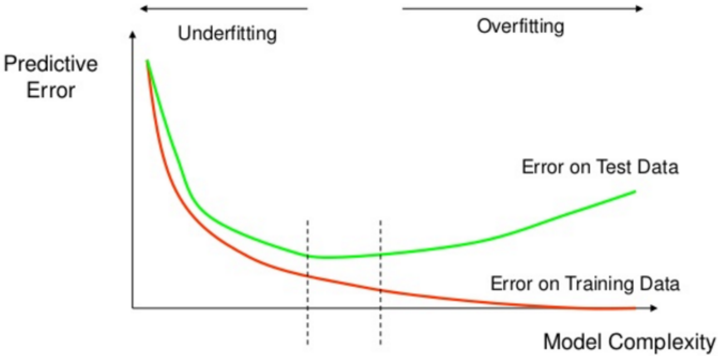
Overfitting

If the model is too complex (too many parameters) and the data is scarce, we run the risk of overfitting:



- We saw one example already when talking about add-one smoothing in Naive Bayes!

Empirical Risk Minimization



Regularization

In practice, we regularize models to prevent overfitting

$$\arg\min_{\boldsymbol{w}} \sum_{t=1}^{N} L(\boldsymbol{w}; (x_t, y_t)) + \lambda\Omega(\boldsymbol{w}),$$

where $\Omega(\boldsymbol{w})$ is the regularization function, and λ controls how much to regularize.

- Gaussian prior (ℓ_2), promotes smaller weights:

$$\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2 = \sum_i \boldsymbol{w}_i^2.$$

- Laplacian prior (ℓ_1), promotes sparse weights!

$$\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

Logistic Regression with ℓ_2 Regularization

$$\sum_{t=1}^{N} L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) + \lambda\Omega(\boldsymbol{w}) = -\sum_{t=1}^{N} \log\left(\exp(\boldsymbol{w} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y}_t))/Z_{\boldsymbol{x}}\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

- What is the new gradient?

$$\sum_{t=1}^{N} \nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) + \nabla_{\boldsymbol{w}} \lambda\Omega(\boldsymbol{w})$$

- We know $\nabla_w L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$
- Just need $\nabla_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \lambda\boldsymbol{w}$

Support Vector Machines

Hinge-loss formulation: ℓ_2 regularization already happening!

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

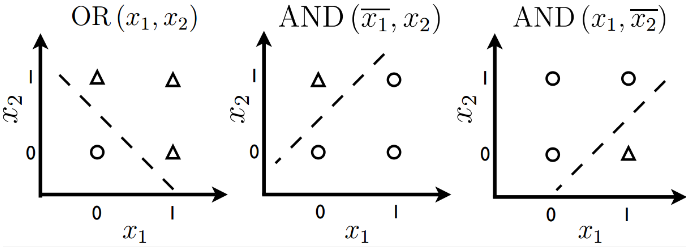
↑ SVM optimization ↑

(Of course, ℓ_1 or other penalties might be better in some cases!)

Outline

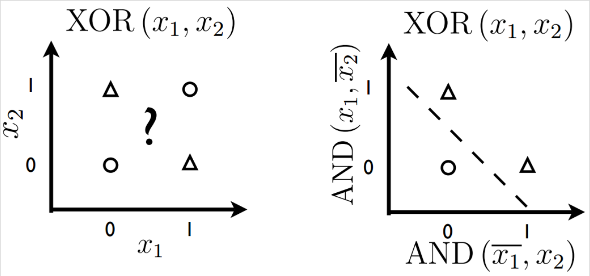
Recap: What a Linear Classifier Can Do

- It **can** solve linearly separable problems (OR, AND)



Recap: What a Linear Classifier Can't Do

- ... but it **can't** solve non-linearly separable problems such as simple XOR (unless input is transformed into a better representation):



- This was observed by Minsky and Papert (1969) (for the perceptron) and motivated strong criticisms

Summary: Linear Classifiers

We've seen

- Perceptron
- Naive Bayes
- Logistic regression
- Support vector machines

All lead to convex optimization problems $\Rightarrow$ no issues with local minima/initialization

All assume the features are well-engineered such that the data is nearly linearly separable

What If Data Are Not Linearly Separable?

What If Data Are Not Linearly Separable?

Engineer better features (often works!)

What If Data Are Not Linearly Separable?

Engineer better features (often works!)

Kernel methods: (not in this class)

- works implicitly in a high-dimensional feature space
- ... but still need to choose/design a good kernel
- model capacity confined to positive-definite kernels

What If Data Are Not Linearly Separable?

Engineer better features (often works!)

Kernel methods: (not in this class)

- works implicitly in a high-dimensional feature space
- ... but still need to choose/design a good kernel
- model capacity confined to positive-definite kernels

Neural networks

- embrace non-convexity and local minima
- instead of engineering features, engineer the model architecture

Conclusions

- Linear classifiers are a broad class including well-known ML methods such as perceptron, Naive Bayes, logistic regression, support vector machines
- They all involve manipulating weights and features
- They either lead to closed-form solutions or convex optimization problems (no local minima)
- Stochastic gradient descent algorithms are useful if training datasets are large
- However, they require manual specification of feature representations

References I

Minsky, M. and Papert, S. (1969). Perceptrons.

Novikoff, A. B. (1962). On convergence proofs for perceptrons. In *Symposium on the Mathematical Theory of Automata*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.