Linear Classifiers

André Martins

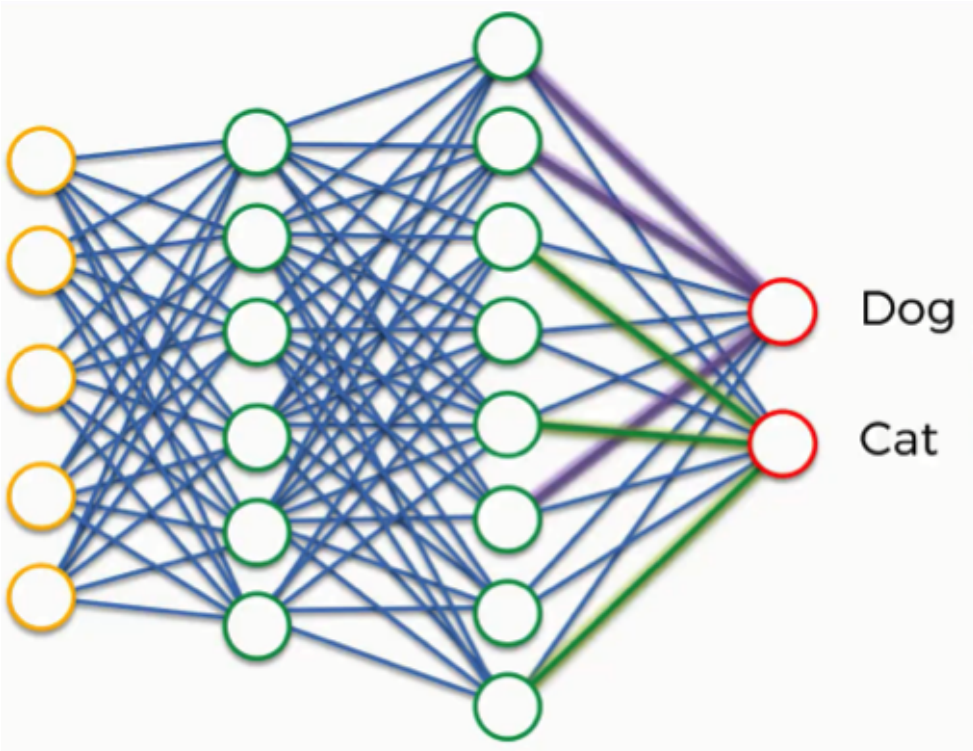
Lisbon Machine Learning School, July 8, 2021

Why Linear Classifiers?

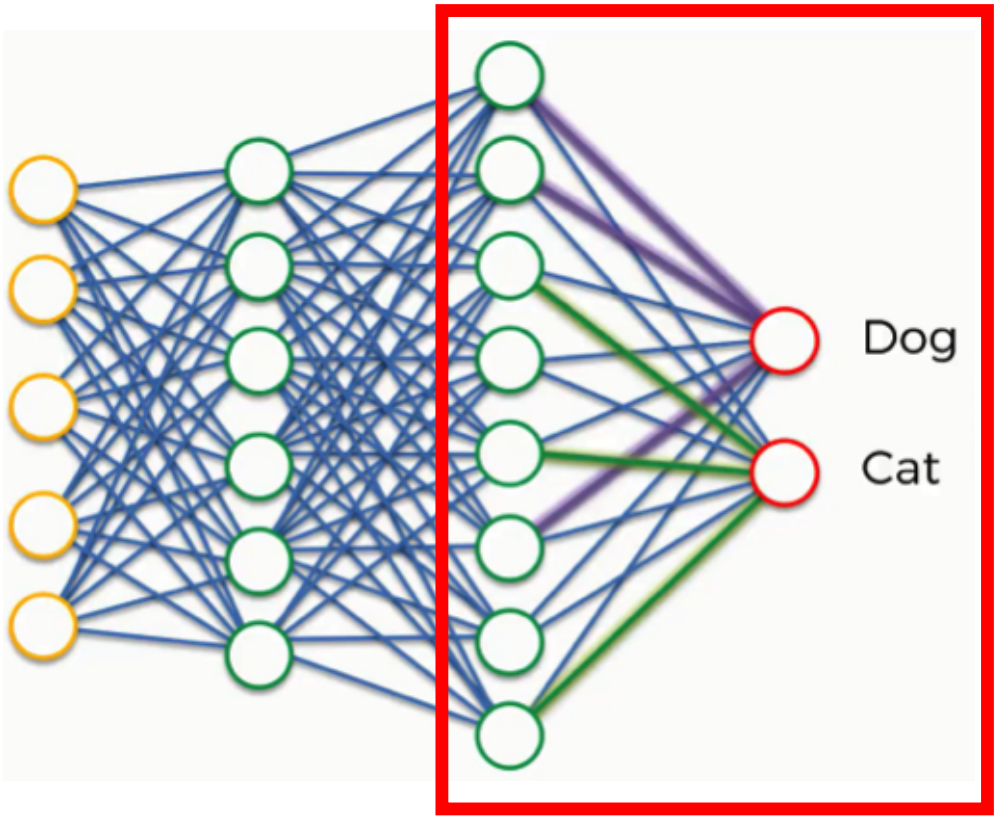
It's 2021 and everybody uses neural networks. Why a lecture on linear classifiers?

- The underlying machine learning concepts are the same
- The theory (statistics and optimization) are much better understood
- Linear classifiers are still widely used (and very effective when data is scarce)
- Linear classifiers are **a component of neural networks**.

Linear Classifiers and Neural Networks

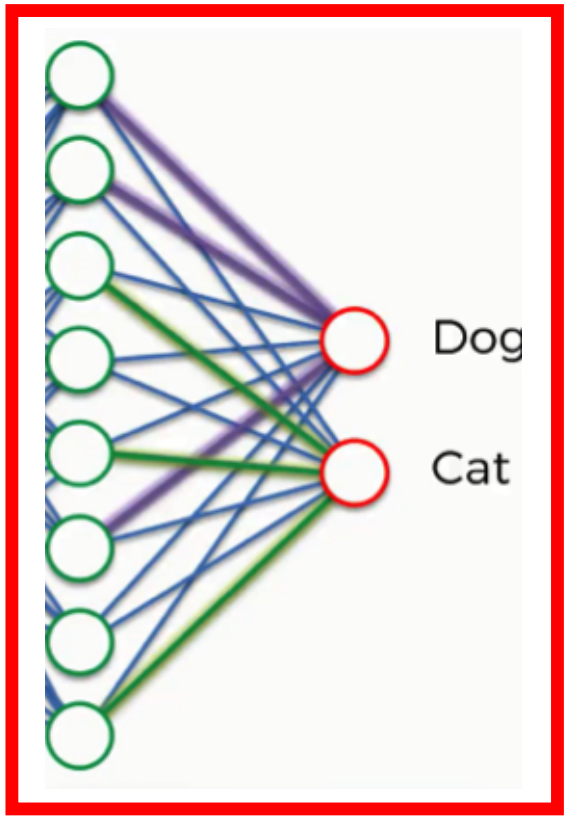


Linear Classifiers and Neural Networks



Linear Classifier

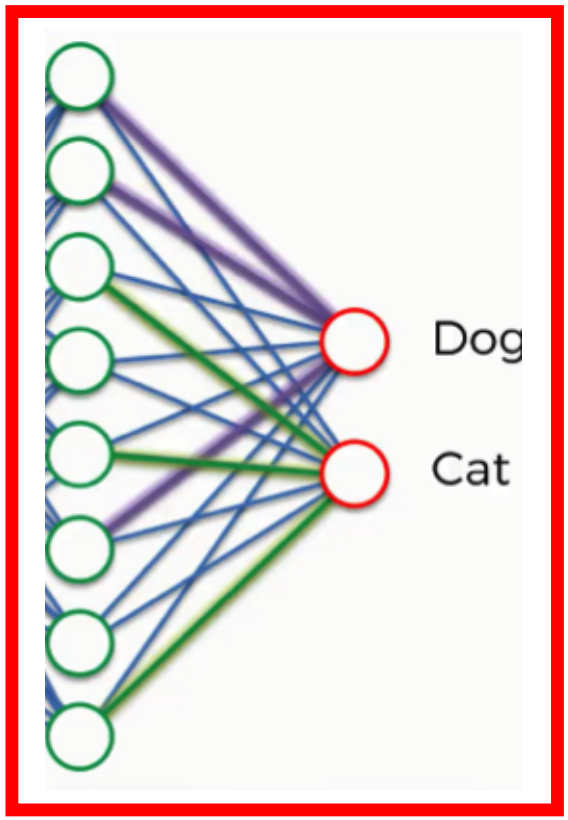
Linear Classifiers and Neural Networks



Linear Classifier

Linear Classifiers and Neural Networks

Handcrafted Features



Linear Classifier

Today's Roadmap

- Linear regression
- Binary and multi-class classification
- Linear classifiers: perceptron, naive Bayes, logistic regression, SVMs
- Softmax and sparsemax
- Regularization and optimization, stochastic gradient descent
- Similarity-based classifiers and kernels.

Example Tasks

Binary: given an e-mail: is it spam or not-spam?

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

AlphaGo Beats Go Human Champ: Godfather Of Deep Learning Tells Us Do Not Be Afraid Of AI

21 March 2016, 10:16 am EDT By Aaron Mamiit Tech Times

Last week, Google's artificial intelligence program

Last week, Google's artificial intelligence program AlphaGo dominated its match with South Korean world Go champion Lee Sedol, winning with a 4-1 score.

The achievement stunned artificial intelligence experts, who previously thought that Google's computer program would need at least 10 more years before developing enough to be able to beat a human world champion.

sports
politics
technology
economy
weather
culture

Outline

Disclaimer

Some 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** $+1$
$P(-1\|\star) = \frac{\text{count}(\star \text{ and } -1)}{\text{count}(\star)} = \frac{2}{3} = 0.67$	vs.	$P(+1\|\star) = \frac{\text{count}(\star \text{ and } +1)}{\text{count}(\star)} = \frac{1}{3} = 0.33$	
$P(-1\|\diamond) = \frac{\text{count}(\diamond \text{ and } -1)}{\text{count}(\diamond)} = \frac{1}{2} = 0.5$	vs.	$P(+1\|\diamond) = \frac{\text{count}(\diamond \text{ and } +1)}{\text{count}(\diamond)} = \frac{1}{2} = 0.5$	
$P(-1\|\heartsuit) = \frac{\text{count}(\heartsuit \text{ and } -1)}{\text{count}(\heartsuit)} = \frac{1}{1} = 1.0$	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** $+1$
$P(-1\|\star) = \frac{\text{count}(\star \text{ and } -1)}{\text{count}(\star)} = \frac{2}{3} = 0.67$	vs.	$P(+1\|\star) = \frac{\text{count}(\star \text{ and } +1)}{\text{count}(\star)} = \frac{1}{3} = 0.33$
$P(-1\|\triangle) = \frac{\text{count}(\triangle \text{ and } -1)}{\text{count}(\triangle)} = \frac{1}{3} = 0.33$	vs.	$P(+1\|\triangle) = \frac{\text{count}(\triangle \text{ and } +1)}{\text{count}(\triangle)} = \frac{2}{3} = 0.67$
$P(-1\|\circ) = \frac{\text{count}(\circ \text{ and } -1)}{\text{count}(\circ)} = \frac{1}{2} = 0.5$	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 $= \text{argmax}_{\text{label}}\, P(\text{label}|\text{sequence})$

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

- Input/Output pair: $(x, y) \in \mathcal{X} \times \mathcal{Y}$
  - 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:

$$\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N} \subseteq \mathcal{X} \times \mathcal{Y}$$

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

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

- Hopefully, $\widehat{y} \approx y$ 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., spam 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

Classification

Deals with **discrete** output variables:

- **Binary classification:** $\mathcal{Y} = \{\pm 1\}$
  - e.g., spam 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
  - See Xavier Carreras' lecture later at LxMLS!

Classification

Deals with **discrete** output variables:

- **Binary classification:** $\mathcal{Y} = \{\pm 1\}$
  - e.g., spam 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
  - See Xavier Carreras' lecture later at LxMLS!

Today we'll focus mostly on multi-class classification.

Sometimes reductions are convenient:

- logistic regression reduces classification to regression
- one-vs-all reduces multi-class to binary
- greedy search reduces structured classification to multi-class

... but other times it's better to tackle the problem in its native form.

More later!

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 x

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

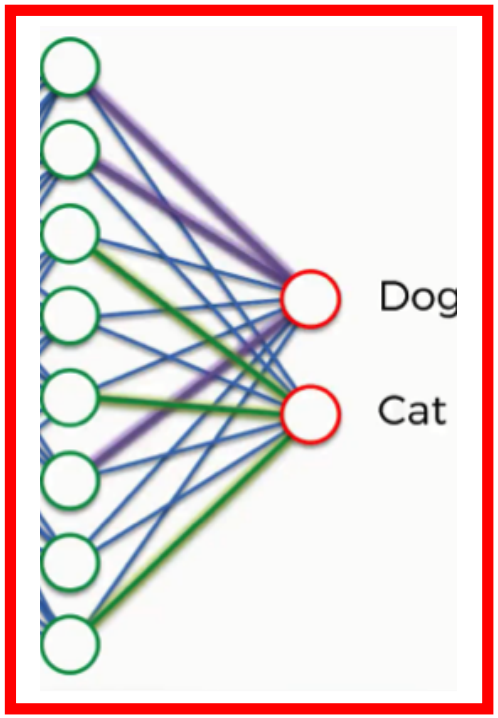
- $\phi(x)$ is a high dimensional feature vector

We can use feature vectors to encapsulate **Boolean**, **categorical**, and **continuous** features

- e.g., categorical features can be reduced to a range of one-hot binary values.

Example: Continuous Features

Handcrafted Features



Linear Classifier

Feature Engineering and NLP Pipelines

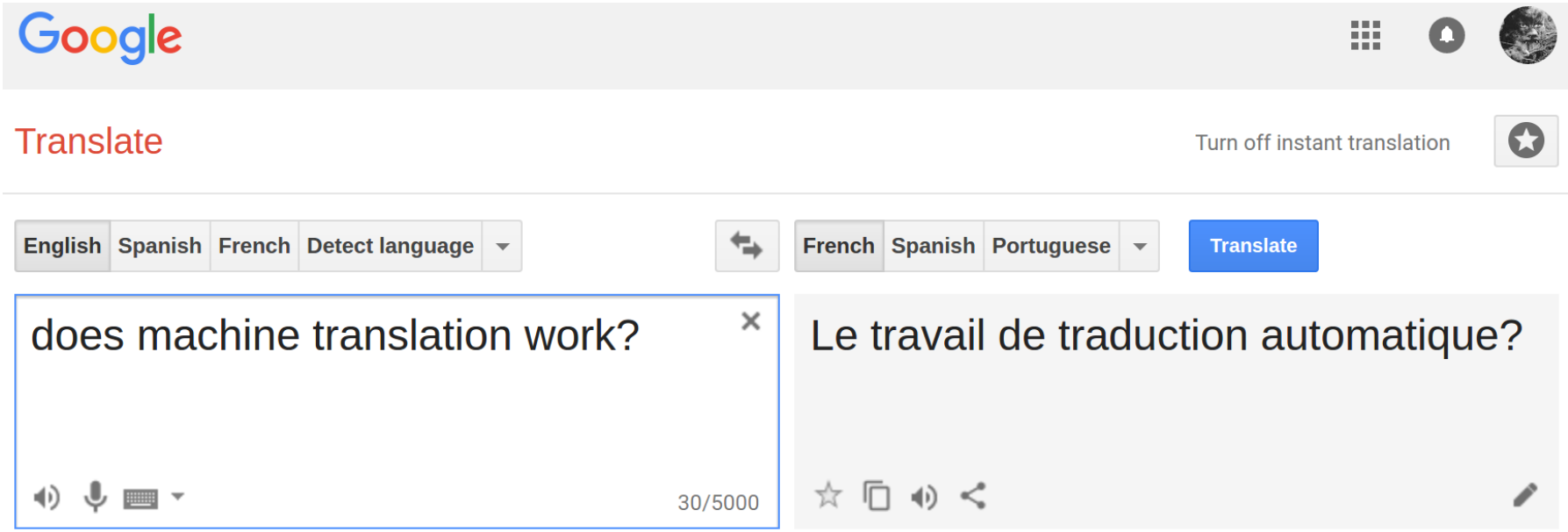
Classical NLP pipelines consist of stacking together several linear classifiers

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

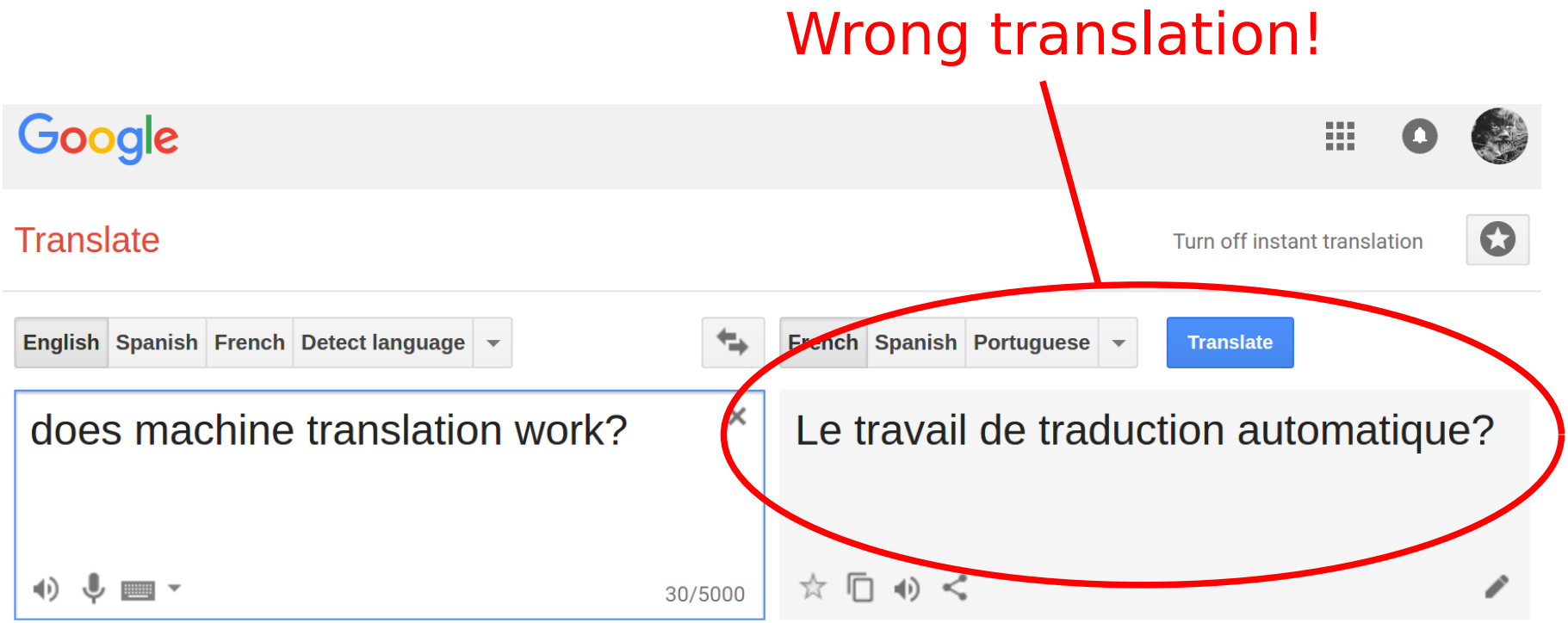
Examples of features:

- Word occurrences: binary feature denoting if a word occurs in not in a document
- Word counts: real-valued feature counting how many times a word occurs
- POS tags: adjective counts for sentiment analysis
- Spell checker: misspellings counts for spam detection

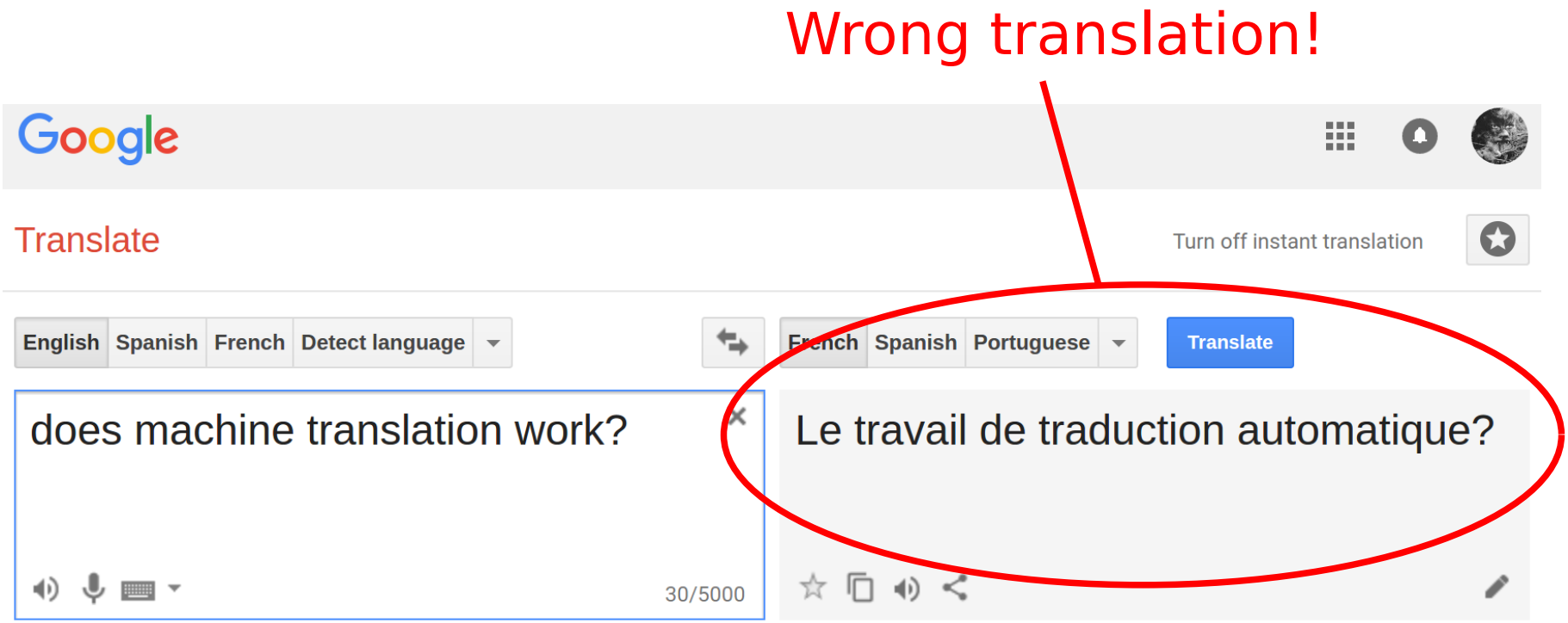
Example: Translation Quality Estimation



Example: Translation Quality Estimation



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")

One alternative to feature engineering: **representation learning**

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")

One alternative to feature engineering: **representation learning**

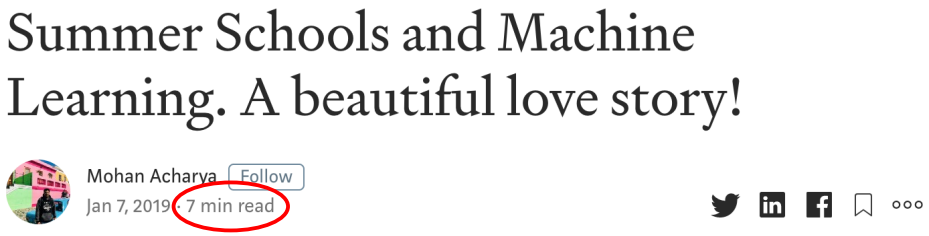
Bhiksha will talk about this tomorrow!

Outline

Regression

Output space $\mathcal{Y}$ is continuous

Example: given an article, how much time a user spends reading it?

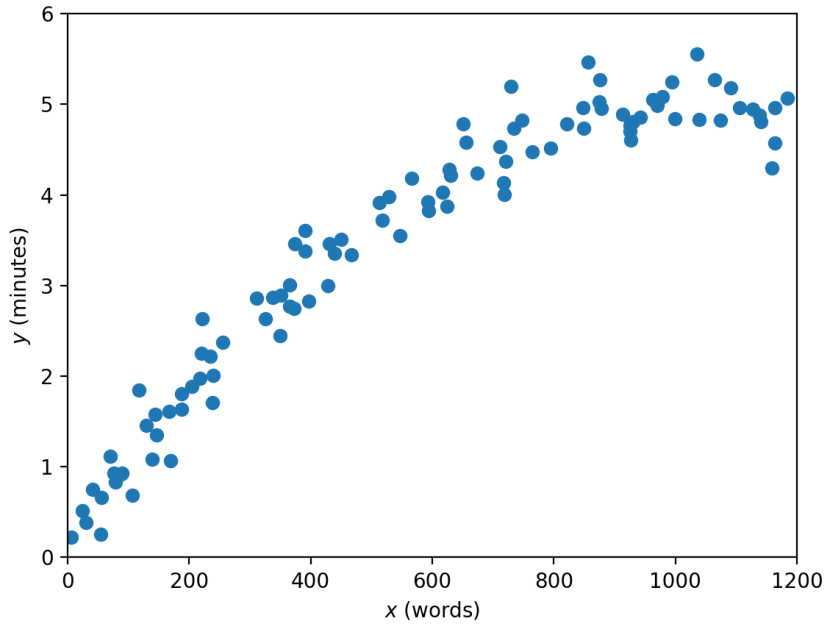


- x is number of words of the article
- y is the reading time (minutes)

How to define a model that predicts $\widehat{y}$ from x?

Linear Regression

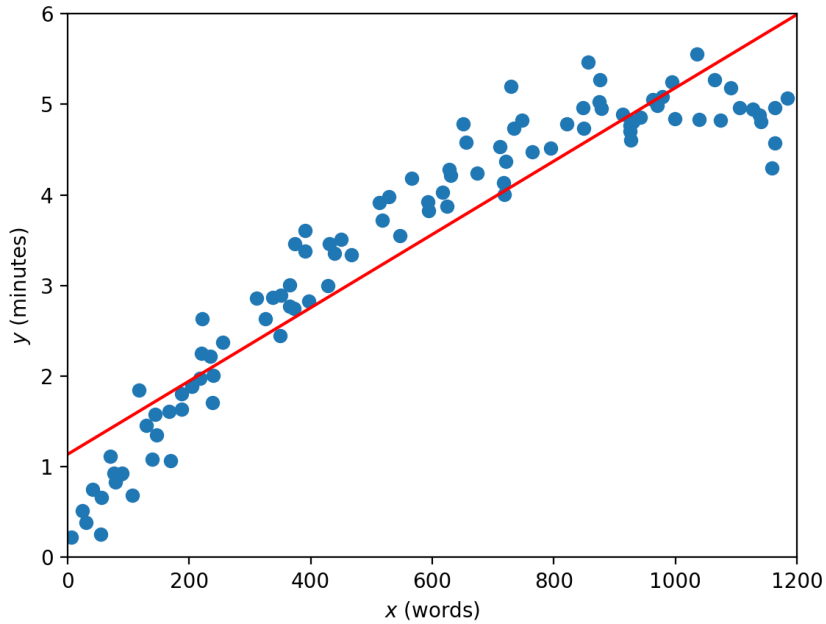
- First take: assume $\widehat{y} = wx + b$
- Model parameters: w and b
- Given training data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$, how to estimate w and b?



Least squares method: fit w and b on the training set by minimizing $\sum_{n=1}^{N}(y_n - (wx_n + b))^2$

Linear Regression

- First take: assume $\widehat{y} = wx + b$
- Model parameters: w and b
- Given training data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$, how to estimate w and b?



Least squares method: fit w and b on the training set by minimizing $\sum_{n=1}^{N}(y_n - (wx_n + b))^2$

Linear Regression

Often a linear dependency of $\widehat{y}$ on x is a poor assumption

Second take: assume $\widehat{y} = \boldsymbol{w} \cdot \boldsymbol{\phi}(x)$, where $\boldsymbol{\phi}(x)$ is a feature vector

- e.g. $\boldsymbol{\phi}(x) = [1, x, x^2, \ldots, x^D]$ (polynomial features degree $\leq D$)
- the bias term b is captured by the constant feature $\phi_0(x) = 1$

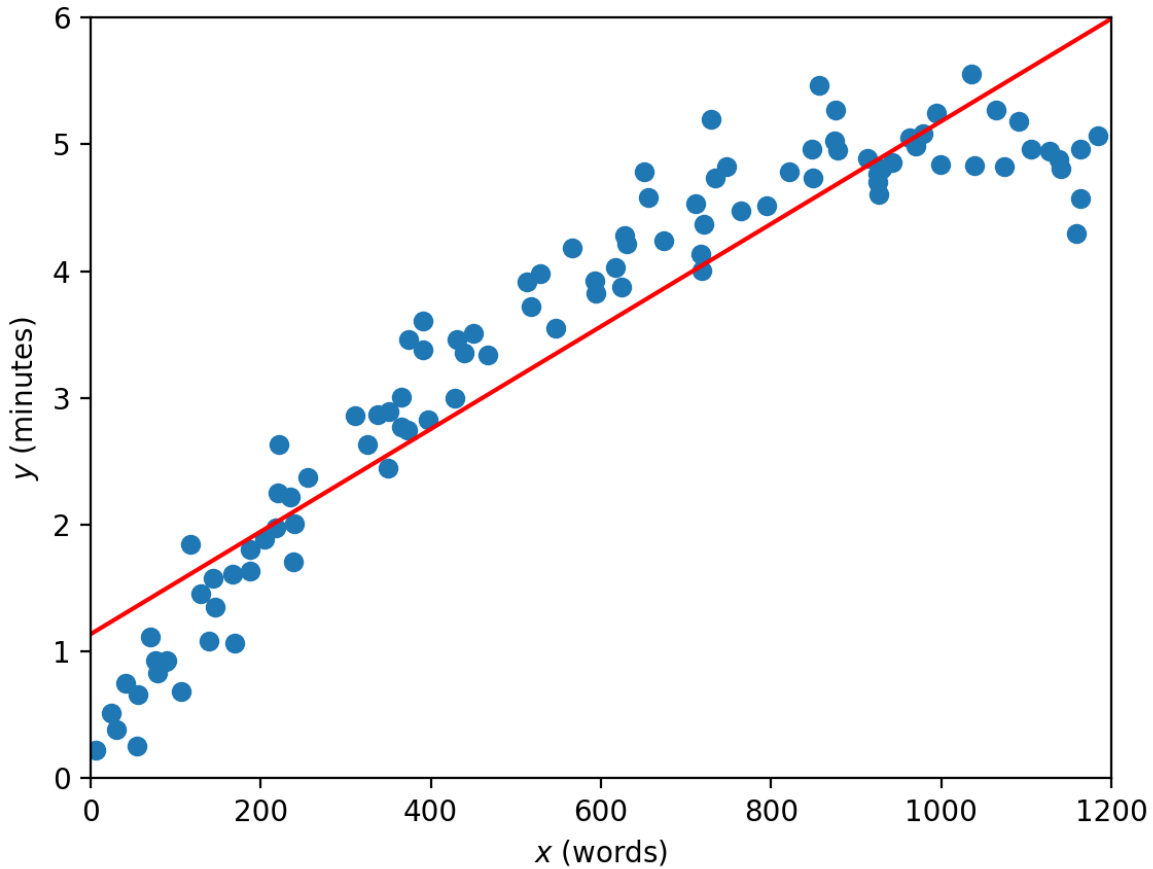
Fit $\boldsymbol{w}$ by minimizing $\sum_n (y_n - (\boldsymbol{w} \cdot \boldsymbol{\phi}(x_n)))^2$

- Closed form solution:

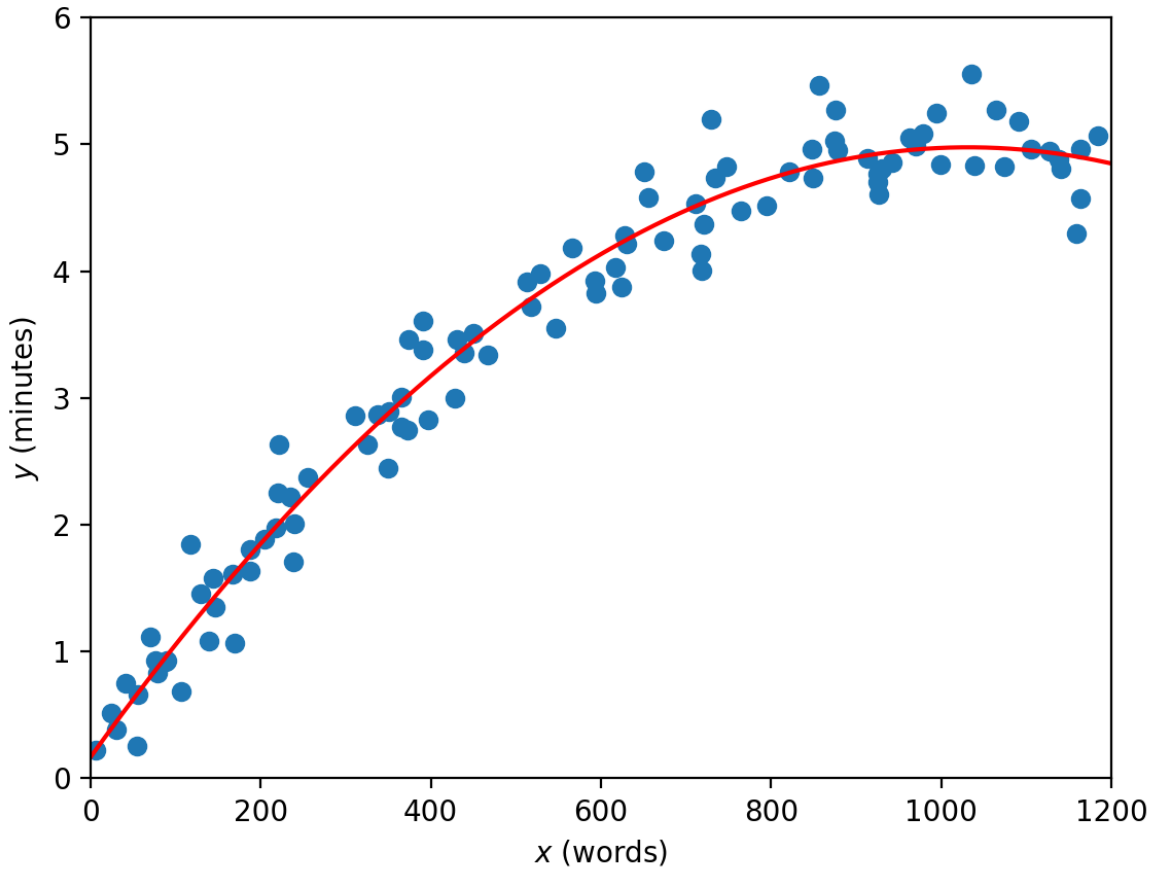
$$\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \text{ with } \boldsymbol{X} = \begin{bmatrix} \vdots \\ \boldsymbol{\phi}(x_n)^\top \\ \vdots \end{bmatrix}, \ \boldsymbol{y} = \begin{bmatrix} \vdots \\ y_n \\ \vdots \end{bmatrix}.$$

Still called linear regression – linearity w.r.t. the model parameters $\boldsymbol{w}$.

Linear Regression ($D = 1$)



Linear Regression ($D = 2$)



Squared Loss Function

Linear regression with the least squares method corresponds to a loss function

$$L(y, \widehat{y}) = \frac{1}{2}(y - \widehat{y})^2, \quad \text{where } \widehat{y} = \boldsymbol{w} \cdot \boldsymbol{\phi}(x).$$

The model is fit to the training data by minimizing this loss function.

This is called the squared loss.

More later.

Least Squares – Probabilistic Interpretation

The least squares method has a probabilistic interpretation.

Assume the data is generated stochastically as

$$y = \boldsymbol{w}^* \cdot \boldsymbol{\phi}(x) + n$$

where $n \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise (with σ fixed), and $\boldsymbol{w}^*$ are the "true" model parameters.

That is, $y \sim \mathcal{N}(\boldsymbol{w}^* \cdot \boldsymbol{\phi}(x), \sigma^2)$.

Then $\boldsymbol{w}$ given by least squares is the maximum likelihood estimate under this model.

One-Slide Proof

Recall $\mathcal{N}(y; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$.

$$
\begin{aligned}
\hat{\boldsymbol{w}}_{\text{MLE}} &= \arg\max_{\boldsymbol{w}} \prod_{n=1}^{N} P(y_n \mid x_n; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \sum_{n=1}^{N} \log P(y_n \mid x_n; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \sum_{n=1}^{N} -\frac{(y_n - \boldsymbol{w} \cdot \boldsymbol{\phi}(x_n))^2}{2\sigma^2} - \underbrace{\log(\sqrt{2\pi}\sigma)}_{\text{constant}} \\
&= \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} (y_n - \boldsymbol{w} \cdot \boldsymbol{\phi}(x_n))^2
\end{aligned}
$$

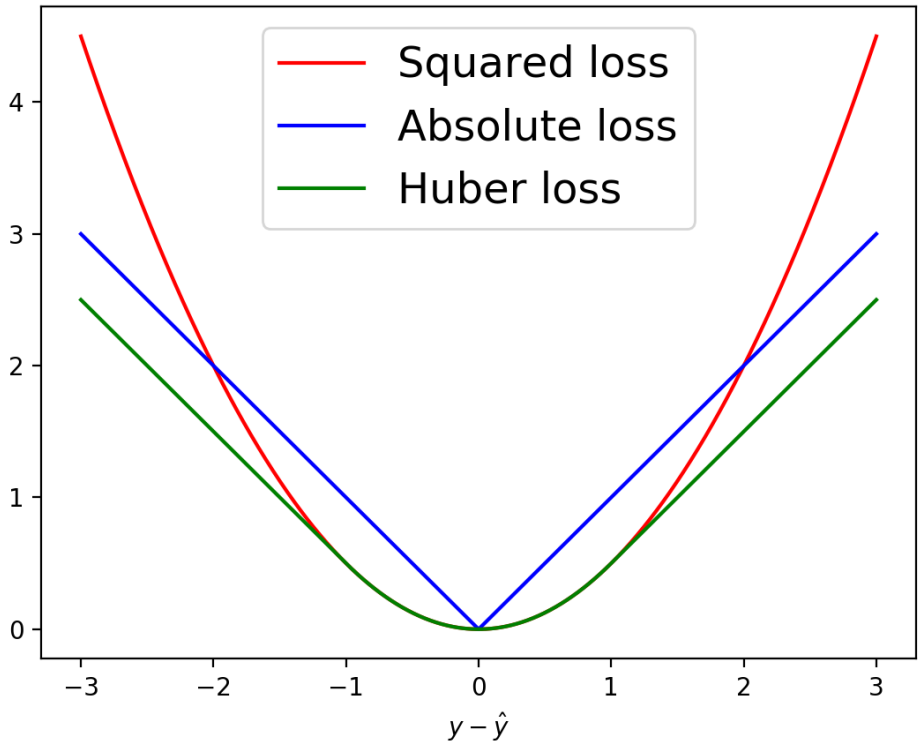
Thus, linear regression with the squared loss = MLE under Gaussian noise.

Other Regression Losses

Squared loss: $L(y, \widehat{y}) = \frac{1}{2}(y - \widehat{y})^2$.

Absolute error loss: $L(y, \widehat{y}) = |y - \widehat{y}|$.

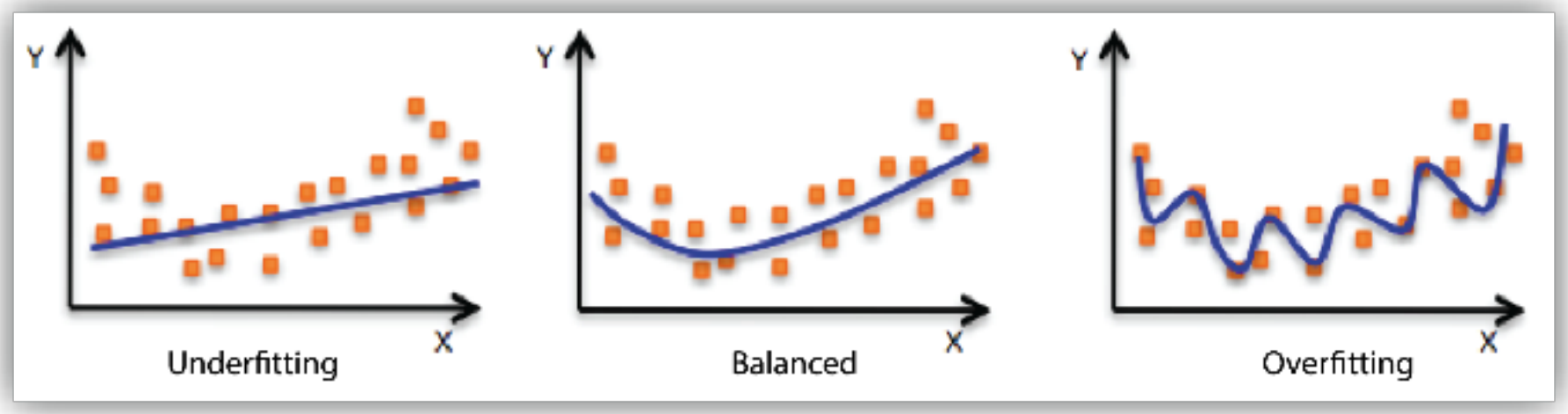
Huber loss: $L(y, \widehat{y}) = \begin{cases} \frac{1}{2}(y - \widehat{y})^2 & \text{if } |y - \widehat{y}| \leq 1 \\ |y - \widehat{y}| - \frac{1}{2} & \text{if } |y - \widehat{y}| \geq 1. \end{cases}$



Overfitting and Underfitting

We saw earlier an example of underfitting.

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



To avoid overfitting, we need regularization (more later).

Maximum A Posteriori

Assuming we have a prior distribution on $\boldsymbol{w}$, $\boldsymbol{w} \sim \mathcal{N}(0, \sigma_{\boldsymbol{w}}^2 \boldsymbol{I})$

A criterion to estimate $\boldsymbol{w}^*$ is maximum a posteriori (MAP):

$$
\begin{aligned}
\hat{\boldsymbol{w}}_{\text{MAP}} &= \arg\max_{\boldsymbol{w}} P(\boldsymbol{w}) \prod_{n=1}^{N} P(y_n \mid x_n; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \log P(\boldsymbol{w}) + \sum_{n=1}^{N} \log P(y_n \mid x_n; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} -\frac{\|\boldsymbol{w}\|^2}{2\sigma_{\boldsymbol{w}}^2} - \sum_{n=1}^{N} -\frac{(y_n - \boldsymbol{w} \cdot \boldsymbol{\phi}(x_n))^2}{2\sigma^2} + \text{constant} \\
&= \arg\min_{\boldsymbol{w}} \frac{\lambda \|\boldsymbol{w}\|^2}{2} + \sum_{n=1}^{N} (y_n - \boldsymbol{w} \cdot \boldsymbol{\phi}(x_n))^2
\end{aligned}
$$

Thus, ℓ_2-regularizarion is equivalent to MAP with a Gaussian prior.

Outline

Binary Classification

Before covering multi-class classification, we address the simpler case of binary classification

Output space $\mathcal{Y} = \{-1, +1\}$

Example: Given a news article, is it true or fake?

- x is the news article, represented a feature vector $\boldsymbol{\phi}(x)$
- y can be either true $(+1)$ or fake (-1)

How to define a model to predict $\widehat{y}$ from x?

Linear Classifier

Defined by $\widehat{y} = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{\phi}(x) + b) = \begin{cases} +1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{\phi}(x) + b \geq 0 \\ -1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{\phi}(x) + b < 0. \end{cases}$

Intuitively, $\boldsymbol{w} \cdot \boldsymbol{\phi}(x) + b$ is a "score" for the positive class: if positive, predict $+1$; if negative, predict -1

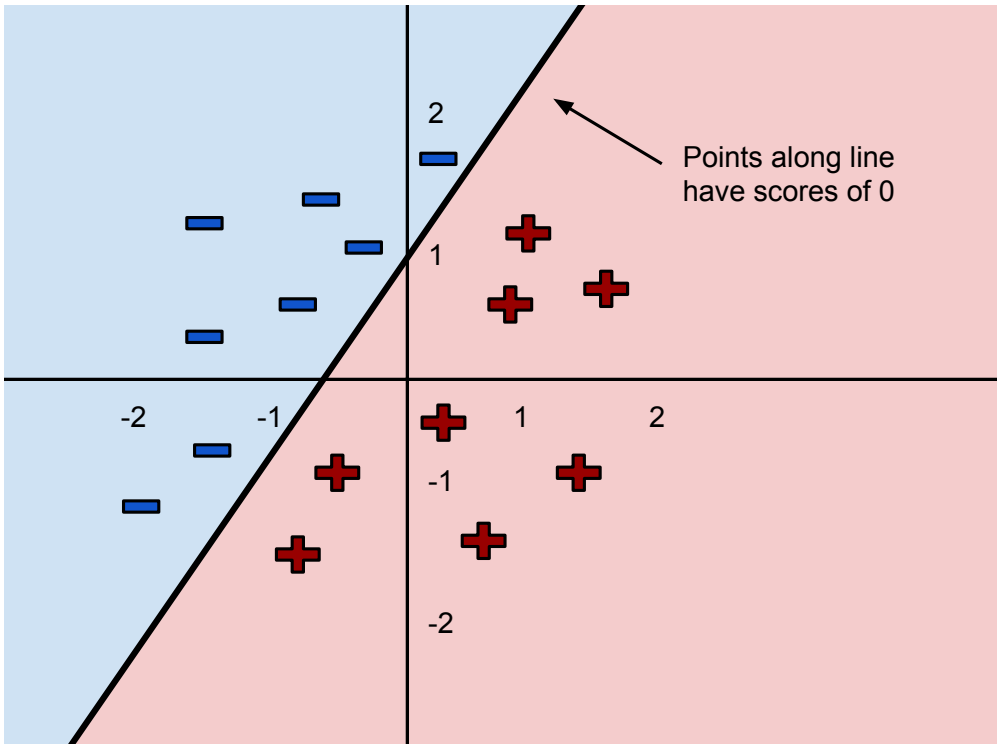
Difference from regression: the sign function converts from continuous to binary

The decision boundary is an hyperplane defined by the model parameters $\boldsymbol{w}$ and b

Also called a "hyperplane classifier."

Linear Classifier

(w, b) is an hyperplane that splits the space into two half spaces:

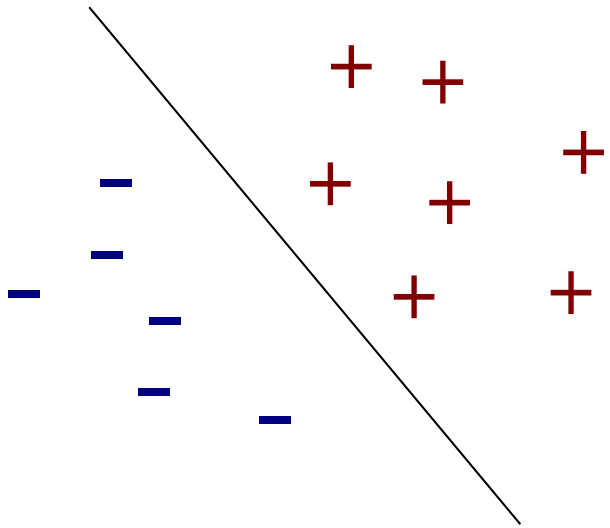


How to learn this hyperplane from the training data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$?

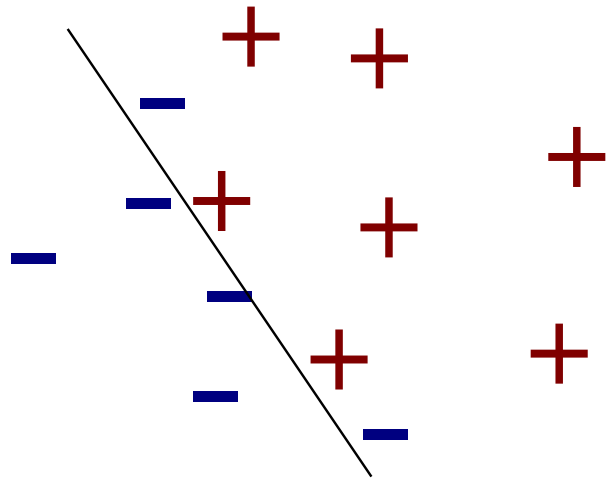
Linear Separability

- A dataset $\mathcal{D}$ is linearly separable if there exists $(\boldsymbol{w}, b)$ such that classification is perfect

Separable



Not Separable



We next present an algorithm that finds such an hyperplane if it exists!

Linear Classifier: No Bias Term

It is common to present linear classifiers without the bias term b:
$\widehat{y} = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{\phi}(x) + b)$

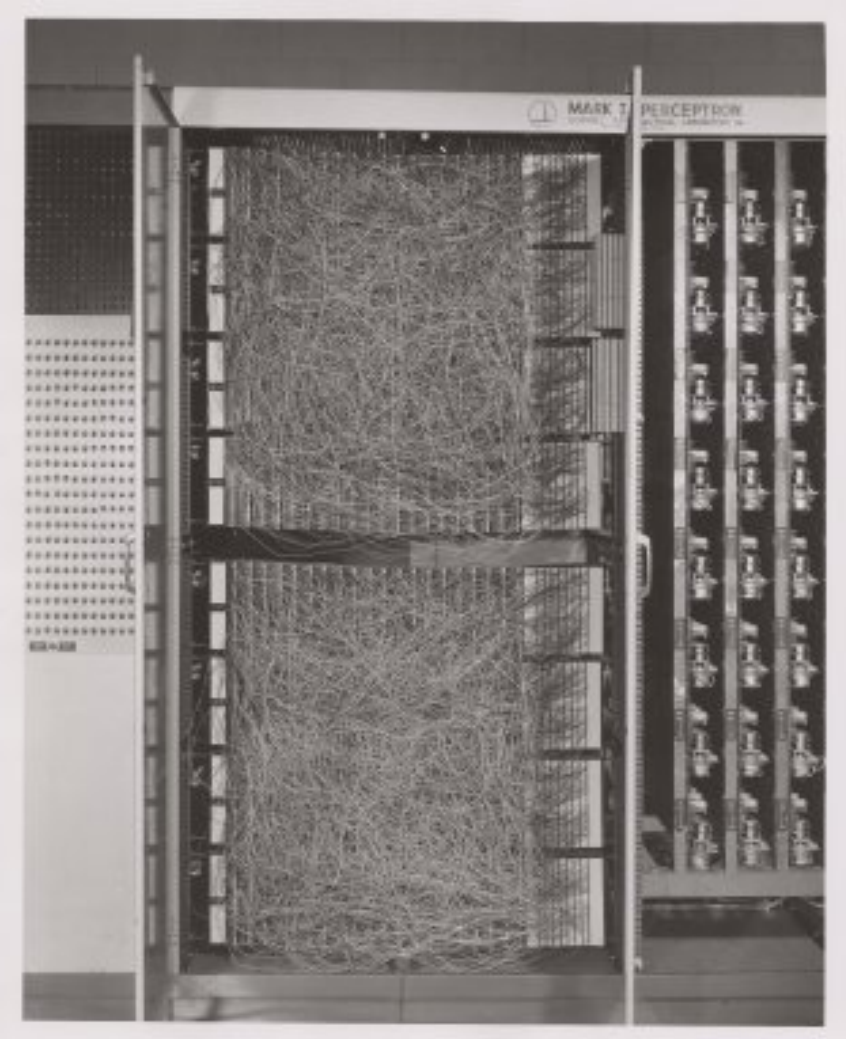
In this case, the decision boundary is a hyperplane that passes through the origin

We can always do this without loss of generality:

- Add a constant feature to $\boldsymbol{\phi}(x)$: $\phi_0(x) = 1$
- Then the corresponding weight w_0 replaces the bias term b

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...

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.

1958 New York Times...

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.

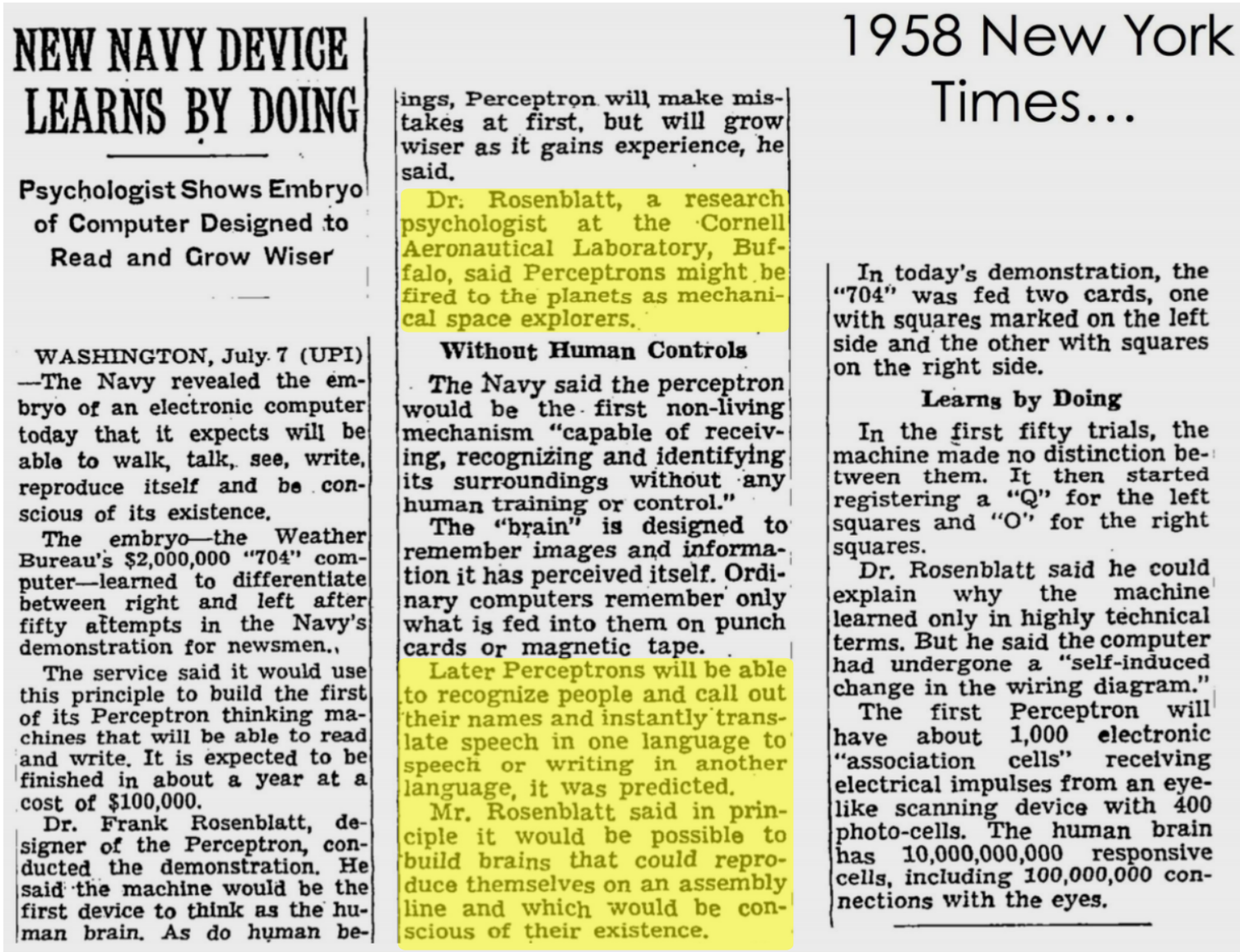
Learns by Doing

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...



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.

1958 New York Times...

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

1. Take x_i; apply the current model to make a prediction for it
2. If prediction is correct, do nothing
3. Else, correct model $\boldsymbol{w}$ by adding/subtracting feature vector $\boldsymbol{\phi}(x_i)$

For simplicity, omit the bias b: assume a constant feature $\phi_0(x) = 1$ as explained earlier.

Perceptron Algorithm

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

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

$$y_i \ \boldsymbol{u} \cdot \boldsymbol{\phi}(x_i) \geq \gamma, \quad \forall i.$$

- radius of the data: $R = \max_i \|\boldsymbol{\phi}(x_i)\|$.

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

$$y_i \ \boldsymbol{u} \cdot \boldsymbol{\phi}(x_i) \geq \gamma, \quad \forall i.$$

- radius of the data: $R = \max_i \|\boldsymbol{\phi}(x_i)\|$.

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

Recall that $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + y_i \boldsymbol{\phi}(x_i)$.

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

$$\begin{aligned} \boldsymbol{u} \cdot \boldsymbol{w}^{(k+1)} &= \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + y_i \boldsymbol{u} \cdot \boldsymbol{\phi}(x_i) \\ &\geq \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + \gamma \\ &\geq k\gamma. \end{aligned}$$

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

Recall that $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + y_i \boldsymbol{\phi}(x_i)$.

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

$$\begin{aligned} \boldsymbol{u} \cdot \boldsymbol{w}^{(k+1)} &= \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + y_i \boldsymbol{u} \cdot \boldsymbol{\phi}(x_i) \\ &\geq \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + \gamma \\ &\geq k\gamma. \end{aligned}$$

  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).

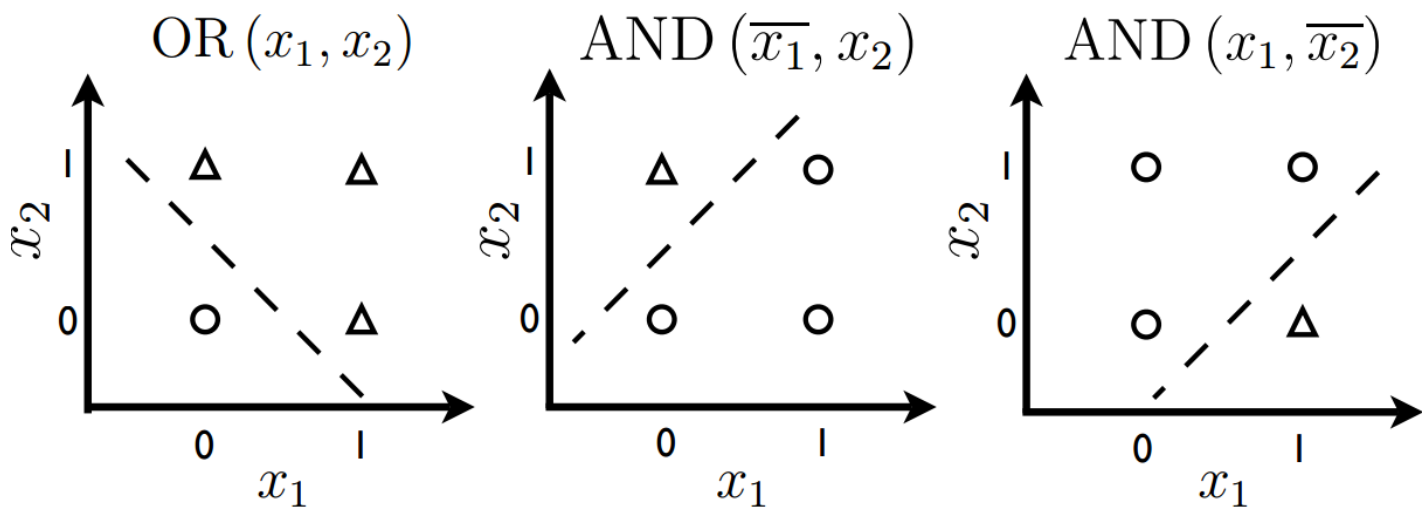
- **Upper bound on $\|\boldsymbol{w}^{(k+1)}\|$:**

$$\begin{aligned} \|\boldsymbol{w}^{(k+1)}\|^2 &= \|\boldsymbol{w}^{(k)}\|^2 + \|\boldsymbol{\phi}(x_i)\|^2 + 2y_i \boldsymbol{w}^{(k)} \cdot \boldsymbol{\phi}(x_i) \\ &\leq \|\boldsymbol{w}^{(k)}\|^2 + R^2 \\ &\leq kR^2. \end{aligned}$$

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

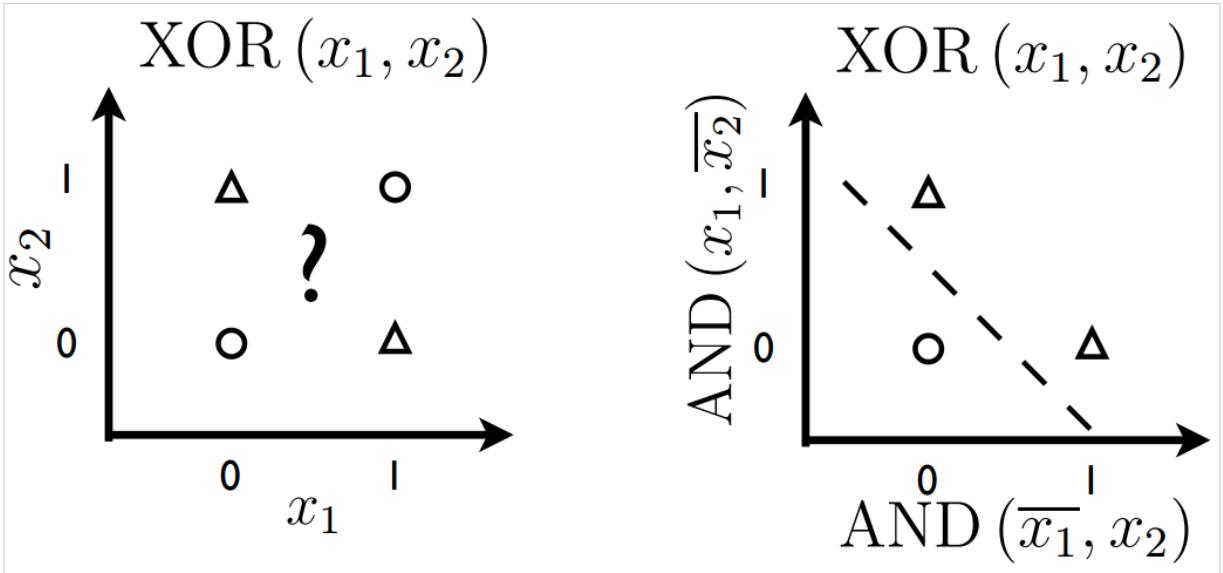
What a Simple Perceptron Can and Can't Do

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



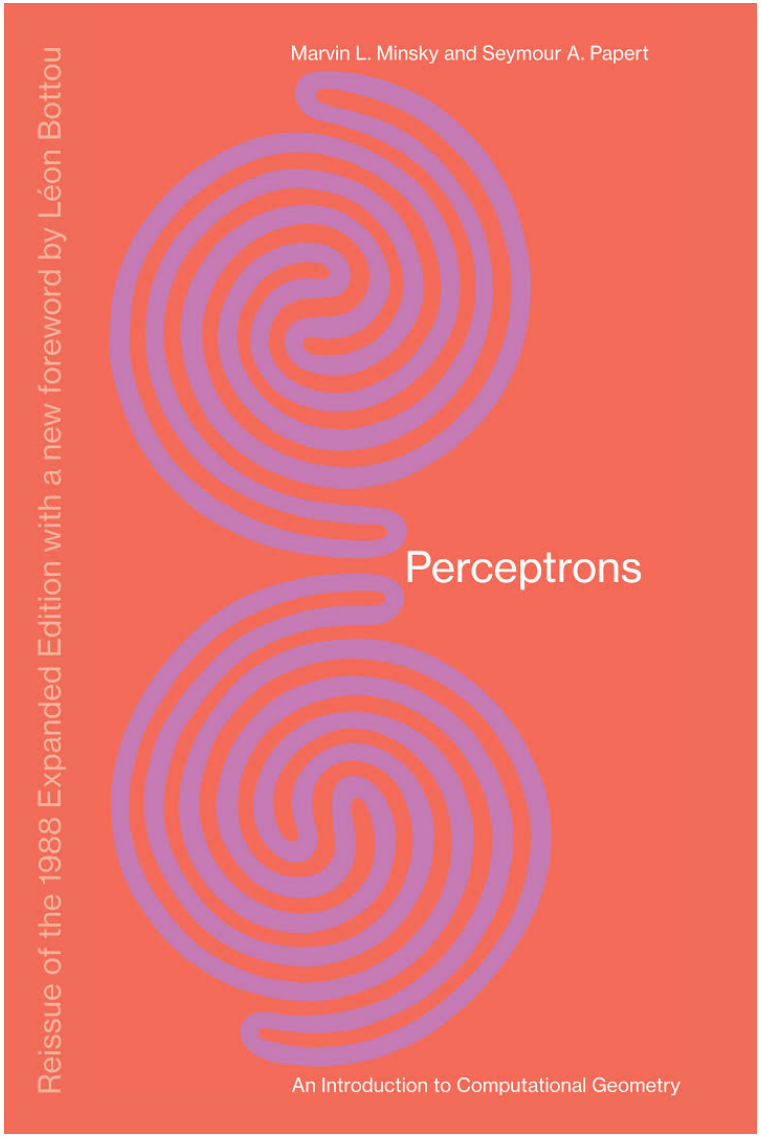
What a Simple Perceptron Can and Can't Do

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



- This result is often attributed to Minsky and Papert (1969) but was known well before.

Limitations of the Perceptron



Minsky and Papert (1969):

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

More tomorrow at Bhiksha's lecture!

Multi-Class Classification

Let's now assume a multi-class classification problem, with $|\mathcal{Y}| \geq 2$ labels (classes).

Reduction to Binary Classification

One strategy for multi-class classification is to train one binary classifier per label (using all the other classes as negative examples) and pick the class with the highest score (one-vs-all)

Another strategy is to train pairwise classifiers and to use majority voting (one-vs-one)

Here, we'll consider classifiers that tackle the multiple classes directly.

Multi-Class Linear Classifiers

- Parametrized by a weight matrix $\boldsymbol{W} \in \mathbb{R}^{|\mathcal{Y}| \times D}$ (one weight per feature/label pair) and a bias vector $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{Y}|}$:

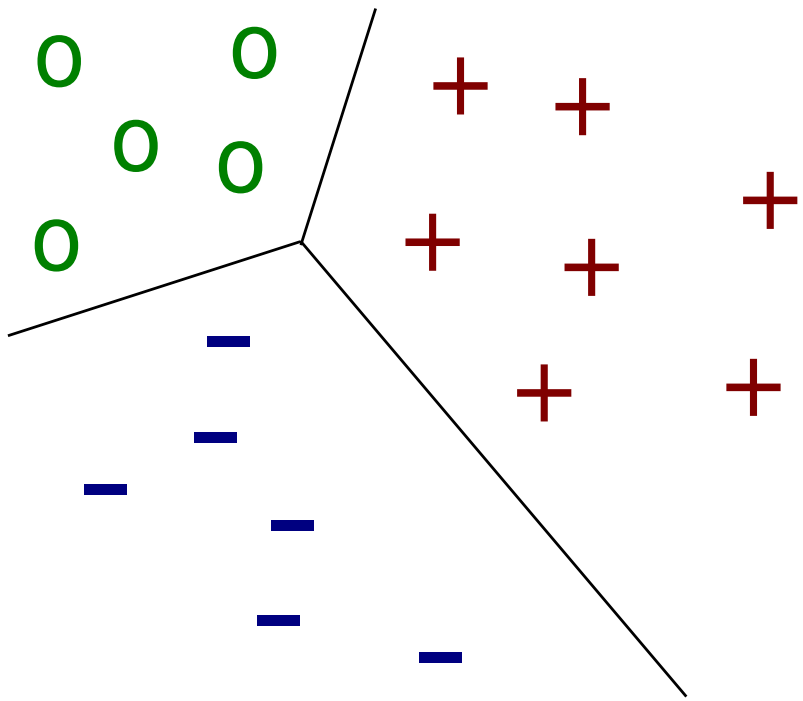
$$\boldsymbol{W} = \begin{bmatrix} \vdots \\ \boldsymbol{w}_y^\top \\ \vdots \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} \vdots \\ b_y \\ \vdots \end{bmatrix}.$$

- Equivalently, $|\mathcal{Y}|$ weight vectors $\boldsymbol{w}_y \in \mathbb{R}^D$ and scalars $b_y \in \mathbb{R}$
- The score (or probability) of a particular label is based on a linear combination of features and their weights
- Predict the $\widehat{y}$ which maximizes this score:

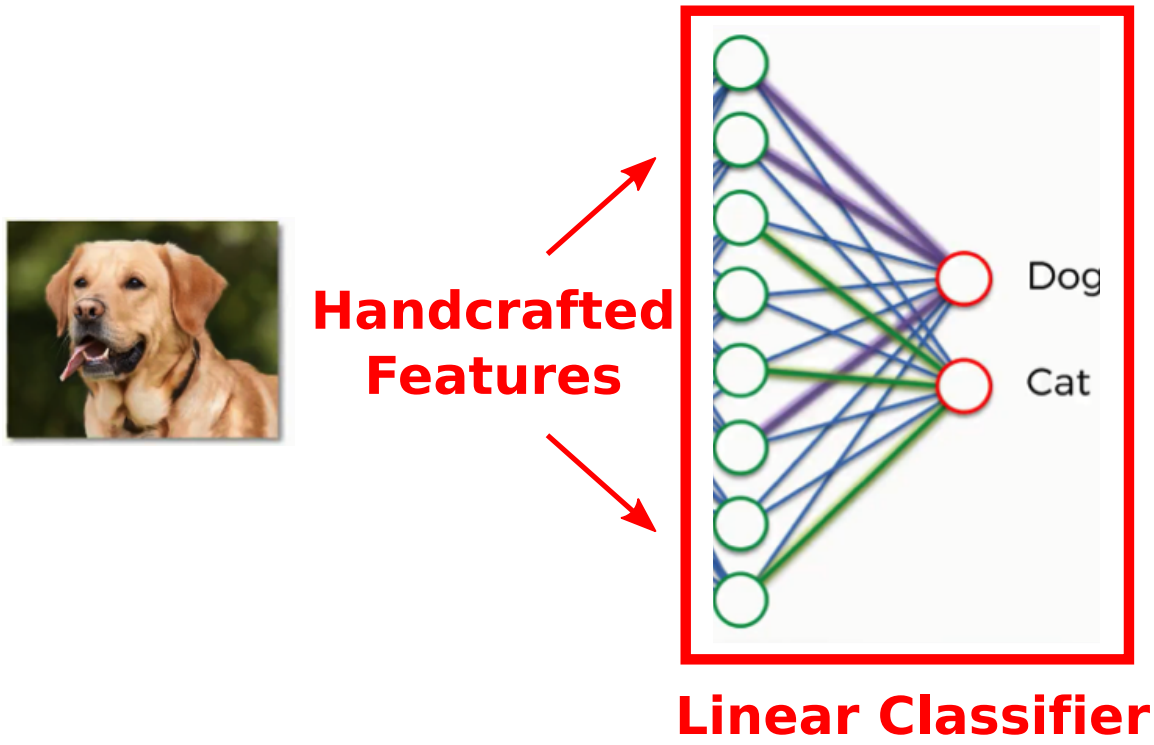
$$\widehat{y} = \arg\max_{y \in \mathcal{Y}} \ \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x) + b_y.$$

Multi-Class Linear Classifier

Geometrically, $(\boldsymbol{W}, \boldsymbol{b})$ split the feature space into regions delimited by hyperplanes.



Commonly Used Notation in Neural Networks



$$\widehat{y} = \text{argmax}\left(\boldsymbol{W}\boldsymbol{\phi}(x) + \boldsymbol{b}\right), \quad \boldsymbol{W} = \begin{bmatrix} \vdots \\ \boldsymbol{w}_y^\top \\ \vdots \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} \vdots \\ b_y \\ \vdots \end{bmatrix}.$$

Multi-Class Recovers Binary

With two classes ($\mathcal{Y} = \{\pm 1\}$), this formulation recovers the binary classifier presented earlier:

$$\widehat{y} = \arg \max_{y \in \{\pm 1\}} \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x) + b_y$$

Multi-Class Recovers Binary

With two classes ($\mathcal{Y} = \{\pm 1\}$), this formulation recovers the binary classifier presented earlier:

$$\begin{aligned}
\widehat{y} &= \arg \max_{y \in \{\pm 1\}} \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x) + b_y \\
&= \begin{cases} +1 & \text{if } \boldsymbol{w}_{+1} \cdot \boldsymbol{\phi}(x) + b_{+1} > \boldsymbol{w}_{-1} \cdot \boldsymbol{\phi}(x) + b_{-1} \\ -1 & \text{otherwise} \end{cases}
\end{aligned}$$

Multi-Class Recovers Binary

With two classes ($\mathcal{Y} = \{\pm 1\}$), this formulation recovers the binary classifier presented earlier:

$$\begin{aligned}
\widehat{y} &= \arg \max_{y \in \{\pm 1\}} \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x) + b_y \\
&= \begin{cases} +1 & \text{if } \boldsymbol{w}_{+1} \cdot \boldsymbol{\phi}(x) + b_{+1} > \boldsymbol{w}_{-1} \cdot \boldsymbol{\phi}(x) + b_{-1} \\ -1 & \text{otherwise} \end{cases} \\
&= \text{sign}(\underbrace{(\boldsymbol{w}_{+1} - \boldsymbol{w}_{-1})}_{\boldsymbol{w}} \cdot \boldsymbol{\phi}(x) + \underbrace{(b_{+1} - b_{-1})}_{b}).
\end{aligned}$$

Multi-Class Recovers Binary

With two classes ($\mathcal{Y} = \{\pm 1\}$), this formulation recovers the binary classifier presented earlier:

$$\begin{aligned}
\widehat{y} &= \arg \max_{y \in \{\pm 1\}} \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x) + b_y \\
&= \begin{cases} +1 & \text{if } \boldsymbol{w}_{+1} \cdot \boldsymbol{\phi}(x) + b_{+1} > \boldsymbol{w}_{-1} \cdot \boldsymbol{\phi}(x) + b_{-1} \\ -1 & \text{otherwise} \end{cases} \\
&= \text{sign}(\underbrace{(\boldsymbol{w}_{+1} - \boldsymbol{w}_{-1})}_{\boldsymbol{w}} \cdot \boldsymbol{\phi}(x) + \underbrace{(b_{+1} - b_{-1})}_{b}).
\end{aligned}$$

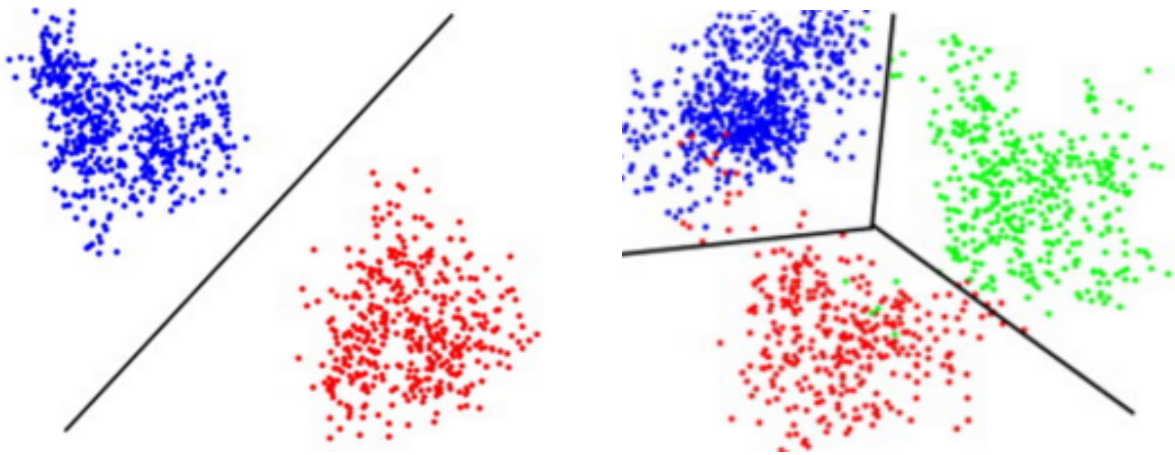
That is: only half of the parameters are needed.

Linear Classifiers (Binary vs Multi-Class)

- Prediction rule:

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

- 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 Classifier – No Bias Term

Again, it is common to omit the bias vector $\boldsymbol{b}$:

$$\widehat{y} = \arg\max_{y \in \mathcal{Y}} \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x) + b_y$$

Like before, this can be done without loss of generality, by assuming a constant feature $\phi_0(x) = 1$

The first column of $\boldsymbol{W}$ replaces the bias vector.

We assume this for simplicity.

Example: Perceptron

The perceptron algorithm also works for the multi-class case!

It has a similar mistake bound: if the data is separable, it's guaranteed to find separating hyperplanes!

Perceptron Algorithm: Multi-Class

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

Outline

Probabilistic Models

- For a moment, forget linear classifiers and parameter vectors $\boldsymbol{w}$
- Let's assume our goal is to model the conditional probability of output labels y given inputs x, i.e. $P(y|x)$
- If we can define this distribution, then classification becomes:

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

Bayes Rule

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

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

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

(since x is fixed!)

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

- Above is a "generative story": 'pick y; then pick x given y."
- Models that consider the joint $P(x, y)$ are called generative models, because they come with a generative story.

Naive Bayes

Assume that an input x is partitioned as $v_1, \ldots, v_L$, where $v_k \in \mathcal{V}_k$

Example:

- x is a document of length L
- v_k is the k^{th} token (a word)
- The set $\mathcal{V}_k = \mathcal{V}$ is a fixed vocabulary (all tokens drawn from $\mathcal{V}$)

Naive Bayes Assumption

(conditional independence)

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

Multinomial Naive Bayes

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

- All tokens are conditionally independently, given the topic
- The word order doesn't change $P(x, y)$ (bag-of-words assumption)

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

This is not realistic.

How to deal with variable length?

Multinomial Naive Bayes – Arbitrary Length

Solution: introduce a distribution over document length $P(|x|)$

- e.g. a Poisson distribution.

We get:

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

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

- the posterior $P(y|x)$ is the same as before.

What Does This Buy Us?

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

What do we gain with the Naive Bayes assumption?

What Does This Buy Us?

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

What do we gain with the Naive Bayes assumption?

- A huge reduction in the number of parameters!
- If we haven't done any factorization assumption, how many parameters would be required for expressing $P(v_1, \ldots, v_L|y)$?

What Does This Buy Us?

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

What do we gain with the Naive Bayes assumption?

- A huge reduction in the number of parameters!
- If we haven't done any factorization assumption, how many parameters would be required for expressing $P(v_1, \ldots, v_L|y)$? $O(|\mathcal{V}|^L)$
- And how many parameters with Naive Bayes?

What Does This Buy Us?

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

What do we gain with the Naive Bayes assumption?

- A huge reduction in the number of parameters!
- If we haven't done any factorization assumption, how many parameters would be required for expressing $P(v_1, \ldots, v_L|y)$? $O(|\mathcal{V}|^L)$
- And how many parameters with Naive Bayes? $O(|\mathcal{V}|)$

What Does This Buy Us?

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

What do we gain with the Naive Bayes assumption?

- A huge reduction in the number of parameters!
- If we haven't done any factorization assumption, how many parameters would be required for expressing $P(v_1, \ldots, v_L|y)$? $O(|\mathcal{V}|^L)$
- And how many parameters with Naive Bayes? $O(|\mathcal{V}|)$

Less parameters $\Longrightarrow$ Less computation; less risk of overfitting

(Though we may underfit if our independence assumptions are too strong.)

Naive Bayes – Learning

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

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

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

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

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

$$\widehat{\Theta} = \arg\max_{\Theta} \prod_{t=1}^{N} \left(P(y_t) \prod_{k=1}^{L} P(v_k(x_t)|y_t) \right)$$

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

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

$[[X]]$ 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:

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

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

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

Naive Bayes Example

Likelihoods: Count adjective v in class y / adjectives in y

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

$P(amazing\|positive)$	$= 2/10$	$P(amazing\|negative)$	$= 0/8$
$P(bad\|positive)$	$= 1/10$	$P(bad\|negative)$	$= 3/8$
$P(excellent\|positive)$	$= 1/10$	$P(excellent\|negative)$	$= 0/8$
$P(fantastic\|positive)$	$= 1/10$	$P(fantastic\|negative)$	$= 0/8$
$P(good\|positive)$	$= 1/10$	$P(good\|negative)$	$= 0/8$
$P(great\|positive)$	$= 1/10$	$P(great\|negative)$	$= 2/8$
$P(lovely\|positive)$	$= 1/10$	$P(lovely\|negative)$	$= 0/8$
$P(original\|positive)$	$= 0/10$	$P(original\|negative)$	$= 1/8$
$P(poor\|positive)$	$= 0/10$	$P(poor\|negative)$	$= 1/8$
$P(renowned\|positive)$	$= 1/10$	$P(renowned\|negative)$	$= 0/8$
$P(unimaginative\|positive)$	$= 0/10$	$P(unimaginative\|negative)$	$= 1/8$

Naive Bayes Example

Given a new segment to classify (**test time**):

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

Final decision

$$\widehat{y} = \arg\max_y \left(P(y) \prod_{k=1}^{L} P(v_k|y) \right)$$

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

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

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

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

So: *sentiment = positive*

Naive Bayes Example

Given a new segment to classify (**test time**):

Doc	Words	Class
9	Great plot, great cast, great everything	???

Final decision

$$P(positive) * P(great|positive) * P(great|positive) * P(great|positive)$$

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

$$P(negative) * P(great|negative) * P(great|negative) * P(great|negative)$$

$$4/7 * 2/8 * 2/8 * 2/8 = 0.00893$$

So: $sentiment = negative$

Naive Bayes Example

But if the new segment to classify (**test time**) is:

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):

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

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

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

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

Discriminative versus Generative

- Generative models attempt to model inputs and outputs
  - e.g., Naive Bayes = MLE of joint distribution $P(x, y)$
  - 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 $P(x, y)$
  - 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 $P(y|x)$ (or something related)
  - Logistic Regression – MLE of $P(y|x)$
  - 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 $P(x, y)$
  - 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 $P(y|x)$ (or something related)
  - Logistic Regression – MLE of $P(y|x)$
  - Perceptron and SVMs – minimize classification error

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

Coffee-break!

So far

We have covered:

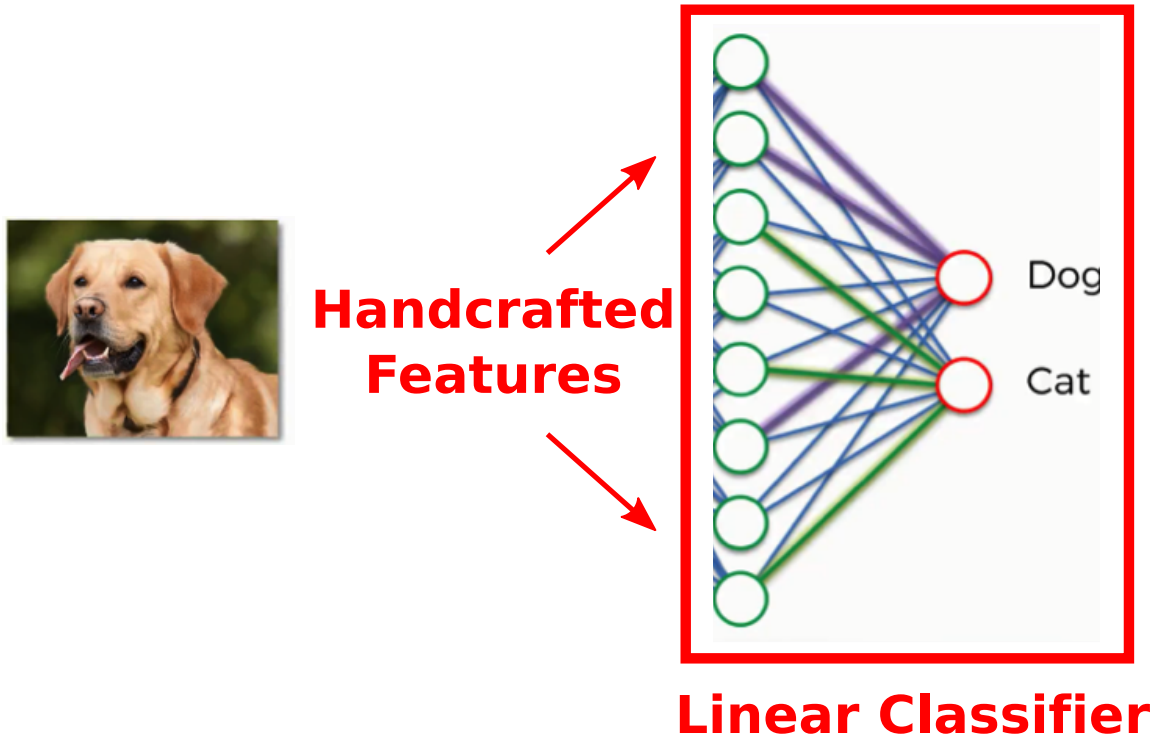
- The perceptron algorithm
- (Multinomial) Naive Bayes.

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.

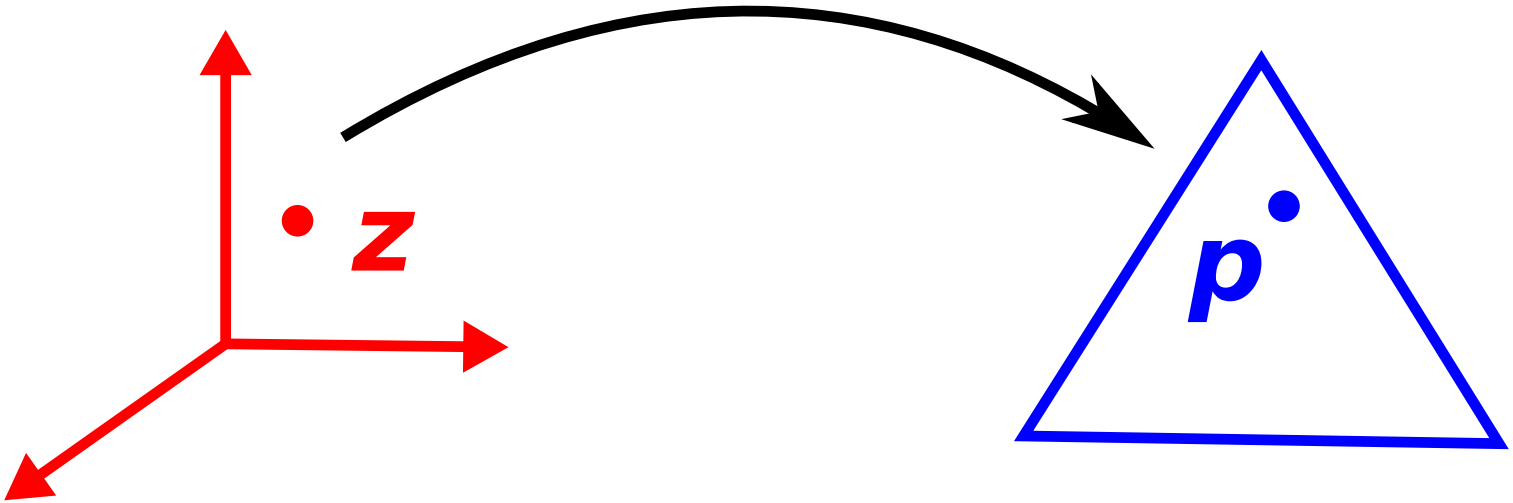
Reminder



$$\widehat{y} = \text{argmax}\left(\boldsymbol{W}\boldsymbol{\phi}(x) + \boldsymbol{b}\right), \quad \boldsymbol{W} = \begin{bmatrix} \vdots \\ \boldsymbol{w}_y^\top \\ \vdots \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} \vdots \\ b_y \\ \vdots \end{bmatrix}.$$

Key Problem

How to map from a set of label scores $\mathbb{R}^{|\mathcal{Y}|}$ to a probability distribution over $\mathcal{Y}$?



We'll see two mappings: softmax (next) and sparsemax (later).

Outline

Logistic Regression

Recall: a linear model gives the score for each class, $\boldsymbol{w}_y \cdot \boldsymbol{\phi}(x)$.

Define a conditional probability:

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

This operation (exponentiating and normalizing) is called the softmax transformation (more later!)

Note: still a linear classifier

$$\begin{aligned}
\arg\max_y \; P(y|x) &= \arg\max_y \; \frac{\exp(\boldsymbol{w}_y \cdot \boldsymbol{\phi}(x))}{Z_x} \\
&= \arg\max_y \; \exp(\boldsymbol{w}_y \cdot \boldsymbol{\phi}(x)) \\
&= \arg\max_y \; \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x)
\end{aligned}$$

Binary Logistic Regression

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

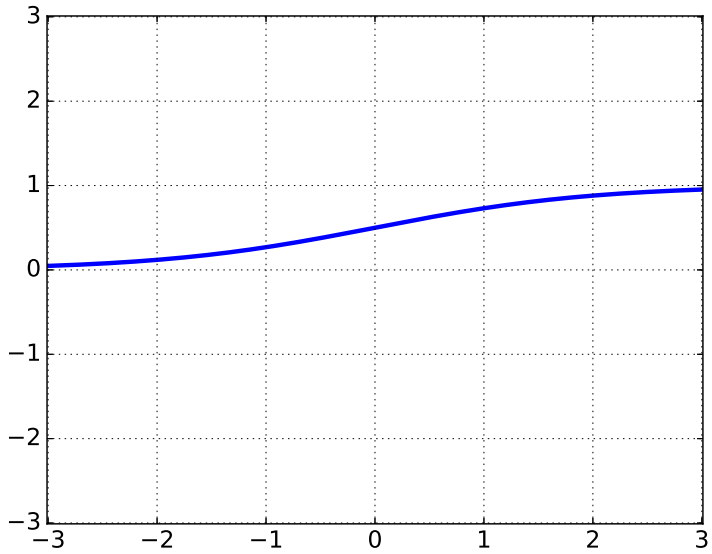
Scores: 0 for negative class, $\boldsymbol{w} \cdot \boldsymbol{\phi}(x)$ for positive class

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

This is called a sigmoid transformation (more later!)

Sigmoid Transformation

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



- Widely used in neural networks (wait for tomorrow!)
- 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

$$P_{\boldsymbol{W}}(y \mid x) = \frac{\exp(\boldsymbol{w}_y \cdot \boldsymbol{\phi}(x))}{Z_x}$$

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

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

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, ...)

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, ...)

- Logistic Regression = Maximum Entropy: maximize entropy subject to constraints on features
- Proof left as an exercise!

Recap: Convex functions

Pro: Guarantee of a global minima ✓

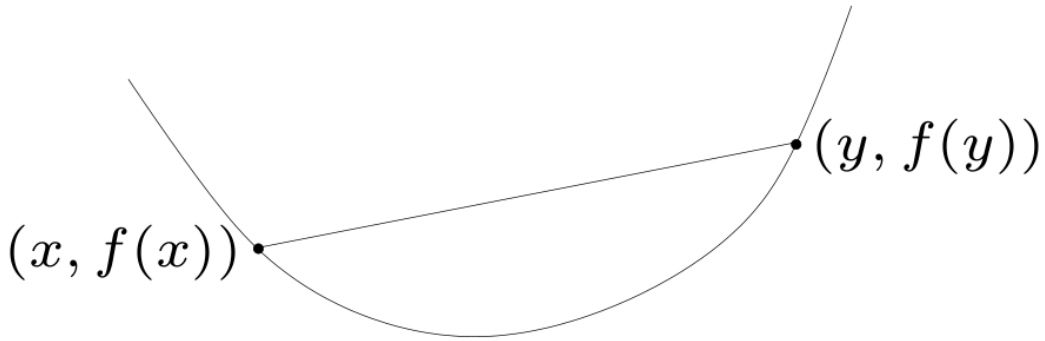


Figure: Illustration of a convex function. The line segment between any two points on the graph lies entirely above the curve.

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: $x^{(k+1)} \leftarrow x^{(k)} - \eta_k \nabla f(x^{(k)})$

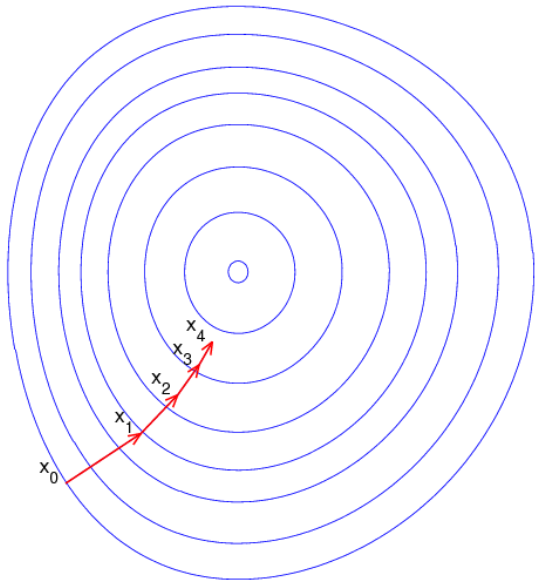


Figure: Illustration of gradient descent. The red lines correspond to steps taken in the negative gradient direction.

Gradient Descent

- Our loss function in logistic regression is

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

- We want to find $\arg\min_{\boldsymbol{W}} \sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t))$
  - Set $\boldsymbol{W}^0 = \boldsymbol{0}$
  - Iterate until convergence (for suitable stepsize η_k):

$$\begin{aligned} \boldsymbol{W}^{k+1} &= \boldsymbol{W}^k - \eta_k \nabla_{\boldsymbol{W}} \left(\sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t)) \right) \\ &= \boldsymbol{W}^k - \eta_k \sum_{t=1}^{N} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}^k; (x_t, y_t)) \end{aligned}$$

- $\nabla_{\boldsymbol{W}} L(\boldsymbol{W})$ is gradient of L w.r.t. $\boldsymbol{W}$
- $L(\boldsymbol{W})$ convex $\Rightarrow$ gradient descent will reach the global optimum $\boldsymbol{W}$.

Stochastic Gradient Descent

It turns out this works with a Monte Carlo approximation of the gradient (more frequent updates, convenient with large datasets):

- Set $\boldsymbol{W}^0 = \mathbf{0}$
- Iterate until convergence
  - Pick (x_t, y_t) randomly
  - Update $\boldsymbol{W}^{k+1} = \boldsymbol{W}^k - \eta_k \nabla_{\boldsymbol{W}} L(\boldsymbol{W}^k; (x_t, y_t))$
- 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 compute $\nabla_{\boldsymbol{W}} L(\boldsymbol{W}; (x_t, y_t))$, where

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

- Some reminders:
  1. $\nabla_{\boldsymbol{W}} \log F(\boldsymbol{W}) = \frac{1}{F(\boldsymbol{W})} \nabla_{\boldsymbol{W}} F(\boldsymbol{W})$
  2. $\nabla_{\boldsymbol{W}} \exp F(\boldsymbol{W}) = \exp(F(\boldsymbol{W})) \nabla_{\boldsymbol{W}} F(\boldsymbol{W})$

- We denote by

$$\boldsymbol{e}_y = [0, \ldots, 0, \underbrace{1}_{y}, 0, \ldots, 0]^\top$$

  the one-hot vector representation of class y.

Computing the Gradient

$$
\begin{aligned}
\nabla_{\boldsymbol{W}} L(\boldsymbol{W};(x,y)) &= \nabla_{\boldsymbol{W}} \left(\log \sum_{y'} \exp(\boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x)) - \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x) \right) \\
&= \nabla_{\boldsymbol{W}} \log \sum_{y'} \exp(\boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x)) - \nabla_{\boldsymbol{W}} \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x) \\
&= \frac{1}{\sum_{y'} \exp(\boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x))} \sum_{y'} \nabla_{\boldsymbol{W}} \exp(\boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x)) - \boldsymbol{e}_y \boldsymbol{\phi}(x)^\top \\
&= \frac{1}{Z_x} \sum_{y'} \exp(\boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x)) \nabla_{\boldsymbol{W}} \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x) - \boldsymbol{e}_y \boldsymbol{\phi}(x)^\top \\
&= \sum_{y'} \frac{\exp(\boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x))}{Z_x} \boldsymbol{e}_{y'} \boldsymbol{\phi}(x)^\top - \boldsymbol{e}_y \boldsymbol{\phi}(x)^\top \\
&= \sum_{y'} P_{\boldsymbol{W}}(y'|x) \boldsymbol{e}_{y'} \boldsymbol{\phi}(x)^\top - \boldsymbol{e}_y \boldsymbol{\phi}(x)^\top \\
&= \left(\begin{bmatrix} \vdots \\ P_{\boldsymbol{W}}(y'|x) \\ \vdots \end{bmatrix} - \boldsymbol{e}_y \right) \boldsymbol{\phi}(x)^\top .
\end{aligned}
$$

Logistic Regression Summary

- Define conditional probability

$$P_{\boldsymbol{W}}(y|x) = \frac{\exp(\boldsymbol{w}_y \cdot \boldsymbol{\phi}(x))}{Z_x}$$

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

$$\boldsymbol{W} = \arg\max_{\boldsymbol{W}} \sum_t \log P_{\boldsymbol{W}}(y_t|x_t) = \arg\min_{\boldsymbol{W}} \sum_t L(\boldsymbol{W}; (x_t, y_t))$$

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

$$\nabla_{\boldsymbol{W}} L(\boldsymbol{W}; (x, y)) = \sum_{y'} P_{\boldsymbol{W}}(y'|x) \boldsymbol{e}_{y'} \boldsymbol{\phi}(x)^\top - \boldsymbol{e}_y \boldsymbol{\phi}(x)^\top$$

The Story So Far

- Naive Bayes is generative: maximizes joint likelihood
  - closed form solution (boils down to counting and normalizing)
- Logistic regression is discriminative: maximizes conditional likelihood
  - also called log-linear model and max-entropy classifier
  - no closed form solution
  - stochastic gradient updates look like

$$\boldsymbol{W}^{k+1} = \boldsymbol{W}^k + \eta \left(\boldsymbol{e}_y \boldsymbol{\phi}(x)^\top - \sum_{y'} P_{\boldsymbol{w}}(y'|x) \boldsymbol{e}_{y'} \boldsymbol{\phi}(x)^\top \right)$$

- Perceptron is a discriminative, non-probabilistic classifier
  - perceptron's updates look like

$$\boldsymbol{W}^{k+1} = \boldsymbol{W}^k + \boldsymbol{e}_y \boldsymbol{\phi}(x)^\top - \boldsymbol{e}_{\hat{y}} \boldsymbol{\phi}(x)^\top$$

SGD updates for logistic regression and perceptron's updates look similar!

Maximizing Margin

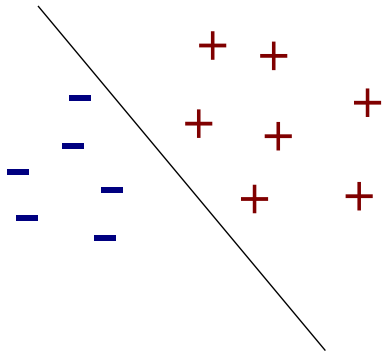
- For a training set $\mathcal{D}$
- Margin of a weight matrix $\boldsymbol{W}$ is smallest γ such that

$$\boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq \gamma$$

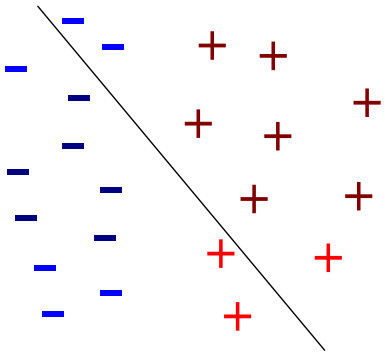
- for every training instance $(x_t, y_t) \in \mathcal{D}$, $y' \in \mathcal{Y}$

Margin

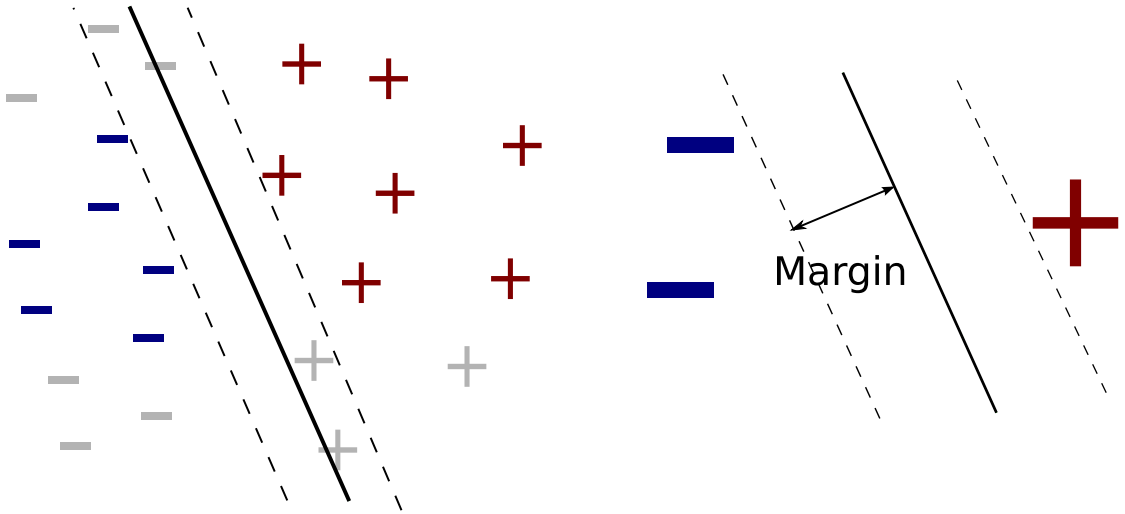
Training



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 $\gamma > 0$

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

such that:

$$\boldsymbol{u}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{u}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq \gamma$$

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

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

- Note: the solution still ensures a separating hyperplane if there is one (zero training error) – due to the hard constraint

Maximizing Margin

Let $\gamma > 0$

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

such that:

$$\boldsymbol{u}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{u}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq \gamma$$

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

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

- Note: the solution still ensures a separating hyperplane if there is one (zero training error) – due to the hard constraint
- We fix $||\boldsymbol{U}|| = 1$ since scaling $\boldsymbol{U}$ to increase $\|\boldsymbol{U}\|$ trivially produces larger margin

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

$$\boldsymbol{u}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{u}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq \gamma$$

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

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

=

Min Norm:

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

such that:

$$\boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq 1$$

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

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

- Instead of fixing $||\boldsymbol{U}||$ we fix the margin to 1

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

$$\boldsymbol{u}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{u}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq \gamma$$

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

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

$=$

Min Norm:

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

such that:

$$\boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq 1$$

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

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

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

Support Vector Machines

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

such that:

$$\boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq 1$$

$$\forall (x_t, y_t) \in \mathcal{D} \text{ and } y' \in \mathcal{Y}$$

- Quadratic programming problem – a well known convex optimization problem
- Can be solved with many techniques.

Support Vector Machines

What if data is not separable?

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

such that:

$$\boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq 1 - \xi_t \text{ and } \xi_t \geq 0$$

$$\forall (x_t, y_t) \in \mathcal{D} \text{ and } y' \in \mathcal{Y}$$

ξ_t: trade-off between margin violations per example and $\|\boldsymbol{W}\|$
Larger C = more examples correctly classified, but smaller margin.

Kernels

Historically, SVMs with kernels co-ocurred together and were extremely popular

Can "kernelize" algorithms to make them non-linear (not only SVMs, but also logistic regression, perceptron, ...)

More later.

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:

$$\boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq 1 - \xi_t \quad \forall y' \neq y_t$$

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:

$$\boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \max_{y' \neq y_t} \ \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) \geq 1 - \xi_t$$

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:

$$\xi_t \geq 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t)$$

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:

$$\xi_t \geq 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t)$$

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:

$$\xi_t \geq 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t)$$

If $\boldsymbol{W}$ classifies (x_t, y_t) with margin 1, penalty $\xi_t = 0$
Otherwise penalty $\xi_t = 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t)$

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:

$$\xi_t \geq 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t)$$

If $\boldsymbol{W}$ classifies (x_t, y_t) with margin 1, penalty $\xi_t = 0$
Otherwise penalty $\xi_t = 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t)$

Hinge loss:

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

Support Vector Machines

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

such that:

$$\xi_t \geq 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t)$$

Hinge loss equivalent:

$$\boldsymbol{W} = \arg\min_{\boldsymbol{W}} \left(\sum_{t=1}^{N} \underbrace{\max\left(0, 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t)\right)}_{L(\boldsymbol{W};(x_t,y_t))} \right) + \frac{\lambda}{2}||\boldsymbol{W}||^2$$

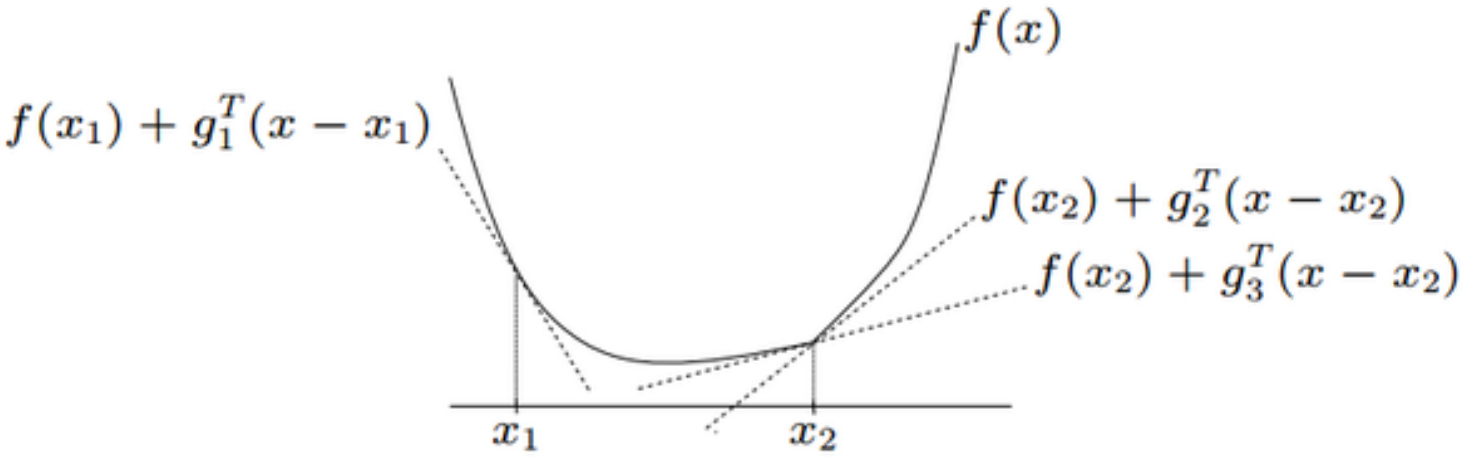
From Gradient to Subgradient

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

Cannot use gradient descent

But... can use subgradient descent (almost the same)!

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

$$
\begin{aligned}
L(\boldsymbol{W};(x,y)) &= \max\left(0, 1 + \max_{y' \neq y} \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x) - \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x)\right) \\
&= \left(\max_{y' \in \mathcal{Y}} \boldsymbol{w}_{y'} \cdot \boldsymbol{\phi}(x) + [[y' \neq y]]\right) - \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x)
\end{aligned}
$$

A subgradient of the hinge is

$$
\tilde{\nabla}_{\boldsymbol{W}} L(\boldsymbol{W};(x,y)) \ni \boldsymbol{e}_{\widehat{y}} \boldsymbol{\phi}(x)^\top - \boldsymbol{e}_y \boldsymbol{\phi}(x)^\top
$$

where

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

Can also train SVMs with (stochastic) sub-gradient descent!

Perceptron and Hinge-Loss

SVM subgradient update looks like perceptron update

$$\boldsymbol{W}^{k+1} = \boldsymbol{W}^{k} - \eta \begin{cases} 0, & \text{if } \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \max_{y \neq y_t} \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x_t) \geq 1 \\ \boldsymbol{e}_y \boldsymbol{\phi}(x_t)^\top - \boldsymbol{e}_{y_t} \boldsymbol{\phi}(x_t)^\top, & \text{otherwise, where } y = \arg\max_y \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x_t) + [[y \neq y_t]] \end{cases}$$

Perceptron

$$\boldsymbol{W}^{k+1} = \boldsymbol{W}^{k} - \eta \begin{cases} 0, & \text{if } \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t) - \max_{y} \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x_t) \geq 0 \\ \boldsymbol{e}_y \boldsymbol{\phi}(x_t)^\top - \boldsymbol{e}_{y_t} \boldsymbol{\phi}(x_t)^\top, & \text{otherwise, where } y = \arg\max_y \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x_t) \end{cases}$$

where $\eta = 1$

Perceptron = SGD with no-margin hinge-loss

$$\max\left(0, 1 + \max_{y \neq y_t} \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t)\right)$$

Summary

What we have covered

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

What is next

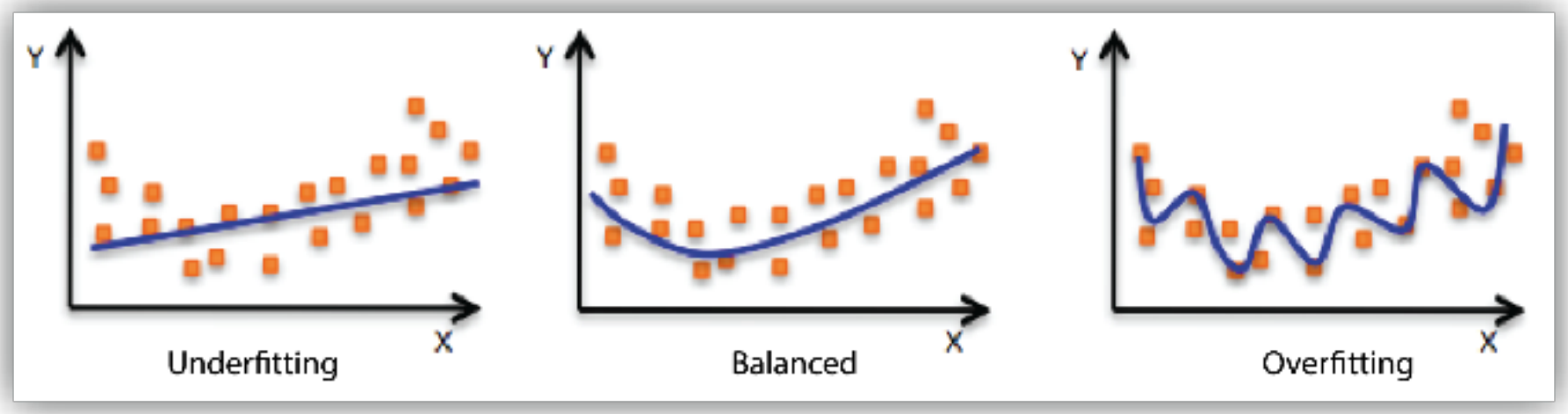
- Regularization
- Softmax and sparsemax
- Non-linear classifiers

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!

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.

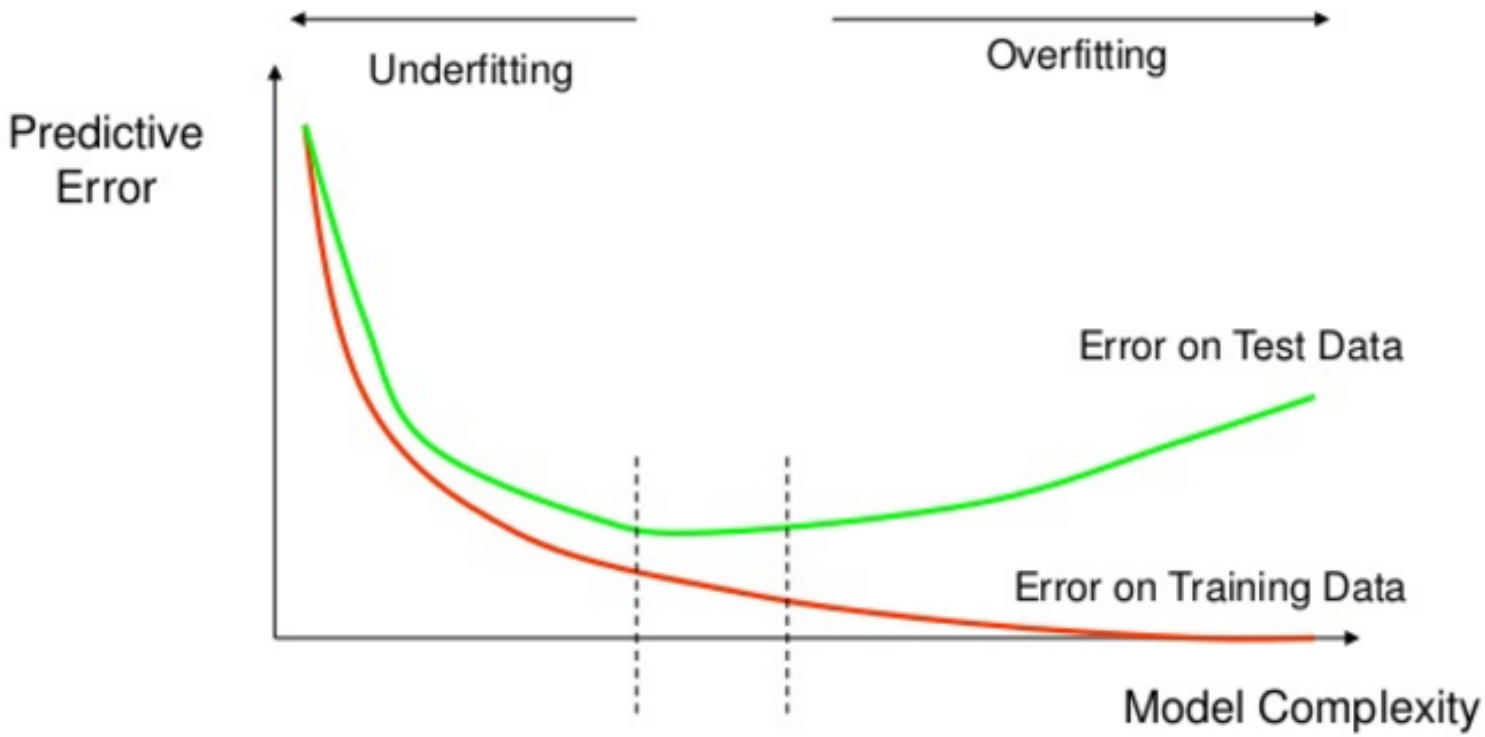
- Gaussian prior (ℓ_2), promotes smaller weights:

$$\Omega(\boldsymbol{W}) = \|\boldsymbol{W}\|_2^2 = \sum_y \|\boldsymbol{w}_y\|_2^2 = \sum_y \sum_j w_{y,j}^2.$$

- Laplacian prior (ℓ_1), promotes sparse weights!

$$\Omega(\boldsymbol{W}) = \|\boldsymbol{W}\|_1 = \sum_y \|\boldsymbol{w}_y\|_1 = \sum_y \sum_j |w_{y,j}|$$

Empirical Risk Minimization



Logistic Regression with ℓ_2 Regularization

$$\sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t)) + \lambda\Omega(\boldsymbol{W}) = -\sum_{t=1}^{N} \log\left(\exp(\boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t))/Z_x\right) + \frac{\lambda}{2}\|\boldsymbol{W}\|^2$$

- What is the new gradient?

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

- We know $\nabla_{\boldsymbol{W}} L(\boldsymbol{W}; (x_t, y_t))$
- 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!

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

$\uparrow$ SVM optimization $\uparrow$

SVMs vs. Logistic Regression

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

- SVMs/hinge-loss:

$$L(\boldsymbol{W}; (x_t, y_t)) = \max\,(0, 1 + \max_{y \neq y_t}\,(\boldsymbol{w}_y \cdot \boldsymbol{\phi}(x_t) - \boldsymbol{w}_{y_t} \cdot \boldsymbol{\phi}(x_t))), \qquad \Omega(\boldsymbol{W}) = \frac{1}{2}\|\boldsymbol{W}\|^2$$

- Logistic Regression/log-loss:

$$L(\boldsymbol{W}; (x_t, y_t)) = -\log\,\left(\exp(\boldsymbol{w} \cdot \boldsymbol{\psi}(x_t, y_t))/Z_x\right), \qquad \Omega(\boldsymbol{W}) = \frac{1}{2}\|\boldsymbol{W}\|^2.$$

Loss Function

Should match as much as possible the metric we want to optimize at test time

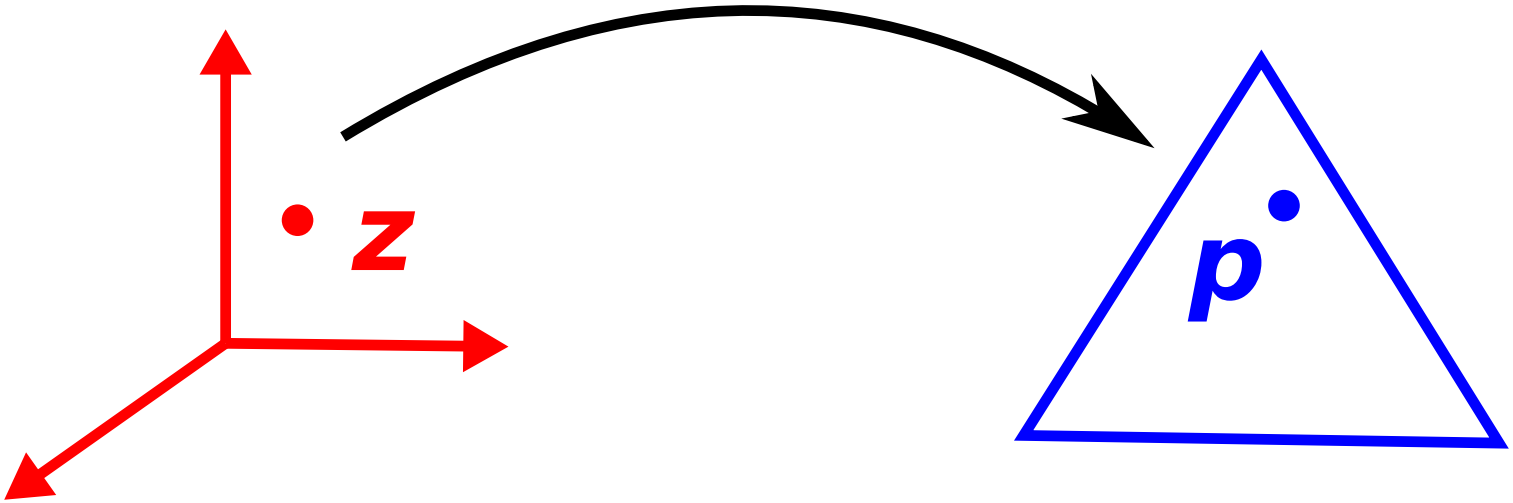
Should be well-behaved (continuous, maybe smooth) to be amenable to optimization (this rules out the 0/1 loss)

Some examples:

- Squared loss for regression
- Negative log-likelihood (cross-entropy): multinomial logistic regression
- Hinge loss: support vector machines
- Sparsemax loss for multi-class and multi-label classification (next)

Recap

How to map from a set of label scores $\mathbb{R}^{|\mathcal{Y}|}$ to a probability distribution over $\mathcal{Y}$?



We already saw one example: softmax.

Next: sparsemax.

Recap: Softmax Transformation

The typical transformation for multi-class classification is **softmax** : $\mathbb{R}^{|\mathcal{Y}|} \rightarrow \Delta^{|\mathcal{Y}|-1}$:

$$\boxed{\textbf{softmax}(\boldsymbol{z}) = \left[\frac{\exp(z_1)}{\sum_c \exp(z_c)}, \ldots, \frac{\exp(z_{|\mathcal{Y}|})}{\sum_c \exp(z_c)}\right]}$$

- Underlies multinomial logistic regression!
- Strictly positive, sums to 1
- Resulting distribution has full support: $\textbf{softmax}(\boldsymbol{z}) > \mathbf{0}, \forall \boldsymbol{z}$
- A disadvantage if a *sparse* probability distribution is desired
- Common workaround: threshold and truncate

Sparsemax (Martins and Astudillo, 2016)

A sparse-friendly alternative is **sparsemax** : $\mathbb{R}^{|\mathcal{Y}|} \to \Delta^{|\mathcal{Y}|-1}$, defined as:

$$\boxed{\textbf{sparsemax}(\boldsymbol{z}) := \arg\min_{\boldsymbol{p} \in \Delta^{|\mathcal{Y}|-1}} \|\boldsymbol{p} - \boldsymbol{z}\|^2.}$$

- In words: Euclidean projection of $\boldsymbol{z}$ onto the probability simplex
- Likely to hit the boundary of the simplex, in which case **sparsemax**$(\boldsymbol{z})$ becomes sparse (hence the name)
- Retains many of the properties of softmax (e.g. differentiability), having in addition the ability of producing sparse distributions
- Projecting onto the simplex amounts to a soft-thresholding operation
- Efficient linear time forward/backward propagation (see paper)

Sparsemax in Closed Form

- Projecting onto the simplex amounts to a soft-thresholding operation:

$$\textbf{sparsemax}_i(\boldsymbol{z}) \;=\; \max\{0, z_i - \tau\}$$

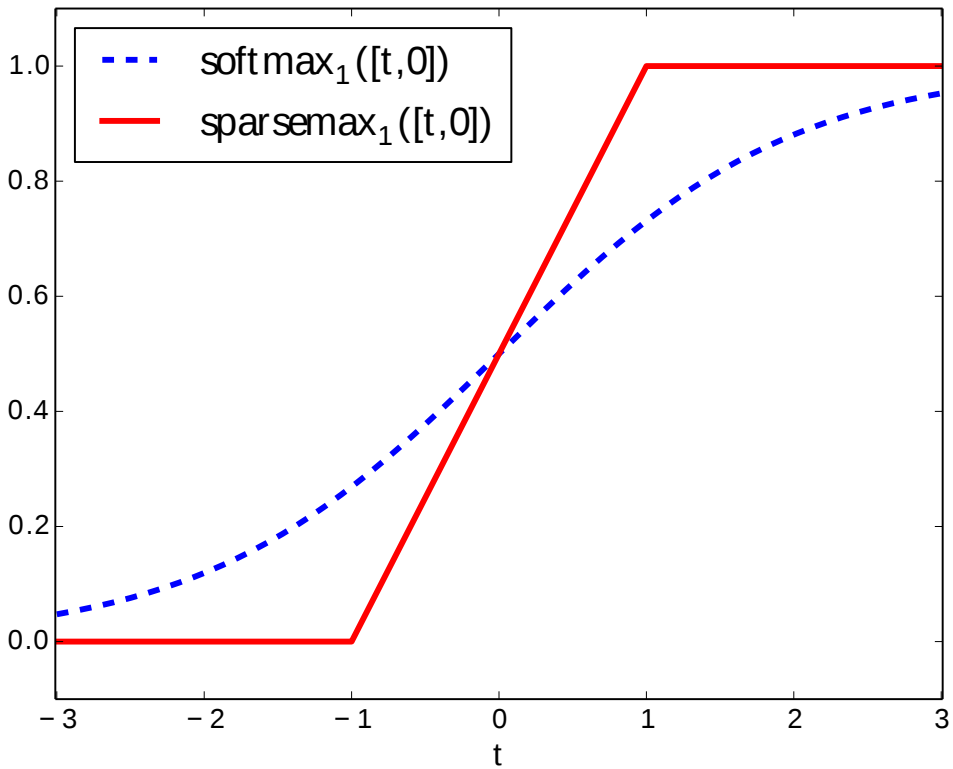
  where τ is a normalizing constant such that $\sum_j \max\{0, z_j - \tau\} = 1$
- To evaluate the sparsemax, all we need is to compute τ
- Coordinates above the threshold will be shifted by this amount; the others will be truncated to zero

Two Dimensions

- Parametrize $z = (t, 0)$
- The 2D **softmax** is the logistic (sigmoid) function:

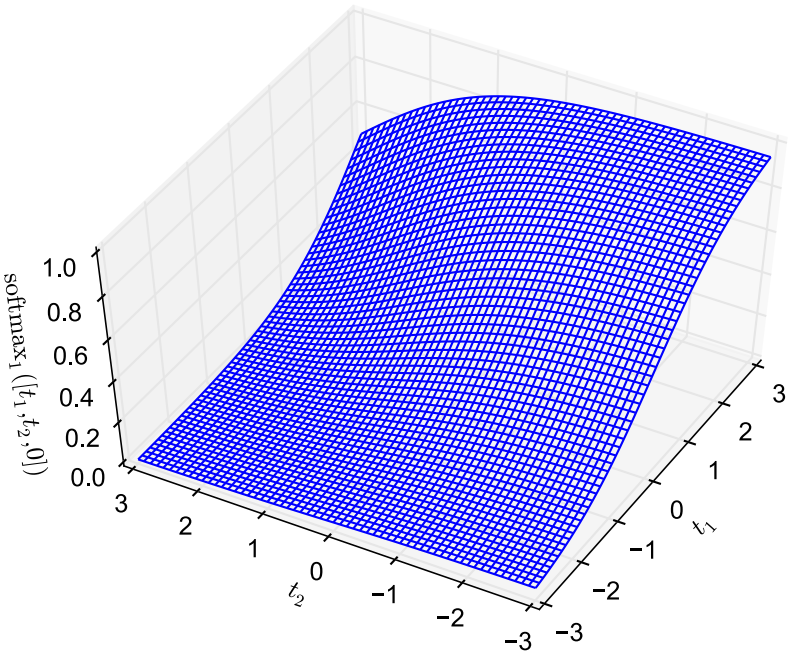
$$\textbf{softmax}_1(z) = (1 + \exp(-t))^{-1}$$

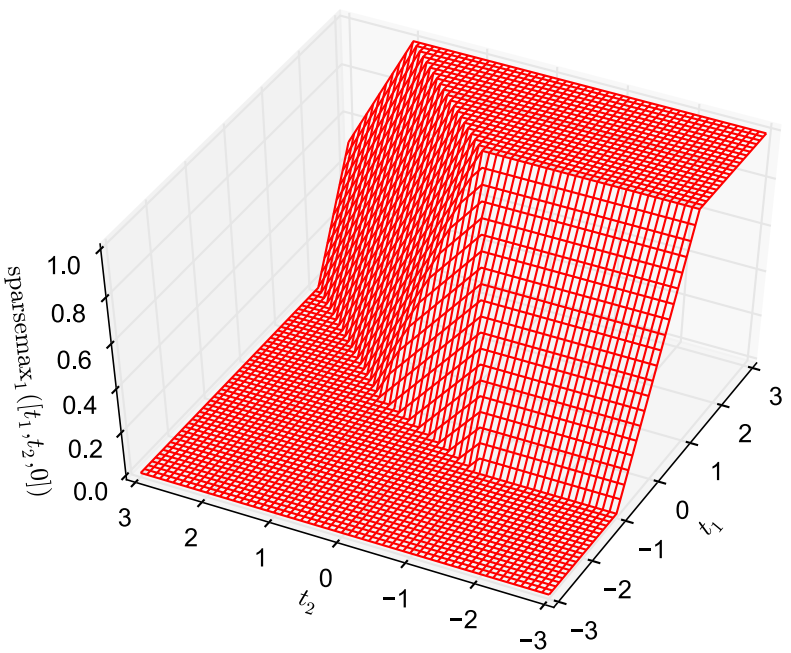
- The 2D **sparsemax** is the "hard" version of the sigmoid:



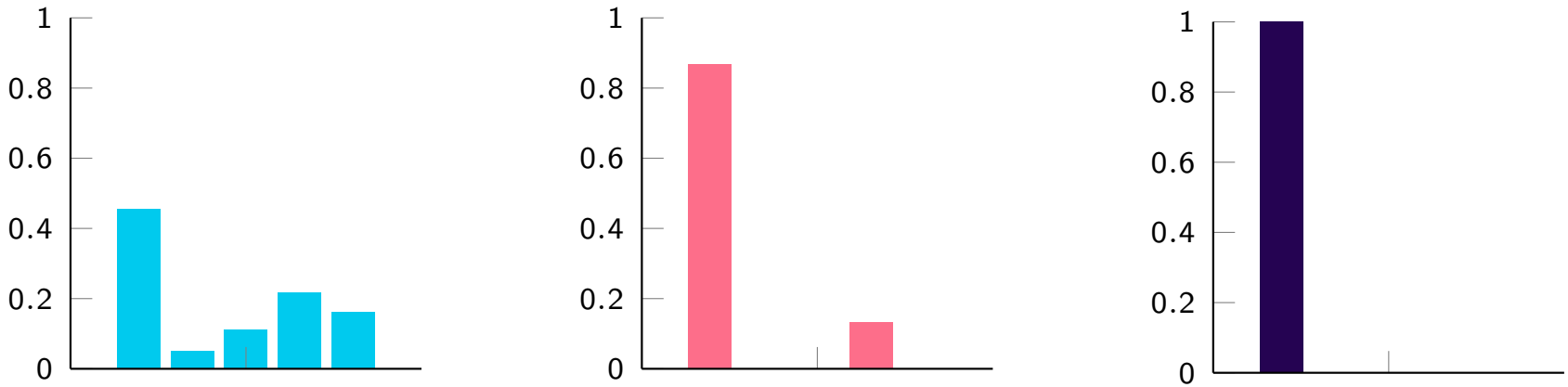
Three Dimensions

- Parameterize $z = (t_1, t_2, 0)$ and plot $\textbf{softmax}_1(z)$ and $\textbf{sparsemax}_1(z)$ as a function of t_1 and t_2
- **sparsemax** is piecewise linear, but asymptotically similar to **softmax**





softmax(z) **sparsemax**(z) argmax(z)



(Same $\boldsymbol{z} = [1.0716, -1.1221, -0.3288, 0.3368, 0.0425]$)

- Sparsemax is in-between softmax and argmax
- It is **sparse** and **differentiable**.

Loss Function

How to use sparsemax as a loss function?

Caveat: sparsemax is sparse and we don't want to take the log of zero...

Recap: Multinomial Logistic Regression

- The common choice for a softmax output layer
- The classifier estimates $P(y = c \mid x; \boldsymbol{W})$
- We minimize the negative log-likelihood:

$$\begin{aligned} L(\boldsymbol{W}; (x, y)) &= -\log P(y \mid x; \boldsymbol{W}) \\ &= -\log \left[\mathbf{softmax}(\boldsymbol{z}(x))\right]_y, \end{aligned}$$

  where $z_c(x) = \boldsymbol{w}_c \cdot \boldsymbol{\phi}(x)$ is the score of class c.
- Loss gradient:

$$\nabla_{\boldsymbol{W}} L((x, y); \boldsymbol{W}) = -\left(\boldsymbol{e}_y \boldsymbol{\phi}(x)^\top - \mathbf{softmax}(\boldsymbol{z}(x))\boldsymbol{\phi}(x)^\top\right)$$

Sparsemax Loss (Martins and Astudillo, 2016)

- The natural choice for a sparsemax output layer
- The neural network estimates $P(y \mid x; \boldsymbol{W})$ as a sparse distribution
- The sparsemax loss is

$$L((x, y); \boldsymbol{W}) = -z_y(x) + \frac{1}{2} - \frac{1}{2}\|\,\textbf{sparsemax}(\boldsymbol{z}(x))\|^2 + \boldsymbol{z}(x)^\top\,\textbf{sparsemax}(\boldsymbol{z}(x)),$$

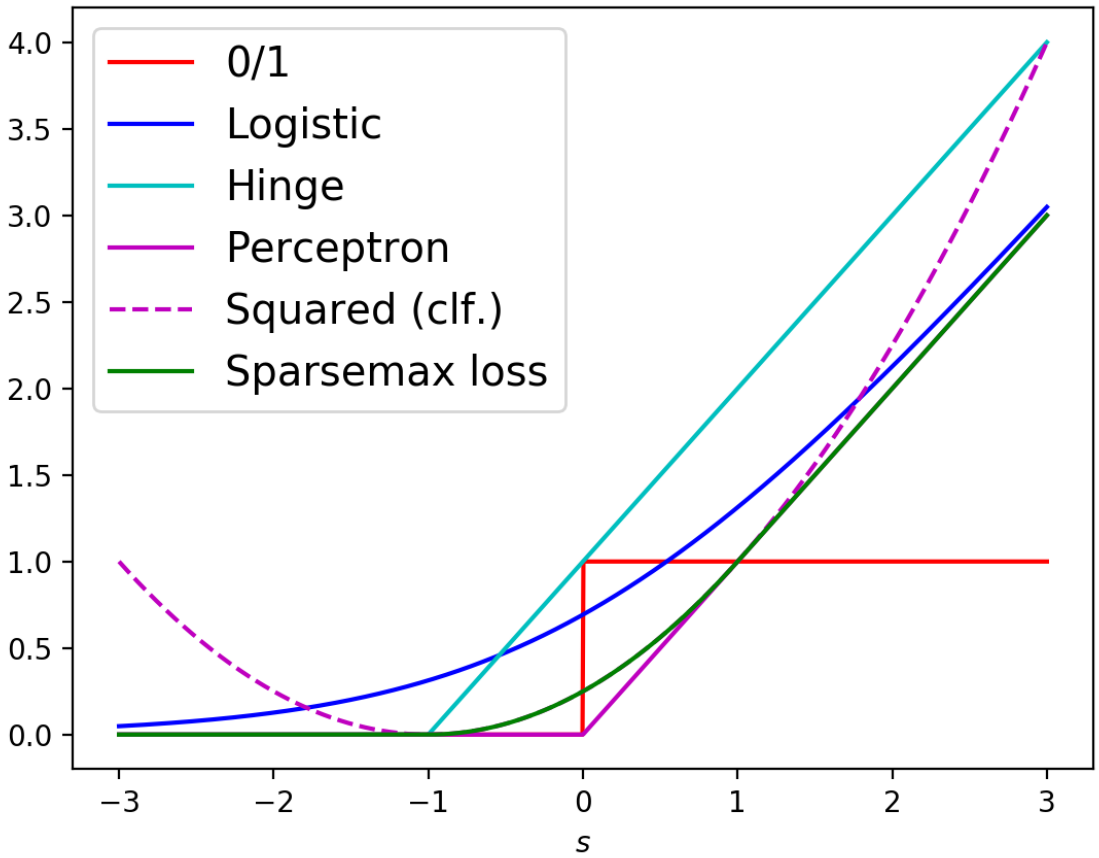
  where $z_y(x) = \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x)$.

- Loss gradient:

$$\nabla_{\boldsymbol{W}} L((x, y); \boldsymbol{W}) = -\left(\boldsymbol{e}_y \boldsymbol{\phi}(x)^\top - \textbf{sparsemax}(\boldsymbol{z}(x))\boldsymbol{\phi}(x)^\top\right)$$

Classification Losses (Binary Case)

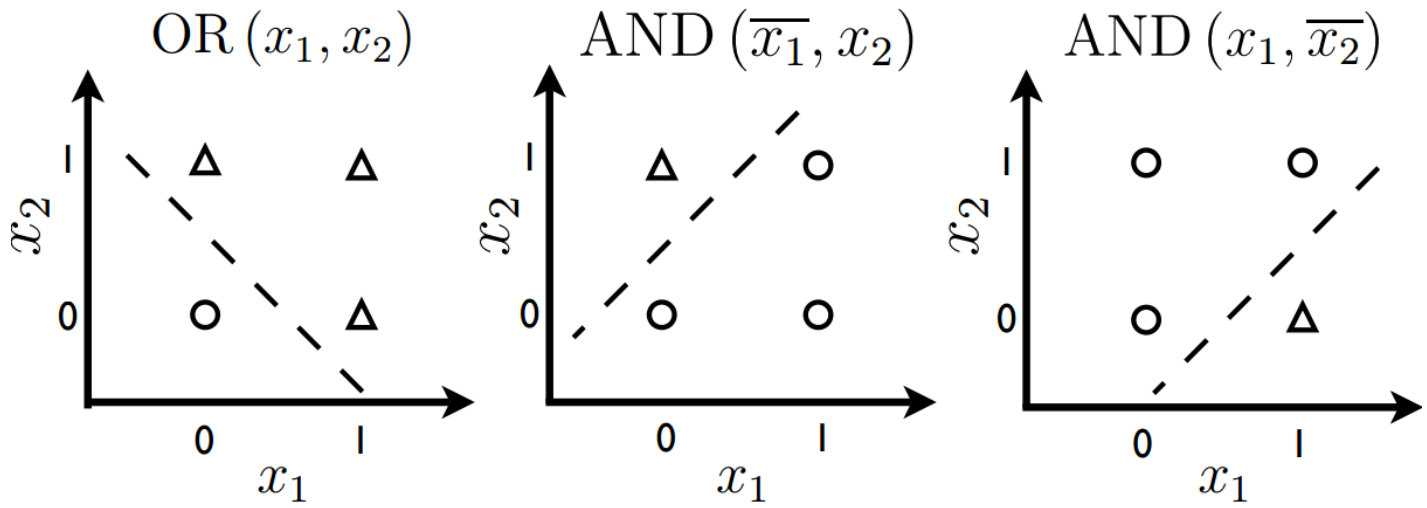
- Let the correct label be $y = +1$ and define $s = z_2 - z_1$.
- Sparsemax loss in 2D becomes a "classification Huber loss":



Outline

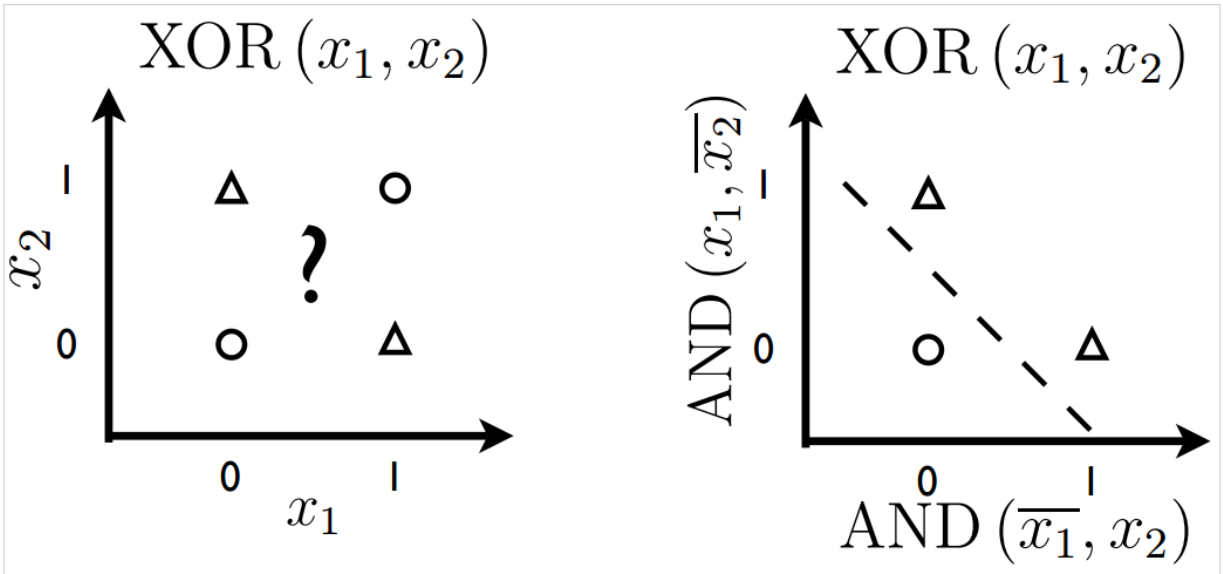
Recap: What a Linear Classifier Can Do

- 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:

- 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:

- 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 (next class!)

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

Two Views of Machine Learning

There's two big ways of building machine learning systems:

1. Feature-based: describe objects' properties (features) and build models that manipulate them
   - everything that we have seen so far.
2. Similarity-based: don't describe objects by their properties; rather, build systems based on **comparing** objects to each other
   - k-th nearest neighbors; kernel methods; Gaussian processes.

Sometimes the two are equivalent!

Nearest Neighbor Classifier

- Not a linear classifier!
- In its simplest version, doesn't require any parameters
- Instead of "training", **memorize** all the data $\mathcal{D} = \{(x_i, y_i)_{i=1}^N\}$
- Given a new input x, find its **most similar** data point x_i and predict

$$\widehat{y} = y_i$$

- Many variants (e.g. k-th nearest neighbor)
- **Disadvantage:** requires searching over the entire training data
- Specialized data structures can be used to speed up search.

Kernels

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

$$\kappa(x_i, x_j) \in \mathbb{R}$$

- Given dataset $\mathcal{D} = \{(x_i, y_i)_{i=1}^N\}$, the Gram matrix $\boldsymbol{K}$ is the $N \times N$ matrix defined as:

$$K_{i,j} = \kappa(x_i, x_j)$$

- **Symmetric:**

$$\kappa(x_i, x_j) = \kappa(x_j, x_i)$$

- **Positive definite:** for all non-zero $\mathbf{v}$

$$\mathbf{v}\boldsymbol{K}\mathbf{v}^T \geq 0$$

Kernels

- **Mercer's Theorem**: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}}$, s.t.:

$$\kappa(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

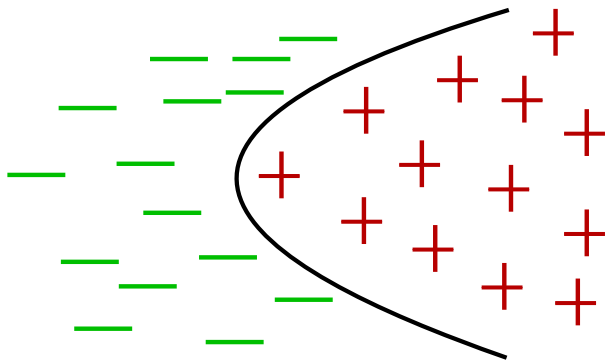
- That is: a kernel corresponds to some a mapping in some **implicit** feature space!
- Kernel trick: take a feature-based algorithm (SVMs, perceptron, logistic regression) and replace all explicit feature computations by **kernel evaluations**!

$$\boldsymbol{w}_y \cdot \phi(x) = \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} \alpha_{i,y} \kappa(x, x_i) \quad \text{for some } \alpha_{i,y} \in \mathbb{R}$$

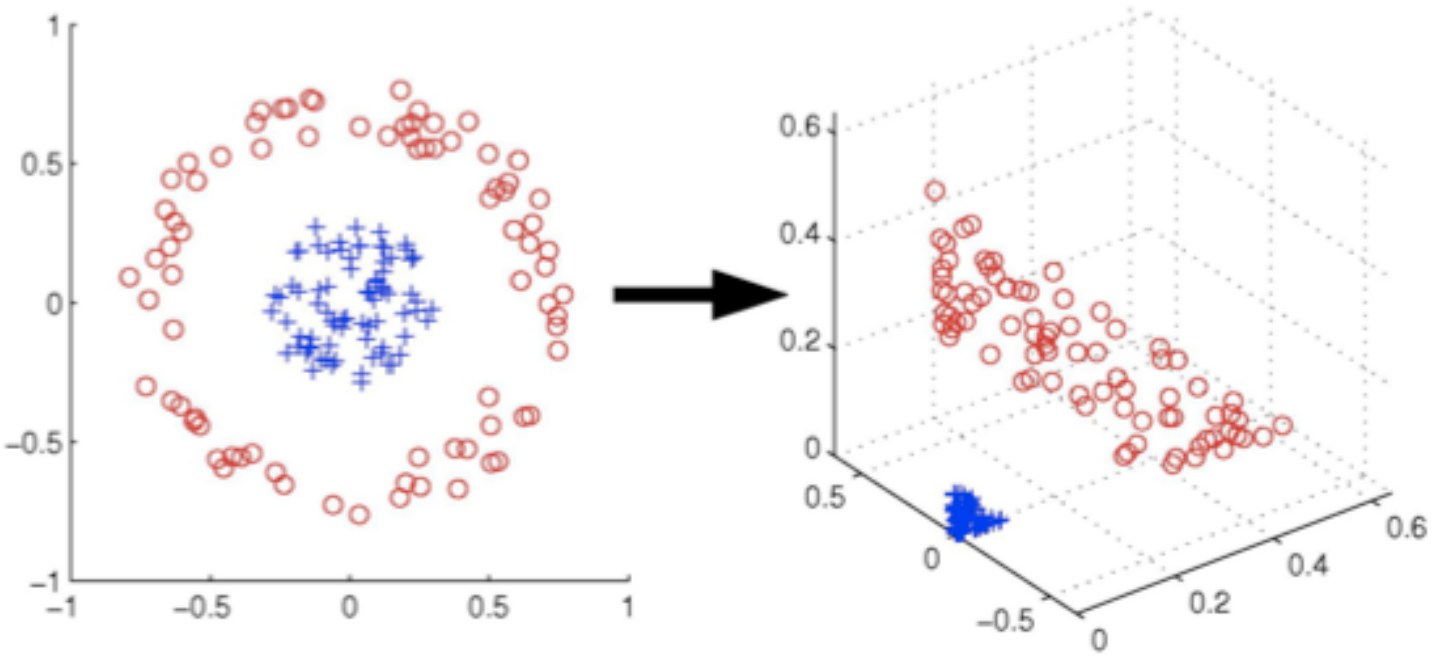
- Extremely popular idea in the 1990-2000s!

Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space
- Computing a non-linear kernel is sometimes better computationally than calculating the corresponding dot product in the high dimension feature space
- Many models can be "kernelized" – learning algorithms generally solve the dual optimization problem (also convex)
- Drawback: quadratic dependency on dataset size



Linear Classifiers in High Dimension



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

Popular Kernels

- Polynomial kernel

$$\kappa(x_i, x_j) = (\phi(x_i) \cdot \phi(x_j) + 1)^d$$

- Gaussian radial basis kernel

$$\kappa(x_i, x_j) = exp(\frac{-||\phi(x_i) - \phi(x_j)||^2}{2\sigma})$$

- String kernels **(Lodhi et al., 2002; Collins and Duffy, 2002)**
- Tree kernels **(Collins and Duffy, 2002)**

Joint Feature Mappings (useful for the labs)

Feature Representations: Joint Feature Mappings

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

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

Each feature now represents a joint property of the input x and the candidate output y.

We'll use this notation in the labs this afternoon!

Examples

- x is a document and y is a label

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

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

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

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

More Examples

- x is a name, y is a label classifying the type of entity

$$\psi_0(x,y) = \begin{cases} 1 & \text{if } x \text{ contains "George"} \\ & \text{and } y = \text{"Person"} \\ 0 & \text{otherwise} \end{cases}$$

$$\psi_1(x,y) = \begin{cases} 1 & \text{if } x \text{ contains "Washington"} \\ & \text{and } y = \text{"Person"} \\ 0 & \text{otherwise} \end{cases}$$

$$\psi_2(x,y) = \begin{cases} 1 & \text{if } x \text{ contains "Bridge"} \\ & \text{and } y = \text{"Person"} \\ 0 & \text{otherwise} \end{cases}$$

$$\psi_3(x,y) = \begin{cases} 1 & \text{if } x \text{ contains "General"} \\ & \text{and } y = \text{"Person"} \\ 0 & \text{otherwise} \end{cases}$$

$$\psi_4(x,y) = \begin{cases} 1 & \text{if } x \text{ contains "George"} \\ & \text{and } y = \text{"Location"} \\ 0 & \text{otherwise} \end{cases}$$

$$\psi_5(x,y) = \begin{cases} 1 & \text{if } x \text{ contains "Washington"} \\ & \text{and } y = \text{"Location"} \\ 0 & \text{otherwise} \end{cases}$$

$$\psi_6(x,y) = \begin{cases} 1 & \text{if } x \text{ contains "Bridge"} \\ & \text{and } y = \text{"Location"} \\ 0 & \text{otherwise} \end{cases}$$

$$\psi_7(x,y) = \begin{cases} 1 & \text{if } x \text{ contains "General"} \\ & \text{and } y = \text{"Location"} \\ 0 & \text{otherwise} \end{cases}$$

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

Block Feature Vectors

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

- Each equal size block of the feature vector corresponds to one label
- Non-zero values allowed only in one block

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

Equivalent if $\psi(x, y)$ conjoins input features $\phi(x)$ with one-hot label representations $\boldsymbol{e}_y := [0, \ldots, 0, 1, 0, \ldots, 0]$

$$\begin{aligned} \psi(x, y) &= \phi(x) \otimes \boldsymbol{e}_y \\ &= [\mathbf{0}, \ldots, \mathbf{0}, \underbrace{\phi(x)}_{y^{\text{th}} \text{ block}}, \mathbf{0}, \ldots, \mathbf{0}] \end{aligned}$$

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

$\phi(x)$ is sometimes simpler and more convenient in binary classification

... but $\psi(x, y)$ is more expressive (allows more complex features over properties of labels)

Linear Classifiers – $\psi(x, y)$

- 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 $\widehat{y}$ which maximizes this score:

$$\widehat{y} = h(x) = \arg\max_{y \in \mathcal{Y}} \boldsymbol{w} \cdot \boldsymbol{\psi}(x, y)$$

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

Linear Classifiers – $\phi(x)$

- Define $|\mathcal{Y}|$ weight vectors $\boldsymbol{w}_y \in \mathbb{R}^D$
  - i.e., one weight vector per output label y

- **Classification**

$$\widehat{y} = \arg\max_{y \in \mathcal{Y}} \; \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x)$$

Linear Classifiers – $\phi(x)$

- Define $|\mathcal{Y}|$ weight vectors $\boldsymbol{w}_y \in \mathbb{R}^D$
  - i.e., one weight vector per output label y

- **Classification**

$$\widehat{y} = \arg\max_{y \in \mathcal{Y}} \ \boldsymbol{w}_y \cdot \boldsymbol{\phi}(x)$$

- $\boldsymbol{\psi}(x, y)$
  - x=General George Washington, y=Person $\rightarrow \boldsymbol{\psi}(x, y) = [1\ 1\ 0\ 1\ 0\ 0\ 0\ 0]$
  - x=General George Washington, y=Object $\rightarrow \boldsymbol{\psi}(x, y) = [0\ 0\ 0\ 0\ 1\ 1\ 0\ 1]$
  - Single $\boldsymbol{w} \in \mathbb{R}^8$

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

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
- **Tomorrow:** methods that are able to learn internal representations

Thank You!

Questions?

References I

Collins, M. and Duffy, N. (2002). Convolution kernels for natural language. *Advances in Neural Information Processing Systems*, 1:625–632.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

Martins, A. F. T. and Astudillo, R. (2016). From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *Proc. of the International Conference on Machine Learning*.

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.