Lecture 2: Linear Classifiers

André Martins

Deep Structured Learning Course, Fall 2018

Course Information

- **Instructor:** André Martins (`amartins@lx.it.pt`)
- **TAs/Guest Lecturers:** Erick Fonseca & Vlad Niculae
- **Location:** LT2 (North Tower, 4th floor)
- **Schedule:** Wednesdays 14:30–18:00
- **Communication:**
  piazza.com/tecnico.ulisboa.pt/fall2018/pdeecdsl



Announcements

Homework 1 is out!

- Deadline: October 10 (two weeks from now)
- Start early!!!

List of potential projects will be sent out soon!

- Deadline for project proposal: October 17 (three weeks from now)
- Teams of 3 people

Today's Roadmap

Before talking about deep learning, let us talk about shallow learning:

- Supervised learning: binary and multi-class classification
- Feature-based linear classifiers
- Rosenblatt's perceptron algorithm
- Linear separability and separation margin: perceptron's mistake bound
- Other linear classifiers: naive Bayes, logistic regression, SVMs
- Regularization and optimization
- Limitations of linear classifiers: the XOR problem
- Kernel trick. Gaussian and polynomial kernels.

Fake News Detection

Task: tell if a news article / quote is fake or real.

This is a **binary classification problem.**

Fake Or Real?


NATIONAL
ENQUIRER
THE WORLD'S LIVELIEST PAPER
FEATURE 15¢
Rita Hayworth Says...
I'M BACK FROM THE DEAD
For Two Years I Was a Zombie
WEEKLY WORLD
NEWS
Space creature survived UFO crash in Arkansas!
HILLARY CLINTON ADOPTS ALIEN BABY
OFFICIAL PHOTO!
Secret Service building special nursery in the White House!

Fake Or Real?


With Artificial
Intelligence we
are summoning
the demons
- Elon Musk

Fake Or Real?

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

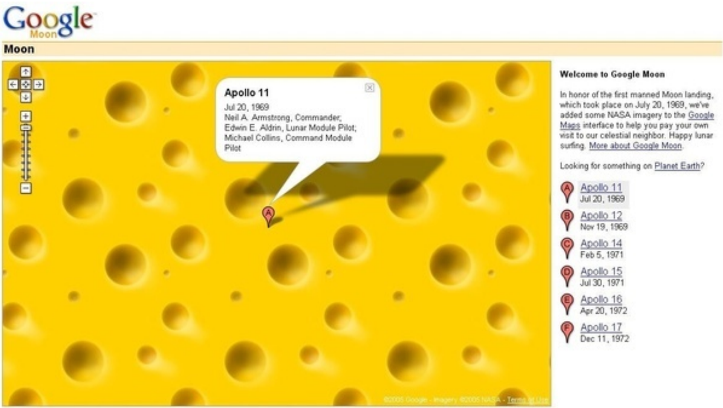
21 March 2016, 10:16 am EDT **By Aaron Mamiit Tech Times**

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.

Fake Or Real?


Google Moon
Moon
Apollo 11
Jul 20, 1969
Neil A. Armstrong, Commander;
Edwin E. Aldrin, Lunar Module Pilot;
Michael Collins, Command Module Pilot
Welcome to Google Moon
In honor of the first manned Moon landing, which took place on July 20, 1969, we've added some NASA imagery to the Google Maps interface to help you pay your own visit to our celestial neighbor. Happy lunar surfing. More about Google Moon.
Looking for something on Planet Earth?
Apollo 11
Jul 20, 1969
Apollo 12
Nov 19, 1969
Apollo 14
Feb 5, 1971
Apollo 15
Jul 30, 1971
Apollo 16
Apr 20, 1972
Apollo 17
Dec 11, 1972

Fake Or Real?

Can a machine determine this automatically?

Can be a very hard problem, since fact-checking is hard and requires combining several knowledge sources

... also, reality surpasses fiction sometimes

Shared task: `http://www.fakenewschallenge.org/`

Topic Classification

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

This is a **multi-class classification problem.**

It's a much easier task, can get 80-90% accuracies with a simple ML model

Topic Classification

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

1. **Preliminaries**

    Data and Feature Representation

2. **Linear Classifiers**

    Perceptron

    Naive Bayes

    Logistic Regression

    Support Vector Machines

    Regularization

3. **Non-Linear Classifiers**

Outline

1. **Preliminaries**

   Data and Feature Representation

2. **Linear Classifiers**

   Perceptron

   Naive Bayes

   Logistic Regression

   Support Vector Machines

   Regularization

3. **Non-Linear Classifiers**

Disclaimer

Many of the following slides are adapted from Ryan McDonald.

Let's Start Simple

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

Let's Start Simple

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

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

Let's Start Simple

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

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

Why can we do this?

Let's Start Simple: Machine Learning

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

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

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

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

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

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

Let's Start Simple: Machine Learning

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

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

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

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

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

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

Machine Learning

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

Machine Learning

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

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

- Assumption (naive Bayes—more later):

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

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

Some Notation: Inputs and Outputs

- Input $\boldsymbol{x} \in \mathcal{X}$
    - e.g., a news article, a sentence, an image, ...
- Output $\boldsymbol{y} \in \mathcal{Y}$
    - e.g., fake/not fake, a topic, a parse tree, an image segmentation
- Input/Output pair: $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$
    - e.g., a **news article** together with a **topic**
    - e.g., a **sentence** together with a **parse tree**
    - e.g., an **image** partitioned into **segmentation regions**

Supervised Machine Learning

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

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

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

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

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

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

Regression

Deals with **continuous** output variables:

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

Classification

Deals with **discrete** output variables:

- **Binary classification:** $\mathcal{Y} = \{\pm 1\}$
  - e.g., fake news detection
- **Multi-class classification:** $\mathcal{Y} = \{1, 2, \ldots, K\}$
  - e.g., topic classification
- **Structured classification:** $\mathcal{Y}$ exponentially large and structured
  - e.g., machine translation, caption generation, image segmentation

This course: **structured classification**

... but to make it simpler, we'll talk about multi-class classification first.

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 "shallow" learning:

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

Feature Representations

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

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

- $\boldsymbol{\psi}(\boldsymbol{x})$ is a high dimensional feature vector

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

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

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.

Examples

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

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

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

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

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

More Examples

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

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

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

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

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

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

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

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

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

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

Block Feature Vectors

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

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

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

Equivalent if $\phi(\boldsymbol{x}, \boldsymbol{y})$ conjoins input features with one-hot label representations

- $\phi(\boldsymbol{x}, \boldsymbol{y}) = \psi(\boldsymbol{x}) \otimes \boldsymbol{e}_{\boldsymbol{y}}$, where $\boldsymbol{e}_{\boldsymbol{y}} := (0, \ldots, 0, 1, 0, \ldots, 0)$
- $\phi(\boldsymbol{x}, \boldsymbol{y})$
  - $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Person $\rightarrow \phi(\boldsymbol{x}, \boldsymbol{y}) = [1\ 1\ 0\ 1\ 0\ 0\ 0\ 0]$
  - $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Object $\rightarrow \phi(\boldsymbol{x}, \boldsymbol{y}) = [0\ 0\ 0\ 0\ 1\ 1\ 0\ 1]$
- $\psi(\boldsymbol{x})$
  - $\boldsymbol{x}$=General George Washington $\rightarrow \psi(\boldsymbol{x}) = [1\ 1\ 0\ 1]$

$\psi(\boldsymbol{x})$ is sometimes simpler and more convenient in binary classification

... but $\phi(\boldsymbol{x}, \boldsymbol{y})$ is more expressive (allows more complex features over properties of labels)

Preprocessing and Feature Engineering

- All sorts of linguistic processing for "meta-counts"
  - POS tagging: adjective counts for sentiment analysis
  - Spell checker: misspellings counts for spam detection
  - Parsing: depth of tree for readability assessment
  - Co-occurrences: language models probabilities, 2nd-order context and word-embeddings for text classification
- Structured inputs for other representations (e.g. string or tree kernels)
- Common dimensionality reduction strategies also used (e.g. PCA)

Example: Translation Quality Estimation

- 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**

We'll discuss this later in the class.

Outline

Linear Classifiers

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

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

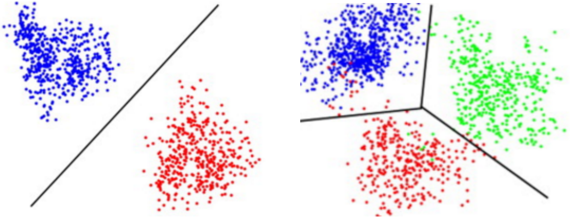
- At training time, different strategies to learn $\boldsymbol{w}$ yield different linear classifiers: perceptron, naïve Bayes, logistic regression, SVMs, ...

Linear Classifiers

- Prediction rule:

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

- 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 Classifiers – $\psi(x)$

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

- **Classification**

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

Linear Classifiers – $\psi(x)$

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

- **Classification**

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

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

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

Linear Classifiers – Bias Terms

- Often linear classifiers are presented as

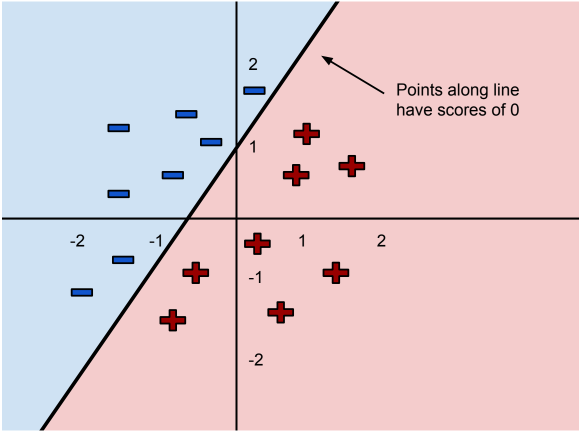
$$\widehat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) + b_{\boldsymbol{y}}$$

  where $b_{\boldsymbol{y}}$ is a bias or offset term
- This can be folded into ϕ (by defining a constant feature for each label)
- We assume this for simplicity

Binary Linear Classifier

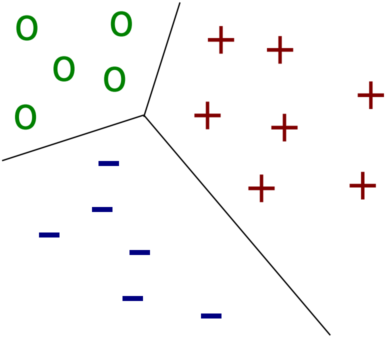
Let's say $\boldsymbol{w} = (1, -1)$ and $b_{\boldsymbol{y}} = 1$, $\forall \boldsymbol{y}$

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



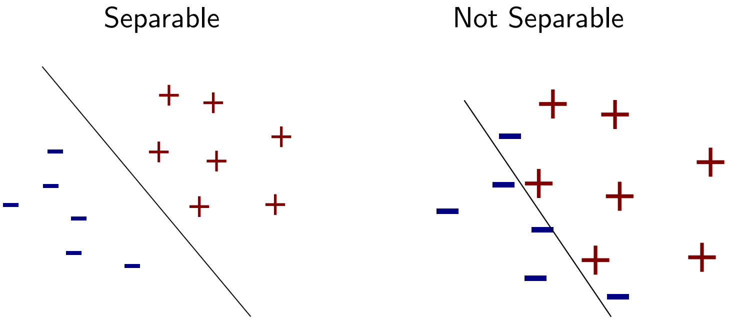
Multiclass Linear Classifier

Defines regions of space.



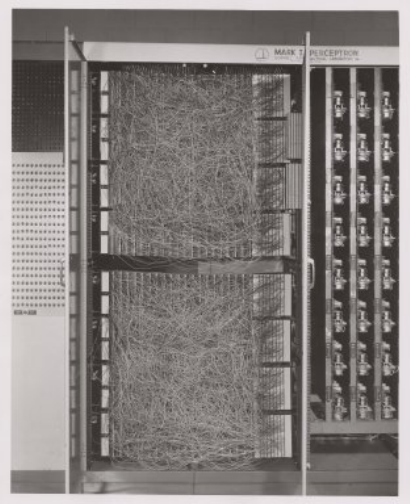
Linear Separability

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



Outline

Perceptron (Rosenblatt, 1958)



(Extracted from Wikipedia)

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

Perceptron in the News...

1958 New York Times...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

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

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

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

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

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

Without Human Controls

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

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

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

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

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

Learns by Doing

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

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

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

Perceptron in the News...

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

Perceptron Algorithm

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

Perceptron's Mistake Bound

A couple definitions:

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

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

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

Perceptron's Mistake Bound

A couple definitions:

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

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

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

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

Theorem (Novikoff (1962))

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

One-Slide Proof

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

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

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

One-Slide Proof

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

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

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

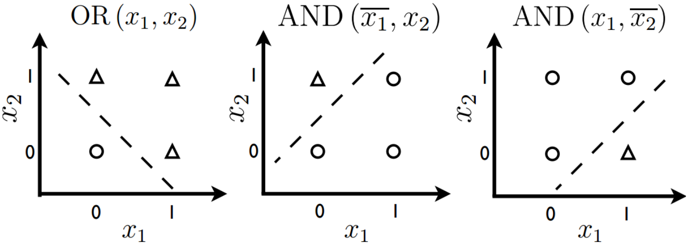
- **Upper bound on $\|\boldsymbol{w}^{(k+1)}\|$:**

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

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

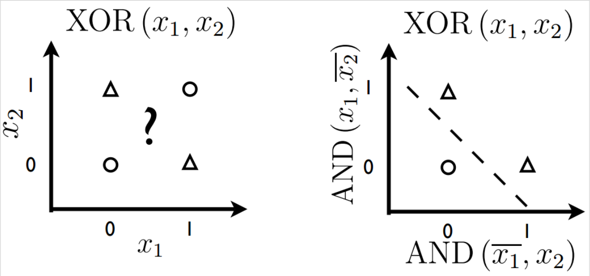
What a Simple Perceptron Can and Can't Do

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



What a Simple Perceptron Can and Can't Do

- … 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) and motivated strong criticisms

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 $\boldsymbol{y}$ given inputs $\boldsymbol{x}$ (or $\phi(\boldsymbol{x})$), i.e. $P(\boldsymbol{y}|\boldsymbol{x})$
- If we can define this distribution, then classification becomes:

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

Bayes Rule

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

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

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

- Since $\boldsymbol{x}$ is fixed
- $P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y}) = P(\boldsymbol{x}, \boldsymbol{y})$: a joint probability
- Modeling the joint input-output distribution is at the core of generative models
  - Because we model a distribution that can randomly generate outputs *and* inputs, not just outputs

Naive Bayes

- Use $\phi(\boldsymbol{x}) \in \mathbb{R}^D$ instead of $\phi(\boldsymbol{x}, \boldsymbol{y})$
- $P(\boldsymbol{x}|\boldsymbol{y}) = P(\phi(\boldsymbol{x})|\boldsymbol{y}) = P(\phi_1(\boldsymbol{x}), \ldots, \phi_D(\boldsymbol{x})|\boldsymbol{y})$

Naive Bayes Assumption

(conditional independence)

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

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

Naive Bayes – Learning

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

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

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

Naive Bayes – Learning via MLE

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

It all boils down to counting and normalizing!!

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

Naive Bayes – Learning via MLE

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

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

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

$[[X]]$ is the identity function for property X
Fraction of times a feature appears among all features in training cases of a given label

Naive Bayes Example

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

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

Naive Bayes Example

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

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

Naive Bayes Example

Relative frequency:

Priors:

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

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

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

Naive Bayes Example

Likelihoods: Count adjective $\phi_i(\boldsymbol{x})$ in class $\boldsymbol{y}$ / adjectives in $\boldsymbol{y}$

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

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

Naive Bayes Example

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

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

Final decision

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

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

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

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

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

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: $sentiment = ???$

Naive Bayes Example

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

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

where $|V|$ = no 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: $sentiment = negative$

Discriminative versus Generative

- Generative models attempt to model inputs and outputs
  - e.g., NB = MLE of joint distribution $P(\boldsymbol{x}, \boldsymbol{y})$
  - Statistical model must explain generation of input
- Occam's Razor: why model input?
- Discriminative models
  - Use loss function that directly optimizes $P(\boldsymbol{y}|\boldsymbol{x})$ (or something related)
  - Logistic Regression – MLE of $P(\boldsymbol{y}|\boldsymbol{x})$
  - Perceptron and SVMs – minimize classification error
- Generative and discriminative models use $P(\boldsymbol{y}|\boldsymbol{x})$ for prediction
  - They differ only on what distribution they use to set $\boldsymbol{w}$

Outline

1. Preliminaries
   - Data and Feature Representation
2. Linear Classifiers
   - Perceptron
   - Naive Bayes
   - Logistic Regression
   - Support Vector Machines
   - Regularization
3. Non-Linear Classifiers

Logistic Regression

Define a conditional probability:

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

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

Note: still a linear classifier

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

Logistic Regression

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

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

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

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

Logistic Regression

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

- 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!

Pro: Guarantee of a global minima ✓

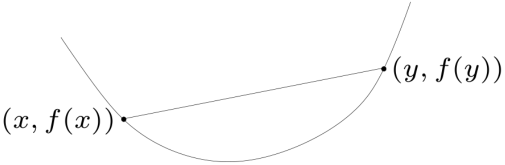


Figure: Illustration of a convex function. The line segment between any two points on the graph lies entirely above the curve.

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: $x^{(t+1)} \leftarrow x^{(t)} - \eta_{(t)} \nabla f(x^{(t)})$

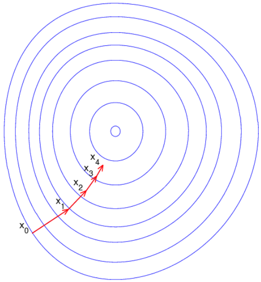


Figure: Illustration of gradient descent. The red lines correspond to steps taken in the negative gradient direction.

Gradient Descent

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

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

- $\nabla_{\boldsymbol{w}} L(\boldsymbol{w})$ is gradient of L w.r.t. $\boldsymbol{w}$

- Gradient descent will always find the optimal $\boldsymbol{w}$

Stochastic Gradient Descent

If the dataset is large, we'd better do SGD instead, for more frequent updates:

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

Computing the Gradient

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

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

Some reminders:

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

Computing the Gradient

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

The gradient equals the "difference between the expected features under the current model and the true features."

Logistic Regression Summary

- Define conditional probability

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

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

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

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

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) = \sum_{\boldsymbol{y}'} P_{\boldsymbol{w}}(\boldsymbol{y}'|\boldsymbol{x}) \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$$

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

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

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

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

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

Maximizing Margin

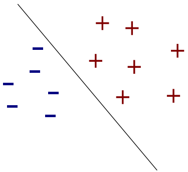
- For a training set $\mathcal{D}$
- Margin of a weight vector $\boldsymbol{w}$ is smallest γ such that

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

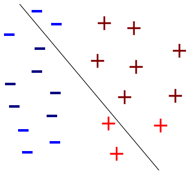
- for every training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$, $\boldsymbol{y}' \in \mathcal{Y}$

Margin

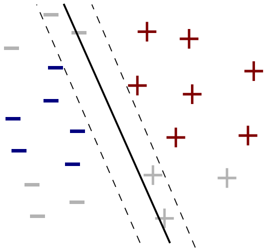
Training

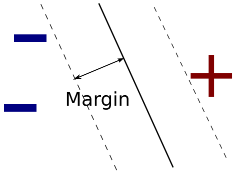


Testing



Denote the value of the margin by γ





Maximizing Margin

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

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

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

Outline

Maximizing Margin

Let $\gamma > 0$

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

such that:

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

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

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

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

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

$$=$$

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

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

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

Min Norm:

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

such that:

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

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

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

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

Max Margin = Min Norm

Max Margin:

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

such that:

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

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

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

=

Min Norm:

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

such that:

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

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

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

- Let's say min norm solution $||\boldsymbol{w}|| = \zeta$
- Now say original objective is $\max_{||\boldsymbol{w}||\leq\zeta}\gamma$
- We know that γ must be 1
  - Or we would have found smaller $||\boldsymbol{w}||$ in min norm solution
- $||\boldsymbol{w}|| \leq 1$ in max margin formulation is an arbitrary scaling choice

Support Vector Machines

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

such that:

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

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

- 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:

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

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

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

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:

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

Support Vector Machines

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

such that:

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

Support Vector Machines

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

such that:

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

Support Vector Machines

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

such that:

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

Support Vector Machines

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

such that:

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

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

Support Vector Machines

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

such that:

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

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

Hinge loss:

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

Support Vector Machines

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

such that:

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

Hinge loss equivalent

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

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

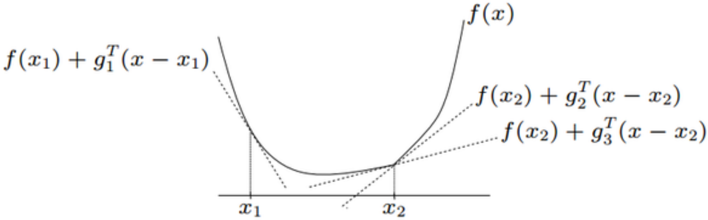
From Gradient to Subgradient

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

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

A subgradient of the hinge is

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

where

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

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

Perceptron and Hinge-Loss

SVM subgradient update looks like perceptron update

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

Perceptron

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

where $\eta = 1$

Perceptron = SGD with no-margin hinge-loss

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

Summary

What we have covered

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

What is next

- Regularization
- Non-linear classifiers

Outline

Regularization

Overfitting

- Early in lecture we made assumption data was i.i.d.
- Rarely is this true
  - E.g., syntactic analyzers typically trained on 40,000 sentences from early 1990s WSJ news text
- Even more common: $\mathcal{D}$ is very small
- This leads to overfitting

Regularization

- We saw one example already when talking about add-one smoothing in Naive Bayes!
- 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
- λ controls how much to regularize

- Common functions
  - ℓ_2: $\Omega(\boldsymbol{w}) \propto \|\boldsymbol{w}\|_2 = \|\boldsymbol{w}\| = \sqrt{\sum_i \boldsymbol{w}_i^2}$ – smaller weights desired
  - ℓ_0: $\Omega(\boldsymbol{w}) \propto \|\boldsymbol{w}\|_0 = \sum_i [[\boldsymbol{w}_i > 0]]$ – zero weights desired
    - Non-convex
    - Approximate with ℓ_1: $\Omega(\boldsymbol{w}) \propto \|\boldsymbol{w}\|_1 = \sum_i |\boldsymbol{w}_i|$

Logistic Regression with ℓ_2 Regularization

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

- What is the new gradient?

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

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

Support Vector Machines

Hinge-loss formulation: ℓ_2 regularization already happening!

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

$\uparrow$ SVM optimization $\uparrow$

SVMs vs. Logistic Regression

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

SVMs vs. Logistic Regression

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

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

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

SVMs vs. Logistic Regression

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

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

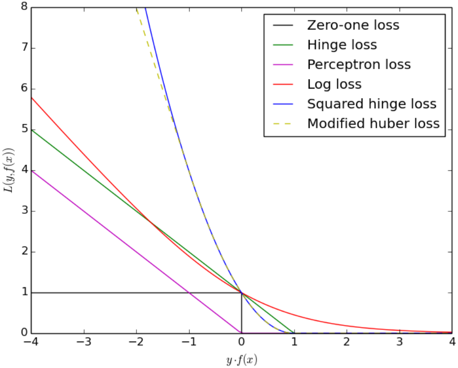
$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \sum_{t=1}^{N} \max\left(0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

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

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

Generalized Linear Classifiers

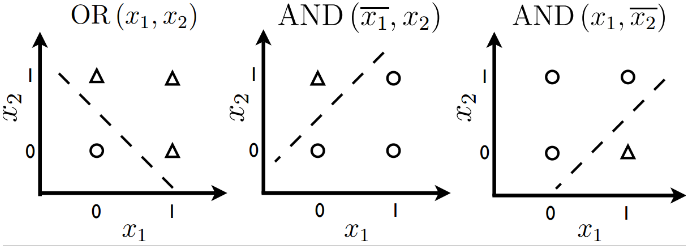
$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \sum_{t=1}^{N} L((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{w}) + \lambda\Omega(\boldsymbol{w})$$



Outline

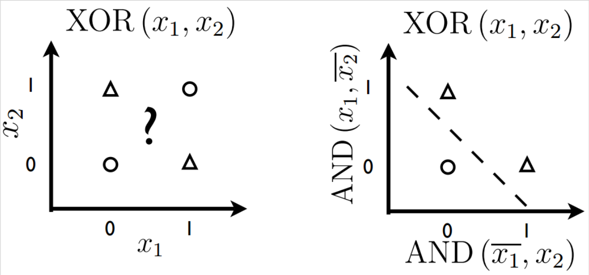
Recap: What a Linear Classifier Can Do

- 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

Kernels

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

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

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

$$K_{t,r} = \kappa(\boldsymbol{x}_t, \boldsymbol{x}_r)$$

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

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

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

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

Kernels

- **Mercer's Theorem**: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{Y}$, there exists a $\boldsymbol{\psi} : \mathcal{X} \rightarrow \mathbb{R}^{\mathcal{X}}$, s.t.:

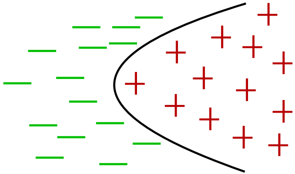
$$\kappa(\boldsymbol{x}_t, \boldsymbol{x}_r) = \boldsymbol{\psi}(\boldsymbol{x}_t) \cdot \boldsymbol{\psi}(\boldsymbol{x}_r)$$

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

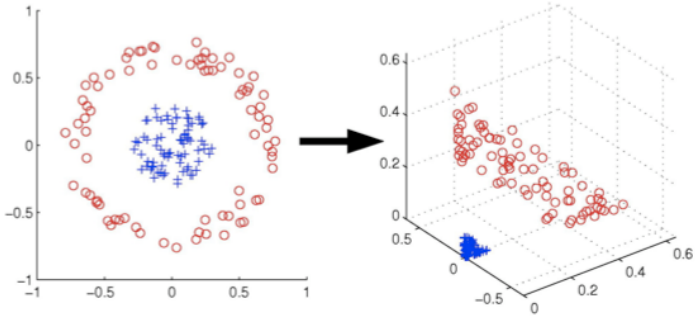
$$\kappa((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}_r, \boldsymbol{y}_r)) = \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) \cdot \boldsymbol{\phi}(\boldsymbol{x}_r, \boldsymbol{y}_r)$$

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



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

Example: Polynomial Kernel

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

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

which equals:

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

Popular Kernels

- Polynomial kernel

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

- Gaussian radial basis kernel

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

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

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
- **Later:** 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.

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.