Learning with Linear Models

Mário Figueiredo and André Martins

instituto de telecomunicações

TÉCNICO LISBOA

Lisbon Machine Learning School, July 14, 2023

Why Linear Models?

- In 2023, deep neural networks are ubiquitous!

Why Linear Models?

- In 2023, deep neural networks are ubiquitous!

- Why a lecture on linear models?

    ✓ The underlying machine learning concepts are the same

Why Linear Models?

- In 2023, deep neural networks are ubiquitous!
- Why a lecture on linear models?
  - ✓ The underlying machine learning concepts are the same
  - ✓ The theory (statistics and optimization) are much better understood

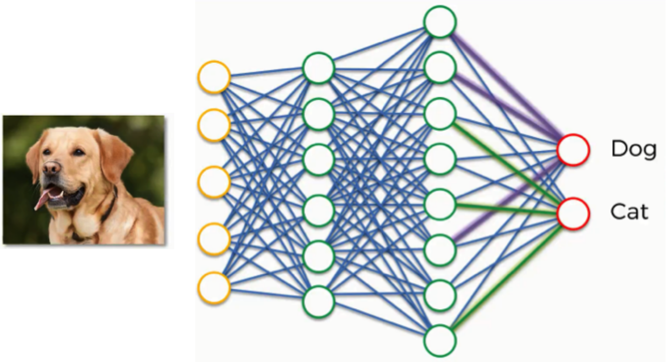
Why Linear Models?

- In 2023, deep neural networks are ubiquitous!
- Why a lecture on linear models?
  - ✓ The underlying machine learning concepts are the same
  - ✓ The theory (statistics and optimization) are much better understood
  - ✓ Linear models are still widely used (very effective if data is scarce)

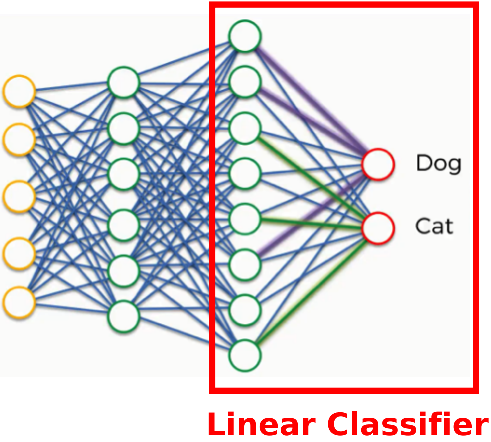
Why Linear Models?

- In 2023, deep neural networks are ubiquitous!
- Why a lecture on linear models?
  - ✓ The underlying machine learning concepts are the same
  - ✓ The theory (statistics and optimization) are much better understood
  - ✓ Linear models are still widely used (very effective if data is scarce)
  - ✓ Linear models are **a component of deep networks**.

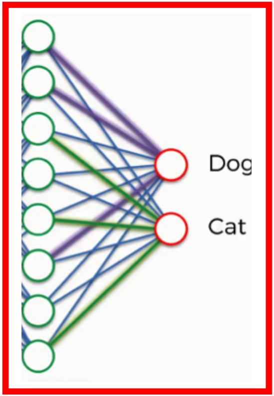
Linear Classifiers and Neural Networks



Linear Classifiers and Neural Networks



Linear Classifiers and Neural Networks



Linear Classifier

Today's Roadmap

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

Some Notation: Inputs and Outputs

- Input $x \in \mathcal{X}$
  - ✓ e.g., a news article, a sentence, an image, ...

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, an image segmentation

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, an image segmentation
- Input/output pair: $(x, y) \in \mathcal{X} \times \mathcal{Y}$

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, 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 its **translation**
  - ✓ e.g., an **image** partitioned into **segmentation regions**

Supervised Machine Learning

- Given a collection of input/output pairs (training data)

$$\mathcal{D} = (x_1, y_1), ..., (x_N, y_N) \in \mathcal{X} \times \mathcal{Y}$$

Supervised Machine Learning

- Given a collection of input/output pairs (training data)

$$\mathcal{D} = (x_1, y_1), ..., (x_N, y_N) \in \mathcal{X} \times \mathcal{Y}$$

- ... **learn** a **predictor** $h : \mathcal{X} \to \mathcal{Y}$.

Supervised Machine Learning

- Given a collection of input/output pairs (training data)

$$\mathcal{D} = (x_1, y_1), ..., (x_N, y_N) \in \mathcal{X} \times \mathcal{Y}$$

- ... **learn** a **predictor** $h : \mathcal{X} \to \mathcal{Y}$.

- To use it for a new input $x \in \mathcal{X}$, **predict/infer** $\widehat{y} = h(x)$.

Supervised Machine Learning

- Given a collection of input/output pairs (training data)

$$\mathcal{D} = (x_1, y_1), ..., (x_N, y_N) \in \mathcal{X} \times \mathcal{Y}$$

- ... **learn** a **predictor** $h : \mathcal{X} \to \mathcal{Y}$.
- To use it for a new input $x \in \mathcal{X}$, **predict/infer** $\widehat{y} = h(x)$.
- Hopefully, $\widehat{y} \approx y$ most of the time, i.e., h should **generalize**.

Supervised Machine Learning

- Given a collection of input/output pairs (training data)

$$\mathcal{D} = (x_1, y_1), \ldots, (x_N, y_N) \in \mathcal{X} \times \mathcal{Y}$$

- ... **learn** a **predictor** $h : \mathcal{X} \to \mathcal{Y}$.
- To use it for a new input $x \in \mathcal{X}$, **predict/infer** $\widehat{y} = h(x)$.
- Hopefully, $\widehat{y} \approx y$ most of the time, i.e., h should **generalize**.
- Standard approach: **empirical risk minimization** (ERM):

$$h = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{N} L\big(h(x_i), y_i\big)$$

where L is a loss function and $\mathcal{H}$ a model class.

Regression vs Classification

Regression: continuous/quantitative $\mathcal{Y}$;

Classification: discrete/categorical $\mathcal{Y}$.

Regression vs Classification

Regression: continuous/quantitative $\mathcal{Y}$;

Classification: discrete/categorical $\mathcal{Y}$.

- **Regression:** $\mathcal{Y} = \mathbb{R}$, or $\mathcal{Y} = [0, 1]$, or $\mathcal{Y} = \mathbb{R}_+$, or ...
  - ✓ e.g., given a news article, how much time a user will spend reading it?

Regression vs Classification

Regression: continuous/quantitative $\mathcal{Y}$;

Classification: discrete/categorical $\mathcal{Y}$.

- **Regression:** $\mathcal{Y} = \mathbb{R}$, or $\mathcal{Y} = [0, 1]$, or $\mathcal{Y} = \mathbb{R}_+$, or ...
  - ✓ e.g., given a news article, how much time a user will spend reading it?
- **Multivariate regression:** $\mathcal{Y} = \mathbb{R}^K$, or $\mathcal{Y} = \mathbb{R}_+^K$, or $\mathcal{Y} = \Delta_K$, or ...
  - ✓ e.g., denoise an image, estimate class probabilities, ...

Regression vs Classification

Regression: continuous/quantitative $\mathcal{Y}$;

Classification: discrete/categorical $\mathcal{Y}$.

- **Regression:** $\mathcal{Y} = \mathbb{R}$, or $\mathcal{Y} = [0, 1]$, or $\mathcal{Y} = \mathbb{R}_+$, or ...
  - ✓ e.g., given a news article, how much time a user will spend reading it?
- **Multivariate regression:** $\mathcal{Y} = \mathbb{R}^K$, or $\mathcal{Y} = \mathbb{R}_+^K$, or $\mathcal{Y} = \Delta_K$, or ...
  - ✓ e.g., denoise an image, estimate class probabilities, ...
- **Binary classification:** $\mathcal{Y} = \{\pm 1\}$
  - ✓ e.g., spam detection, fraud detection, ...

Regression vs Classification

Regression: continuous/quantitative $\mathcal{Y}$;

Classification: discrete/categorical $\mathcal{Y}$.

- **Regression:** $\mathcal{Y} = \mathbb{R}$, or $\mathcal{Y} = [0, 1]$, or $\mathcal{Y} = \mathbb{R}_+$, or ...
  - ✓ e.g., given a news article, how much time a user will spend reading it?
- **Multivariate regression:** $\mathcal{Y} = \mathbb{R}^K$, or $\mathcal{Y} = \mathbb{R}_+^K$, or $\mathcal{Y} = \Delta_K$, or ...
  - ✓ e.g., denoise an image, estimate class probabilities, ...
- **Binary classification:** $\mathcal{Y} = \{\pm 1\}$
  - ✓ e.g., spam detection, fraud detection, ...
- **Multi-class classification:** $\mathcal{Y} = \{1, 2, \ldots, K\}$ (order is irrelevant)
  - ✓ e.g., topic classification, image classification, ...

Regression vs Classification

Regression: continuous/quantitative $\mathcal{Y}$;

Classification: discrete/categorical $\mathcal{Y}$.

- **Regression:** $\mathcal{Y} = \mathbb{R}$, or $\mathcal{Y} = [0, 1]$, or $\mathcal{Y} = \mathbb{R}_+$, or ...
  - ✓ e.g., given a news article, how much time a user will spend reading it?
- **Multivariate regression:** $\mathcal{Y} = \mathbb{R}^K$, or $\mathcal{Y} = \mathbb{R}_+^K$, or $\mathcal{Y} = \Delta_K$, or ...
  - ✓ e.g., denoise an image, estimate class probabilities, ...
- **Binary classification:** $\mathcal{Y} = \{\pm 1\}$
  - ✓ e.g., spam detection, fraud detection, ...
- **Multi-class classification:** $\mathcal{Y} = \{1, 2, \ldots, K\}$ (order is irrelevant)
  - ✓ e.g., topic classification, image classification, ...
- **Structured classification:** $\mathcal{Y}$ exponentially large and structured
  - ✓ e.g., machine translation, caption generation, image segmentation, ...

Reductions

- Sometimes reductions are convenient:

Reductions

- Sometimes reductions are convenient:

  ✓ logistic regression reduces classification to regression

Reductions

- Sometimes reductions are convenient:
  - ✓ logistic regression reduces classification to regression
  - ✓ one-vs-all reduces multi-class to binary

Reductions

- 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

Reductions

- 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

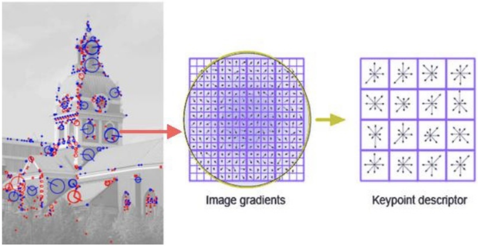
- Feature engineering is (was?) an important step for linear models:

Feature Representations

- Feature engineering is (was?) an important step for linear models:
  - ✓ Bag-of-words features for text, parts-of-speech, ...

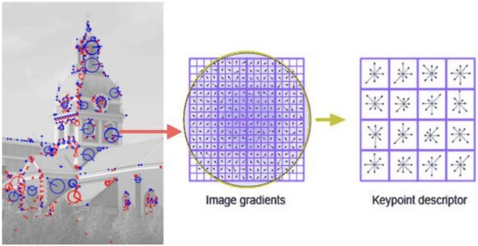
Feature Representations

- Feature engineering is (was?) an important step for linear models:
  - ✓ Bag-of-words features for text, parts-of-speech, ...
  - ✓ SIFT features and wavelet representations in computer vision



Feature Representations

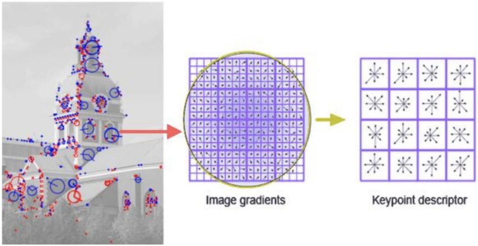
- Feature engineering is (was?) an important step for linear models:
  - ✓ Bag-of-words features for text, parts-of-speech, ...
  - ✓ SIFT features and wavelet representations in computer vision



  - ✓ Other categorical, Boolean, continuous features, ...

Feature Representations

- Feature engineering is (was?) an important step for linear models:
  - ✓ Bag-of-words features for text, parts-of-speech, ...
  - ✓ SIFT features and wavelet representations in computer vision



  - ✓ Other categorical, Boolean, continuous features, ...
  - ✓ Decades of research in machine learning, natural language processing, computer vision, image analysis, speech processing, ...

Feature Representations

- Feature represent information about an “object” x

Feature Representations

- Feature represent information about an "object" x

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

Feature Representations

- Feature represent information about an “object” x
- Typical approach: a **feature map** $\phi : \mathcal{X} \rightarrow \mathbb{R}^D$
- $\phi(x)$ is a (maybe high-dimensional) feature vector

Feature Representations

- Feature represent information about an "object" x
- Typical approach: a **feature map** $\phi : \mathcal{X} \rightarrow \mathbb{R}^D$
- $\phi(x)$ is a (maybe high-dimensional) feature vector
- Feature vectors may mix categorical and continuous features

Feature Representations

- Feature represent information about an "object" x
- Typical approach: a **feature map** $\phi : \mathcal{X} \to \mathbb{R}^D$
- $\phi(x)$ is a (maybe high-dimensional) feature vector
- Feature vectors may mix categorical and continuous features
- Categorical features can be reduced to one-hot binary features:

$$\boldsymbol{e}_y := (0, \ldots, 0, \underbrace{1}_{\text{position } y}, 0, \ldots, 0) \in \{0, 1\}^K \text{ represents class } y$$

Feature Engineering and NLP Pipelines

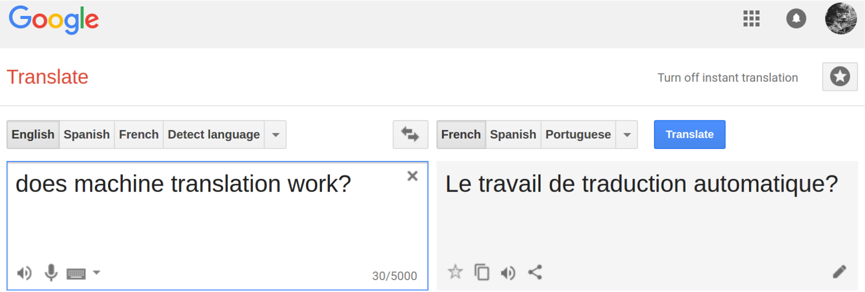
- Classical NLP pipelines consist of stacking together several linear classifiers

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

Feature Engineering and NLP Pipelines

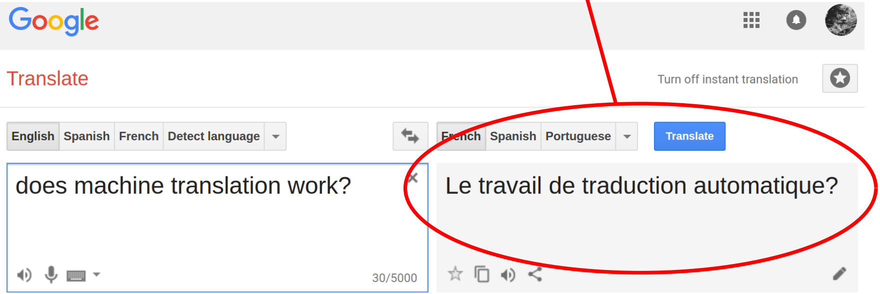
- Classical NLP pipelines consist of stacking together several linear classifiers
- Each classifier's predictions are used to handcraft features for other classifiers
- Examples of features:
  - ✓ Word occurrences (binary feature)
  - ✓ Word counts (numerical feature)
  - ✓ POS tags; e.g., adjective counts for sentiment analysis
  - ✓ Spell checker; e.g., misspellings counts for spam detection

Example: Translation Quality Estimation



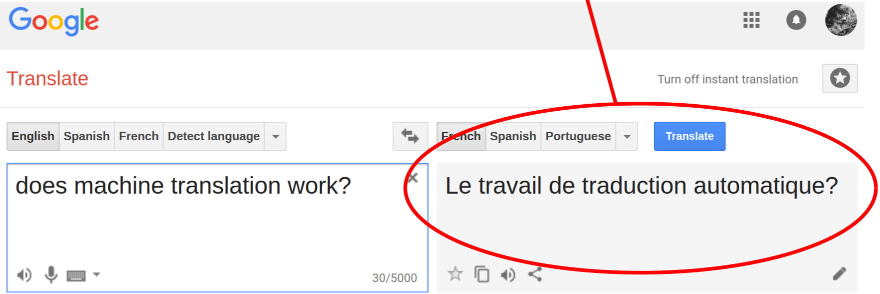
Example: Translation Quality Estimation

Wrong translation!



Example: Translation Quality Estimation

Wrong translation!



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
- language model probability of source/target segment and their ratio
- average number 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/Feature Engineering vs Learning

- Feature engineering (FE) is a "black art":
  - ✓ it can be very time-consuming
  - ✓ it requires deep domain knowledge (e.g., linguistics in NLP)

Representation/Feature Engineering vs Learning

- Feature engineering (FE) is a “black art”:
  - ✓ it can be very time-consuming
  - ✓ it requires deep domain knowledge (e.g., linguistics in NLP)
- FE allows encoding prior knowledge, it is a form of inductive bias

Representation/Feature Engineering vs Learning

- Feature engineering (FE) is a "black art":
  - ✓ it can be very time-consuming
  - ✓ it requires deep domain knowledge (e.g., linguistics in NLP)
- FE allows encoding prior knowledge, it is a form of inductive bias
- FE is still widely used in practice, specially in data-scarce scenarios

Representation/Feature Engineering vs Learning

- Feature engineering (FE) is a "black art":
  - ✓ it can be very time-consuming
  - ✓ it requires deep domain knowledge (e.g., linguistics in NLP)
- FE allows encoding prior knowledge, it is a form of inductive bias
- FE is still widely used in practice, specially in data-scarce scenarios
- Modern alternative: representation learning a.k.a. deep learning

Representation/Feature Engineering vs Learning

- Feature engineering (FE) is a "black art":
  - ✓ it can be very time-consuming
  - ✓ it requires deep domain knowledge (e.g., linguistics in NLP)
- FE allows encoding prior knowledge, it is a form of inductive bias
- FE is still widely used in practice, specially in data-scarce scenarios
- Modern alternative: representation learning a.k.a. deep learning

Tomorrow's lecture, by Bhiksha Raj

Outline

1. **Regression**

2. Classification
   - Perceptron
   - Logistic Regression
   - Support Vector Machines
   - Sparsemax

3. Regularization

4. Non-Linear Models

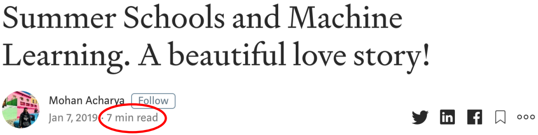
Regression

- Output is a quantity, a number, thus $\mathcal{Y} \subseteq \mathbb{R}$,

Regression

- Output is a quantity, a number, thus $\mathcal{Y} \subseteq \mathbb{R}$,

- Example: given an article, how long will a user spend reading it?

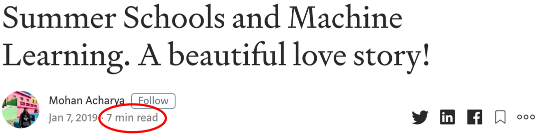


  ✓ x is number of words of the article

  ✓ y is the reading time, in minutes

Regression

- Output is a quantity, a number, thus $\mathcal{Y} \subseteq \mathbb{R}$,
- Example: given an article, how long will a user spend reading it?



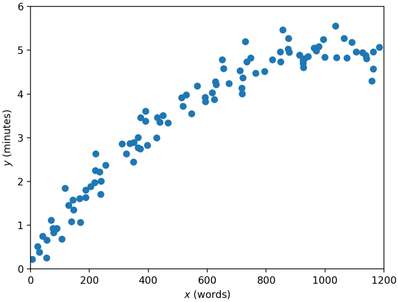
  ✓ x is number of words of the article

  ✓ y is the reading time, in minutes

- How to define a model that yields a prediction $\widehat{y}$ from x?

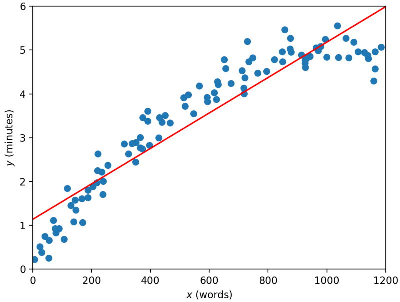
Linear Regression

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



Linear Regression

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



- Least squares (LS) criterion: fit w and b on the training set by solving

$$(\hat{w}_{\text{LS}}, \hat{b}_{\text{LS}}) = \arg\min_{w,\, b} \sum_{i=1}^N \big(y_i - (w\, x_i + b)\big)^2$$

Linear Regression

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

Linear Regression

- Often a linear dependency of $\widehat{y}$ on x is a poor assumption
- Second take: assume $\widehat{y} = \boldsymbol{w}^T \boldsymbol{\phi}(x)$, 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 b is captured by the constant feature $\phi_0(x) = 1$

Linear Regression

- Often a linear dependency of $\widehat{y}$ on x is a poor assumption
- Second take: assume $\widehat{y} = \boldsymbol{w}^T \phi(x)$, where $\phi(x)$ is a feature vector
  - ✓ e.g. $\phi(x) = [1, x, x^2, \ldots, x^D]$ (polynomial features degree $\leq D$)
  - ✓ the bias b is captured by the constant feature $\phi_0(x) = 1$
- Minimize squared loss: $\sum_i \left(y_i - \left(\boldsymbol{w}^T \phi(x_i)\right)\right)^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2$, where

$$\boldsymbol{X} = \begin{bmatrix} \phi(x_1)^\top \\ \vdots \\ \phi(x_N)^\top \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Linear Regression

- Often a linear dependency of $\widehat{y}$ on x is a poor assumption
- Second take: assume $\widehat{y} = \boldsymbol{w}^T \phi(x)$, where $\phi(x)$ is a feature vector
  - ✓ e.g. $\phi(x) = [1, x, x^2, \dots, x^D]$ (polynomial features degree $\leq D$)
  - ✓ the bias b is captured by the constant feature $\phi_0(x) = 1$
- Minimize squared loss: $\sum_i \left(y_i - \left(\boldsymbol{w}^T \phi(x_i)\right)\right)^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2$, where

$$\boldsymbol{X} = \begin{bmatrix} \phi(x_1)^\top \\ \vdots \\ \phi(x_N)^\top \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

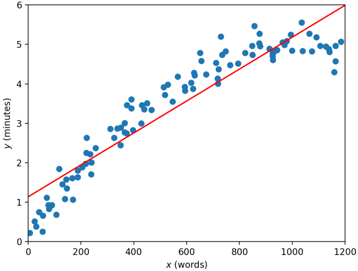
- Closed form solution: $\hat{\boldsymbol{w}}_{\text{LS}} = \arg\min_{\boldsymbol{w}} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$

Linear Regression

- Often a linear dependency of $\widehat{y}$ on x is a poor assumption
- Second take: assume $\widehat{y} = \boldsymbol{w}^T \phi(x)$, where $\phi(x)$ is a feature vector
  - ✓ e.g. $\phi(x) = [1, x, x^2, \ldots, x^D]$ (polynomial features degree $\leq D$)
  - ✓ the bias b is captured by the constant feature $\phi_0(x) = 1$
- Minimize squared loss: $\sum_i \left(y_i - \left(\boldsymbol{w}^T \phi(x_i)\right)\right)^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2$, where

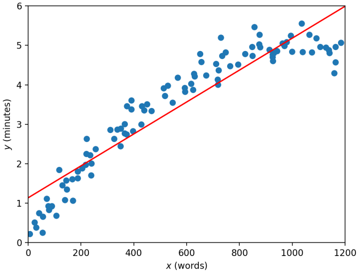
$$\boldsymbol{X} = \begin{bmatrix} \phi(x_1)^\top \\ \vdots \\ \phi(x_N)^\top \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

- Closed form solution: $\hat{\boldsymbol{w}}_{\text{LS}} = \arg\min_{\boldsymbol{w}} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$
- Still called linear regression: linear w.r.t. the model parameters $\boldsymbol{w}$.



y (minutes)
x (words)
0
1
2
3
4
5
6
0
200
400
600
800
1000
1200

Linear Regression: $D = 1$ vs $D = 2$

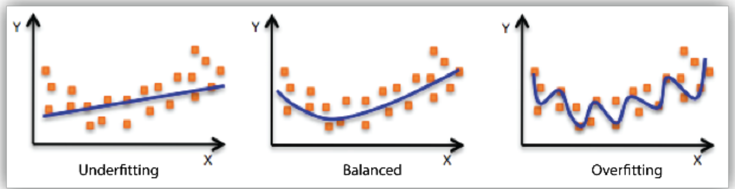


Overfitting and Underfitting

- We saw above an example of underfitting ($D = 1$).
- Choosing $D = 2$ "seems OK"

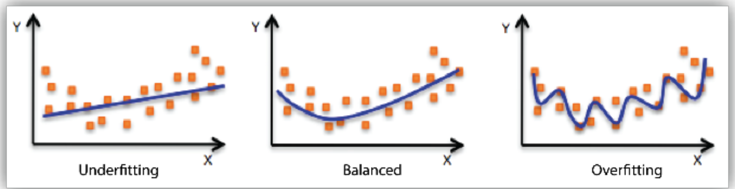
Overfitting and Underfitting

- We saw above an example of underfitting ($D = 1$).
- Choosing $D = 2$ "seems OK"
- However, if the model is too complex, overfitting may occur:



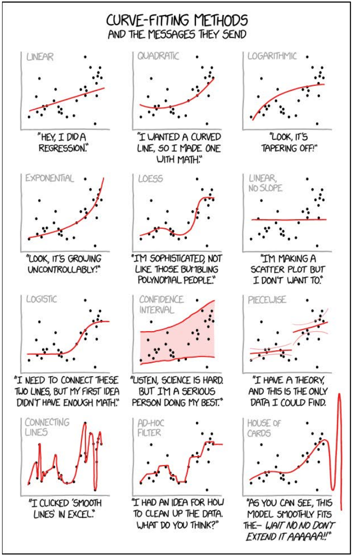
Overfitting and Underfitting

- We saw above an example of underfitting ($D = 1$).
- Choosing $D = 2$ "seems OK"
- However, if the model is too complex, overfitting may occur:



- Avoiding overfitting:
  - ✓ regularization (later)
  - ✓ some way to choose D (model complexity)

Inductive Biases



from xkcd.com

Least Squares: Probabilistic Interpretation

- The least squares criterion has a probabilistic interpretation.

- The least squares criterion has a probabilistic interpretation.
- Assume the following probabilistic observation model:

$$y_i = \boldsymbol{w}^{*T} \phi(x_i) + n_i$$

where

✓ $n_i \sim \mathcal{N}(0, \sigma^2)$ are independent Gaussian, with σ^2 fixed

✓ $\boldsymbol{w}^*$ are the "true" model parameters.

Least Squares: Probabilistic Interpretation

- The least squares criterion has a probabilistic interpretation.
- Assume the following probabilistic observation model:

$$y_i = {\boldsymbol{w}^*}^T \phi(x_i) + n_i$$

where

✓ $n_i \sim \mathcal{N}(0, \sigma^2)$ are independent Gaussian, with σ^2 fixed

✓ $\boldsymbol{w}^*$ are the "true" model parameters.

- That is, $P(y_i|x_i;\boldsymbol{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - {\boldsymbol{w}^*}^T \phi(x_i))^2}{2\sigma^2}\right)$

Least Squares: Probabilistic Interpretation

- The least squares criterion has a probabilistic interpretation.
- Assume the following probabilistic observation model:

$$y_i = \boldsymbol{w}^{*T} \phi(x_i) + n_i$$

where

✓ $n_i \sim \mathcal{N}(0, \sigma^2)$ are independent Gaussian, with σ^2 fixed

✓ $\boldsymbol{w}^*$ are the "true" model parameters.

- That is, $P(y_i | x_i; \boldsymbol{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \boldsymbol{w}^{*T}\phi(x_i))^2}{2\sigma^2}\right)$
- Then, $\hat{\boldsymbol{w}}_{\text{LS}}$ is the maximum likelihood (ML) estimate under this model.

One-Slide Proof

- Proof:

$$
\begin{aligned}
\hat{\boldsymbol{w}}_{\text{ML}} &= \arg\max_{\boldsymbol{w}} P(y_1, \ldots, y_N | x_1, \ldots., x_N; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \prod_{i=1}^{N} P(y_i \mid x_i; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \sum_{i=1}^{N} \log P(y_i \mid x_i; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \sum_{i=1}^{N} -\frac{(y_i - \boldsymbol{w}^T \phi(x_i))^2}{2\sigma^2} - \underbrace{\log(\sqrt{2\pi}\sigma)}_{\text{constant}} \\
&= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} (y_i - \boldsymbol{w}^T \phi(x_i))^2 \;=\; \hat{\boldsymbol{w}}_{\text{LS}}
\end{aligned}
$$

One-Slide Proof

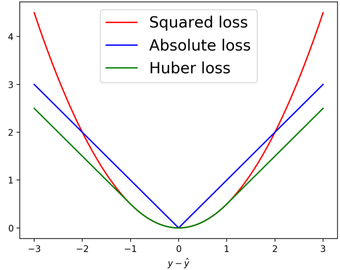
- Proof:

$$\begin{aligned}
\hat{\boldsymbol{w}}_{\mathrm{ML}} &= \arg\max_{\boldsymbol{w}} P(y_1, ..., y_N | x_1, ..., x_N; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \prod_{i=1}^{N} P(y_i \mid x_i; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \sum_{i=1}^{N} \log P(y_i \mid x_i; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \sum_{i=1}^{N} -\frac{(y_i - \boldsymbol{w}^T \phi(x_i))^2}{2\sigma^2} - \underbrace{\log(\sqrt{2\pi}\sigma)}_{\text{constant}} \\
&= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} (y_i - \boldsymbol{w}^T \phi(x_i))^2 \;=\; \hat{\boldsymbol{w}}_{\mathrm{LS}}
\end{aligned}$$

- Conclusion: LS linear regression $\Leftrightarrow$ ML under Gaussian noise.

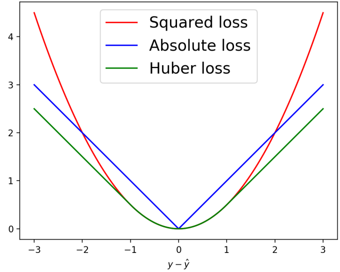
Other Regression Losses

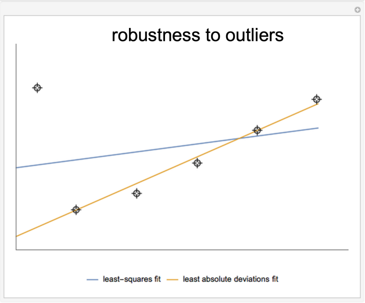
- Squared loss: $L(y, \widehat{y}) = \frac{1}{2}(y - \widehat{y})^2$.
- Absolute error loss: $L(y, \widehat{y}) = |y - \widehat{y}|$
- Huber loss: $L(y, \widehat{y}) = \begin{cases} \frac{1}{2}(y - \widehat{y})^2 & \text{if } |y - \widehat{y}| \leq 1 \\ |y - \widehat{y}| - \frac{1}{2} & \text{if } |y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



Other Regression Losses

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





Ridge Regression and Regularization

- Recall that LS linear regression has a closed form solution:

$$\hat{\boldsymbol{w}}_{\mathsf{LS}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

Ridge Regression and Regularization

- Recall that LS linear regression has a closed form solution:

$$\hat{\boldsymbol{w}}_{\text{LS}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

- What if $\boldsymbol{X}^\top \boldsymbol{X}$ is not invertible? (for example, with colinear features)

Ridge Regression and Regularization

- Recall that LS linear regression has a closed form solution:

$$\hat{\boldsymbol{w}}_{\text{LS}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

- What if $\boldsymbol{X}^\top \boldsymbol{X}$ is not invertible? (for example, with colinear features)

- Standard approach: ridge regression:

$$\hat{\boldsymbol{w}}_{\text{ridge}} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

Ridge Regression and Regularization

- Recall that LS linear regression has a closed form solution:

$$\hat{\boldsymbol{w}}_{\text{LS}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

- What if $\boldsymbol{X}^\top \boldsymbol{X}$ is not invertible? (for example, with colinear features)

- Standard approach: ridge regression:

$$\hat{\boldsymbol{w}}_{\text{ridge}} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

- This is equivalent to (with $\|\boldsymbol{w}\|_2^2 = \sum_i w_i^2$, the squared ℓ_2 norm)

$$\hat{\boldsymbol{w}}_{\text{ridge}} = \arg\min_{\boldsymbol{w}} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2 + \lambda \|\boldsymbol{w}\|_2^2$$

Ridge Regression and Regularization

- Recall that LS linear regression has a closed form solution:

$$\hat{\boldsymbol{w}}_{\text{LS}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

- What if $\boldsymbol{X}^\top \boldsymbol{X}$ is not invertible? (for example, with colinear features)

- Standard approach: ridge regression:

$$\hat{\boldsymbol{w}}_{\text{ridge}} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

- This is equivalent to (with $\|\boldsymbol{w}\|_2^2 = \sum_i w_i^2$, the squared ℓ_2 norm)

$$\hat{\boldsymbol{w}}_{\text{ridge}} = \arg\min_{\boldsymbol{w}} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2 + \lambda \|\boldsymbol{w}\|_2^2$$

- ℓ_2 regularization is also called weight decay, or penalized LS.

Maximum A Posteriori Regression

- Assume a prior distribution $\boldsymbol{w} \sim \mathcal{N}(0, \tau^2 \boldsymbol{I})$

Maximum A Posteriori Regression

- Assume a prior distribution $\boldsymbol{w} \sim \mathcal{N}(0, \tau^2 \boldsymbol{I})$
- Maximum a posteriori (MAP) criterion;

$$
\begin{aligned}
\hat{\boldsymbol{w}}_{\text{MAP}} &= \arg\max_{\boldsymbol{w}} P(\boldsymbol{w}|y_1, \ldots, y_N; x_1, \ldots, x_N) \\
&= \arg\max_{\boldsymbol{w}} \frac{P(\boldsymbol{w})\, P(y_1, \ldots, y_N | x_1, \ldots x_N; \boldsymbol{w})}{P(y_1, \ldots, y_N | x_1, \ldots x_N)} \\
&= \arg\max_{\boldsymbol{w}} \big(\log P(\boldsymbol{w}) + \log P(y_1, \ldots, y_N | x_1, \ldots x_N; \boldsymbol{w}) \big) \\
&= \arg\max_{\boldsymbol{w}} -\frac{\|\boldsymbol{w}\|^2}{2\tau^2} - \sum_{n=1}^{N} -\frac{(y_n - \boldsymbol{w}^T \phi(x_n))^2}{2\sigma^2} + \text{constant} \\
&= \arg\min_{\boldsymbol{w}} \lambda \|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} (y_n - \boldsymbol{w}^T \phi(x_n))^2 \quad (\text{with } \lambda = \sigma^2/\tau^2)
\end{aligned}
$$

Maximum A Posteriori Regression

- Assume a prior distribution $\boldsymbol{w} \sim \mathcal{N}(0, \tau^2 \boldsymbol{I})$
- Maximum a posteriori (MAP) criterion;

$$
\begin{aligned}
\hat{\boldsymbol{w}}_{\text{MAP}} &= \arg\max_{\boldsymbol{w}} P(\boldsymbol{w}|y_1, \ldots, y_N; x_1, \ldots, x_N) \\
&= \arg\max_{\boldsymbol{w}} \frac{P(\boldsymbol{w})\, P(y_1, \ldots, y_N | x_1, \ldots x_N; \boldsymbol{w})}{P(y_1, \ldots, y_N | x_1, \ldots x_N)} \\
&= \arg\max_{\boldsymbol{w}} \big(\log P(\boldsymbol{w}) + \log P(y_1, \ldots, y_N | x_1, \ldots x_N; \boldsymbol{w})\big) \\
&= \arg\max_{\boldsymbol{w}} -\frac{\|\boldsymbol{w}\|^2}{2\tau^2} - \sum_{n=1}^{N} -\frac{(y_n - \boldsymbol{w}^T \phi(x_n))^2}{2\sigma^2} + \text{constant} \\
&= \arg\min_{\boldsymbol{w}} \lambda \|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} (y_n - \boldsymbol{w}^T \phi(x_n))^2 \quad (\text{with } \lambda = \sigma^2/\tau^2)
\end{aligned}
$$

- Conclusion: ℓ_2 regularizarion $\Leftrightarrow$ MAP regression with Gaussian prior.

Ridge Regression: Optimal λ

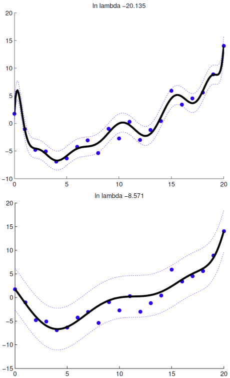
- Even if $\hat{\boldsymbol{w}}_{\text{LS}}$ can be computed, $\hat{\boldsymbol{w}}_{\text{ridge}}$ may be better.

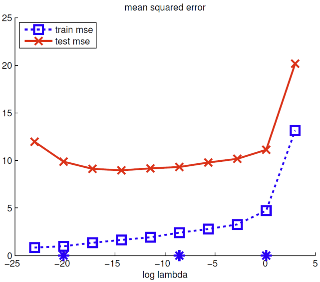
Ridge Regression: Optimal λ

- Even if $\hat{\boldsymbol{w}}_{\text{LS}}$ can be computed, $\hat{\boldsymbol{w}}_{\text{ridge}}$ may be better.
- Example: fitting an order-14 polynomial to 21 points,

Ridge Regression: Optimal λ

- Even if $\hat{\boldsymbol{w}}_{\mathrm{LS}}$ can be computed, $\hat{\boldsymbol{w}}_{\mathrm{ridge}}$ may be better.
- Example: fitting an order-14 polynomial to 21 points,





Outline

Binary Classification

- Before multi-class classification, we look at binary classification

Binary Classification

- Before multi-class classification, we look at binary classification

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

Binary Classification

- Before multi-class classification, we look at binary classification
- Output set $\mathcal{Y} = \{-1, +1\}$
- Example: Given a news article, is it true or fake?
  - ✓ x is the news article, represented a feature vector $\phi(x)$
  - ✓ y can be either true $(+1)$ or fake (-1)

Binary Classification

- Before multi-class classification, we look at binary classification

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

- Example: Given a news article, is it true or fake?
  - ✓ x is the news article, represented a feature vector $\phi(x)$
  - ✓ y can be either true $(+1)$ or fake (-1)

- How to define a model to predict y from x?

Linear Classifier

- Defined by

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

Linear Classifier

- Defined by

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

- Intuitively, $\boldsymbol{w}^T \phi(x) + b$ is a "score" for the positive class

Linear Classifier

- Defined by

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

- Intuitively, $\boldsymbol{w}^T \phi(x) + b$ is a "score" for the positive class
- The sign function converts from continuous to binary

Linear Classifier

- Defined by

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

- Intuitively, $\boldsymbol{w}^T \phi(x) + b$ is a "score" for the positive class
- The sign function converts from continuous to binary
- Decision boundary: $\boldsymbol{w}^T \phi(x) + b = 0$ (hyperplane defined by $\boldsymbol{w}$ and b)

Linear Classifier

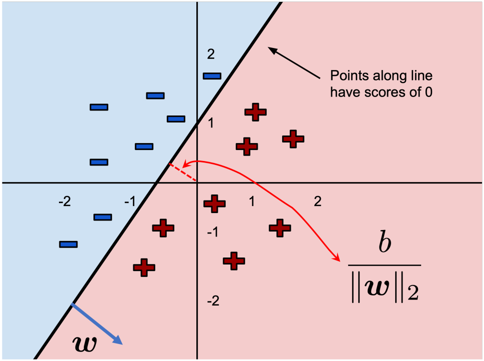
- Defined by

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

- Intuitively, $\boldsymbol{w}^T \phi(x) + b$ is a "score" for the positive class
- The sign function converts from continuous to binary
- Decision boundary: $\boldsymbol{w}^T \phi(x) + b = 0$ (hyperplane defined by $\boldsymbol{w}$ and b)
- Also called a hyperplane classifier

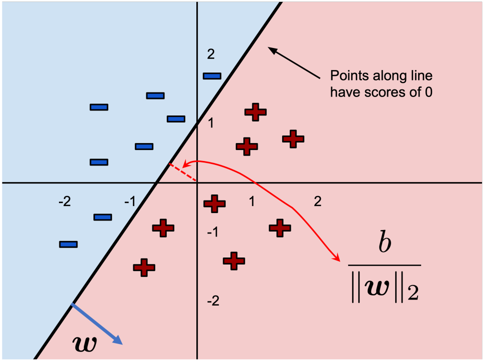
Linear Classifier

- $(\boldsymbol{w}, b)$ define an hyperplane that splits the space into two halfs



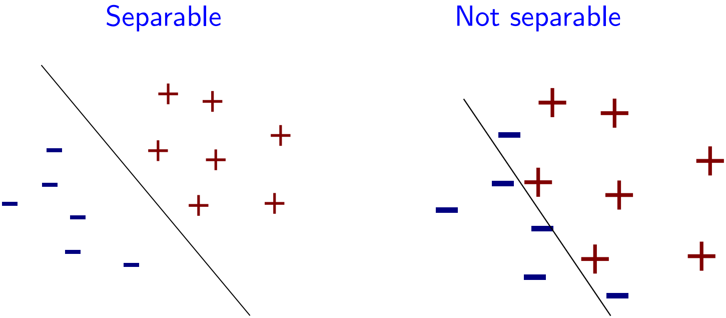
Linear Classifier

- $(\boldsymbol{w}, b)$ define an hyperplane that splits the space into two halfs



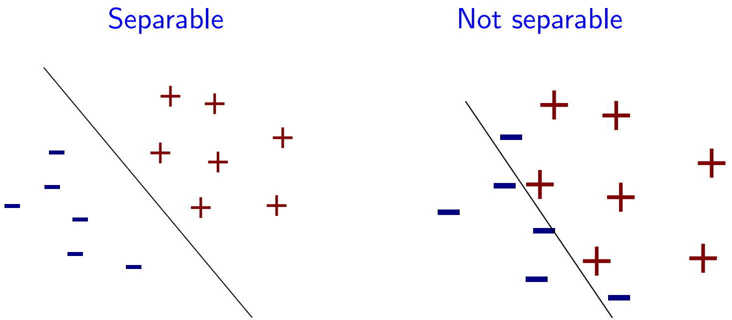
- How to learn it from training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$?

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



Linear Separability

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



- We next present an (old!) algorithm that finds such an hyperplane, if it exists.

Linear Classifier: No Bias Term

- It is common to ommit the bias term b: $\widehat{y} = \text{sign}(\boldsymbol{w}^T \phi(x))$

Linear Classifier: No Bias Term

- It is common to ommit the bias term b: $\widehat{y} = \text{sign}(\boldsymbol{w}^T \phi(x))$
- In this case, the decision boundary is a hyperplane that passes through the origin

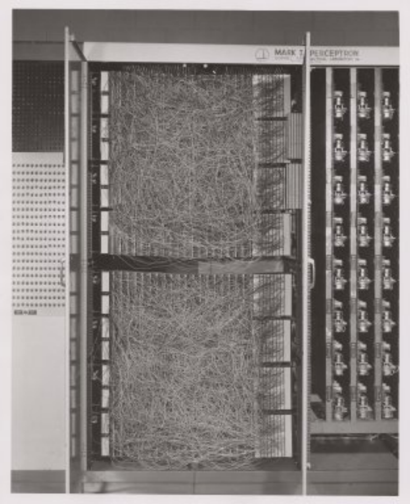
Linear Classifier: No Bias Term

- It is common to ommit the bias term b: $\widehat{y} = \text{sign}(\boldsymbol{w}^T \phi(x))$
- In this case, the decision boundary is a hyperplane that passes through the origin
- There is no loss of generality:
  - ✓ Add a constant feature to $\phi(x)$: $\phi_0(x) = 1$
  - ✓ The corresponding weight w_0 is a bias term b

Outline

1. Regression
2. **Classification**
   - Perceptron
   - Logistic Regression
   - Support Vector Machines
   - Sparsemax
3. Regularization
4. Non-Linear Models

Perceptron (Rosenblatt, 1958)



(Extracted from Wikipedia)

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

Perceptron in the News...

1958 New York Times...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

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

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

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

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

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

Without Human Controls

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

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

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

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

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

Learns by Doing

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

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

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

Perceptron in the News...

1958 New York Times...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

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

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

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

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

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

Without Human Controls

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

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

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

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

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

Learns by Doing

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

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

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

Perceptron Algorithm

- Online algorithm: process one data point at each round

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

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

Perceptron's Mistake Bound

- Some 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

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

Perceptron's Mistake Bound

- Some 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
  $$y_i \, \boldsymbol{u}^T \phi(x_i) \geq \gamma, \quad \forall i.$$
  - ✓ radius of the data: $R = \max_i \|\phi(x_i)\|$.

Perceptron's Mistake Bound

- Some 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

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

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

- Then, the following bound of the number of mistakes holds:

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 $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + y_i \phi(x_i)$ and that $\|\boldsymbol{u}\| = 1$
- **Lower bound on** $\|\boldsymbol{w}^{(k+1)}\|$:

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

Thus: $\|\boldsymbol{w}^{(k+1)}\| = \|\boldsymbol{u}\| \, \|\boldsymbol{w}^{(k+1)}\| \geq \boldsymbol{u}^T \boldsymbol{w}^{(k+1)} \geq k\gamma$ (Cauchy-Schwarz)

One-Slide Proof

- Recall that $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + y_i \phi(x_i)$ and that $\|\boldsymbol{u}\| = 1$
- **Lower bound on $\|\boldsymbol{w}^{(k+1)}\|$:**

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

Thus: $\|\boldsymbol{w}^{(k+1)}\| = \|\boldsymbol{u}\|\,\|\boldsymbol{w}^{(k+1)}\| \geq \boldsymbol{u}^T \boldsymbol{w}^{(k+1)} \geq k\gamma$ (Cauchy-Schwarz)

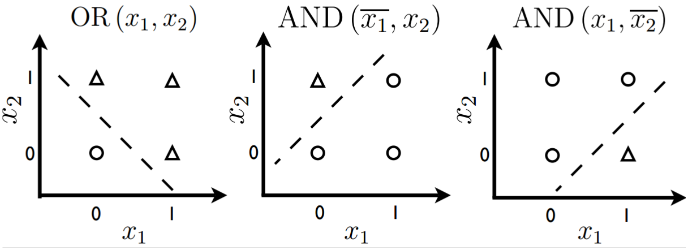
- **Upper bound on $\|\boldsymbol{w}^{(k+1)}\|$:**

$$\begin{aligned}
\|\boldsymbol{w}^{(k+1)}\|^2 &= \|\boldsymbol{w}^{(k)}\|^2 + \|\phi(x_i)\|^2 + 2\,\overbrace{y_i {\boldsymbol{w}^{(k)}}^T \phi(x_i)}^{<0} \\
&\leq \|\boldsymbol{w}^{(k)}\|^2 + R^2 \\
&\leq kR^2.
\end{aligned}$$

- Equating both sides: $(k\gamma)^2 \leq kR^2 \;\Rightarrow\; k \leq R^2/\gamma^2$ (QED).

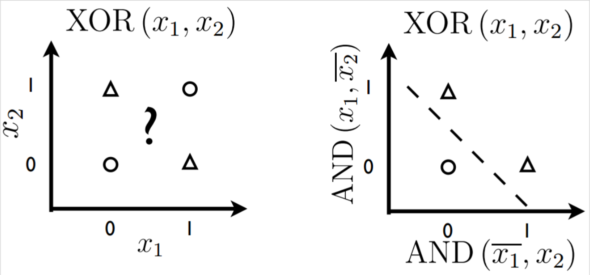
What a Simple Perceptron Can and Can't Do

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



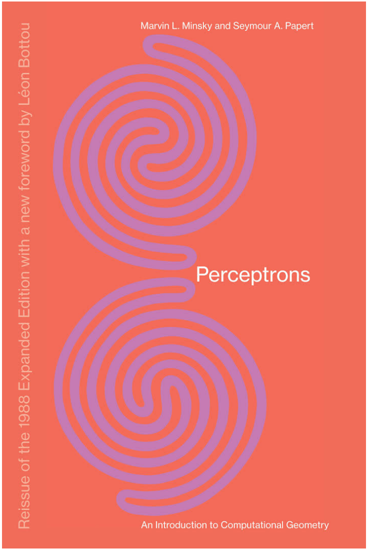
What a Simple Perceptron Can and Can't Do

- ... 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 (1996) showed limitations of multi-layer perceptrons and fostered an "AI winter" period.

Multi-Class Classification

- Consider multi-class problems, with $|\mathcal{Y}| = K \geq 2$ labels (classes).

Multi-Class Classification

- Consider multi-class problems, with $|\mathcal{Y}| = K \geq 2$ labels (classes).

- Reduction approaches:

  ✓ One-vs-all (OVA): one binary classifier per label, with all the other classes as negative examples. Choose the class with the highest score.

Multi-Class Classification

- Consider multi-class problems, with $|\mathcal{Y}| = K \geq 2$ labels (classes).

- Reduction approaches:

  ✓ One-vs-all (OVA): one binary classifier per label, with all the other classes as negative examples. Choose the class with the highest score.

  ✓ One-vs-one (OVO): train $K(K-1)/2$ pairwise classifiers and use majority voting.

Multi-Class Classification

- Consider multi-class problems, with $|\mathcal{Y}| = K \geq 2$ labels (classes).
- Reduction approaches:
  - ✓ One-vs-all (OVA): one binary classifier per label, with all the other classes as negative examples. Choose the class with the highest score.
  - ✓ One-vs-one (OVO): train $K(K-1)/2$ pairwise classifiers and use majority voting.
  - ✓ Error correcting codes (ECoC): use a redundant binary code for each class and train one classifier per bit.

Multi-Class Classification

- Consider multi-class problems, with $|\mathcal{Y}| = K \geq 2$ labels (classes).
- Reduction approaches:
  - ✓ One-vs-all (OVA): one binary classifier per label, with all the other classes as negative examples. Choose the class with the highest score.
  - ✓ One-vs-one (OVO): train $K(K-1)/2$ pairwise classifiers and use majority voting.
  - ✓ Error correcting codes (ECoC): use a redundant binary code for each class and train one classifier per bit.
- Here, we consider classifiers that tackle the multiple classes directly.

Multi-Class Linear Classifiers

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

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^T \\ \vdots \\ \boldsymbol{w}_K^T \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix}.$$

Multi-Class Linear Classifiers

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

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^T \\ \vdots \\ \boldsymbol{w}_K^T \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix}.$$

- Equivalently, K weight vectors $\boldsymbol{w}_y \in \mathbb{R}^D$ and K scalars $b_y \in \mathbb{R}$

Multi-Class Linear Classifiers

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

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^T \\ \vdots \\ \boldsymbol{w}_K^T \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix}.$$

- Equivalently, K weight vectors $\boldsymbol{w}_y \in \mathbb{R}^D$ and K scalars $b_y \in \mathbb{R}$

- Score of each class: linear combination of features and their weights

Multi-Class Linear Classifiers

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

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^T \\ \vdots \\ \boldsymbol{w}_K^T \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix}.$$

- Equivalently, K weight vectors $\boldsymbol{w}_y \in \mathbb{R}^D$ and K scalars $b_y \in \mathbb{R}$
- Score of each class: linear combination of features and their weights
- Predict the $\widehat{y}$ which maximizes the score:

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

Multi-Class Linear Classifiers

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

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^T \\ \vdots \\ \boldsymbol{w}_K^T \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix}.$$

- Equivalently, K weight vectors $\boldsymbol{w}_y \in \mathbb{R}^D$ and K scalars $b_y \in \mathbb{R}$

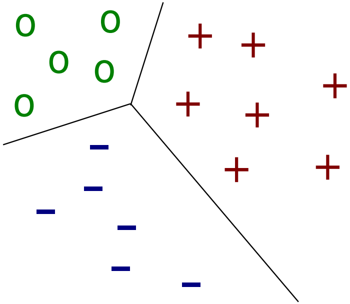
- Score of each class: linear combination of features and their weights

- Predict the $\widehat{y}$ which maximizes the score:

$$\widehat{y} = \arg\max_{y \in \mathcal{Y}} \ \boldsymbol{w}_y{}^T \phi(x) + b_y = \arg\max(\boldsymbol{W}\phi(x) + \boldsymbol{b})$$

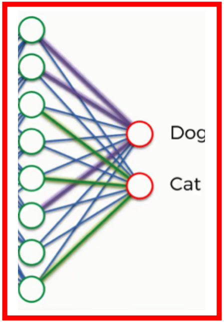
Multi-Class Linear Classifier

- $(\boldsymbol{W}, \boldsymbol{b})$ split the feature space into regions delimited by hyperplanes.
- Each region in the intersection of $K-1$ half-spaces.



Commonly Used Notation in Neural Networks

Handcrafted Features



Linear Classifier

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

Multi-Class Recovers Binary

- With two classes (e.g. $\mathcal{Y} = \{+1, -1\}$), we recover the binary classifier:

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

Multi-Class Recovers Binary

- With two classes (e.g. $\mathcal{Y} = \{+1, -1\}$), we recover the binary classifier:

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

Multi-Class Recovers Binary

- With two classes (e.g. $\mathcal{Y} = \{+1, -1\}$), we recover the binary classifier:

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

Multi-Class Recovers Binary

- With two classes (e.g. $\mathcal{Y} = \{+1, -1\}$), we recover the binary classifier:

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

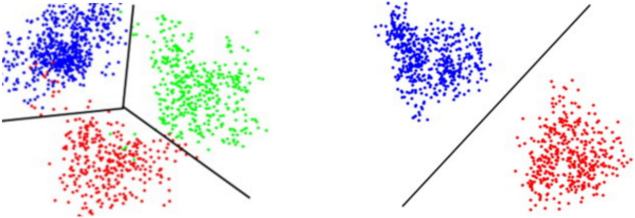
- Only half of the parameters are needed.

Linear Classifiers (Binary vs Multi-Class)

- Prediction rule (omitting the bias term, without loss of generality):

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

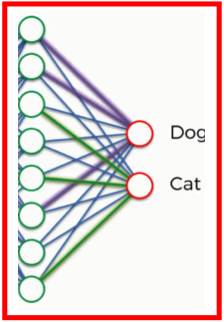
- The decision boundary is defined by the intersection of half spaces
- In the binary case ($|\mathcal{Y}| = 2$) this corresponds to a hyperplane classifier



Perceptron Algorithm: Multi-Class

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

Reminder



Linear Classifier

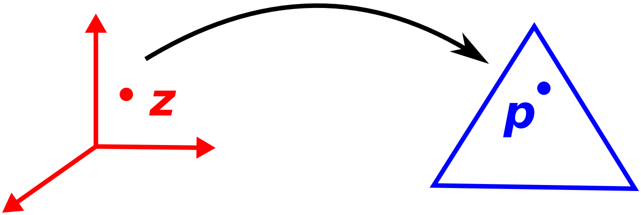
$$\widehat{y} = \operatorname{argmax}\left(\boldsymbol{W}\phi(x) + \boldsymbol{b}\right), \quad \boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^\top \\ \vdots \\ \boldsymbol{w}_K^\top \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix}.$$

Class Probabilities

- What if we need/want class probabilities?

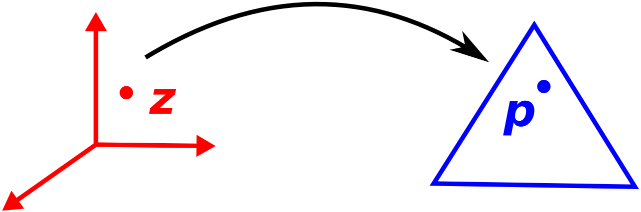
Class Probabilities

- What if we need/want class probabilities?
- How to map fro K label scores to a probability distribution over $\mathcal{Y}$?



Class Probabilities

- What if we need/want class probabilities?
- How to map fro K label scores to a probability distribution over $\mathcal{Y}$?



- Two possible mappings: softmax, a.k.a. logistic regression (next) and sparsemax (later).

Outline

Logistic Regression

- Recall: a linear model gives score ${w_y}^T \phi(x)$ for class y

Logistic Regression

- Recall: a linear model gives score ${\boldsymbol{w}_y}^T \phi(x)$ for class y
- Mapping scores to posterior class conditional probabilities:

$$P(y|x) = \frac{\exp({\boldsymbol{w}_y}^T \phi(x))}{Z_x}, \qquad \text{where } Z_x = \sum_{y' \in \mathcal{Y}} \exp({\boldsymbol{w}_{y'}}^T \phi(x))$$

Logistic Regression

- Recall: a linear model gives score ${\boldsymbol{w}_y}^T \phi(x)$ for class y
- Mapping scores to posterior class conditional probabilities:

$$P(y|x) = \frac{\exp({\boldsymbol{w}_y}^T \phi(x))}{Z_x}, \qquad \text{where } Z_x = \sum_{y' \in \mathcal{Y}} \exp({\boldsymbol{w}_{y'}}^T \phi(x))$$

- Softmax transformation: exponentiation followed by normalization.

Logistic Regression

- Recall: a linear model gives score ${\boldsymbol{w}_y}^T \phi(x)$ for class y
- Mapping scores to posterior class conditional probabilities:

$$P(y|x) = \frac{\exp({\boldsymbol{w}_y}^T \phi(x))}{Z_x}, \qquad \text{where } Z_x = \sum_{y' \in \mathcal{Y}} \exp({\boldsymbol{w}_{y'}}^T \phi(x))$$

- Softmax transformation: exponentiation followed by normalization.
- Adding a constant to all the scores does not change the probabilities.

Logistic Regression

- Recall: a linear model gives score ${\boldsymbol{w}_y}^T \phi(x)$ for class y
- Mapping scores to posterior class conditional probabilities:

$$P(y|x) = \frac{\exp({\boldsymbol{w}_y}^T \phi(x))}{Z_x}, \qquad \text{where } Z_x = \sum_{y' \in \mathcal{Y}} \exp({\boldsymbol{w}_{y'}}^T \phi(x))$$

- Softmax transformation: exponentiation followed by normalization.
- Adding a constant to all the scores does not change the probabilities.
- Z_x doesn't depend on y: still a linear classifier. E.g., the MAP rule,

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

Logistic Regression

- Recall: a linear model gives score ${\boldsymbol{w}_y}^T \phi(x)$ for class y
- Mapping scores to posterior class conditional probabilities:

$$P(y|x) = \frac{\exp({\boldsymbol{w}_y}^T \phi(x))}{Z_x}, \qquad \text{where } Z_x = \sum_{y' \in \mathcal{Y}} \exp({\boldsymbol{w}_{y'}}^T \phi(x))$$

- Softmax transformation: exponentiation followed by normalization.
- Adding a constant to all the scores does not change the probabilities.
- Z_x doesn't depend on y: still a linear classifier. E.g., the MAP rule,

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

- Allows for cost-sensitive decisions, beyond simple MAP.

Binary Logistic Regression

- Binary case: $\mathcal{Y} = \{\pm 1\}$

Binary Logistic Regression

- Binary case: $\mathcal{Y} = \{\pm 1\}$
- Scores: 0 for $y = -1$ and $\boldsymbol{w}^T \phi(x)$ for $y = 1$

$$
\begin{aligned}
P(y = +1 \mid x) &= \frac{\exp(\boldsymbol{w}^T \phi(x))}{\exp(0) + \exp(\boldsymbol{w}^T \phi(x))} \\
&= \frac{1}{1 + \exp(-\boldsymbol{w}^T \phi(x))} \\
&\equiv \sigma(\boldsymbol{w}^T \phi(x)).
\end{aligned}
$$

Binary Logistic Regression

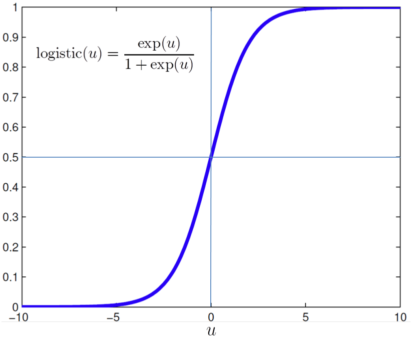
- Binary case: $\mathcal{Y} = \{\pm 1\}$
- Scores: 0 for $y = -1$ and $\boldsymbol{w}^T \phi(x)$ for $y = 1$

$$\begin{aligned} P(y = +1 \mid x) &= \frac{\exp(\boldsymbol{w}^T \phi(x))}{\exp(0) + \exp(\boldsymbol{w}^T \phi(x))} \\ &= \frac{1}{1 + \exp(-\boldsymbol{w}^T \phi(x))} \\ &\equiv \sigma(\boldsymbol{w}^T \phi(x)). \end{aligned}$$

- Sigmoid, or logistic, transformation (more later!)

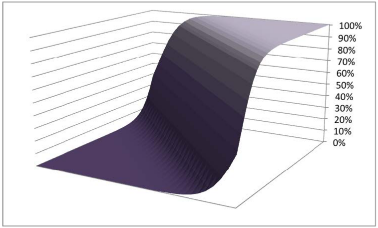
Sigmoid/Logistic Transformation

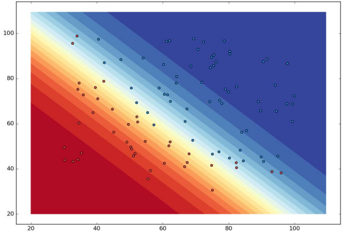
$$\boxed{\sigma(z) = \frac{1}{1 + e^{-z}}}$$



- Widely used in neural networks (more tomorrow!)
- "Squashes" a real number into [0, 1]
- The output can be interpreted as a probability
- Positive, bounded, strictly increasing, differentiable

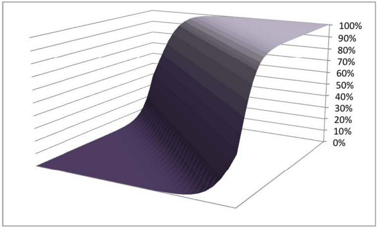
- In two dimensions, i.e., $\boldsymbol{w}$, $\phi(x) \in \mathbb{R}^2$

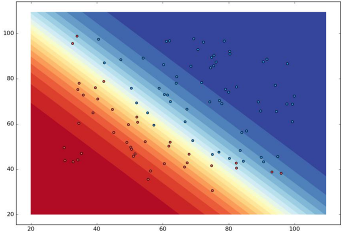




Binary Logistic Regression

- In two dimensions, i.e., $\boldsymbol{w}$, $\phi(x) \in \mathbb{R}^2$

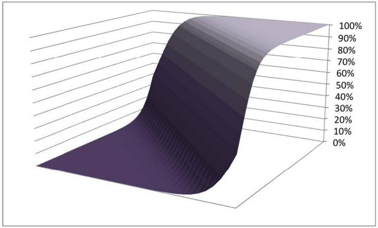


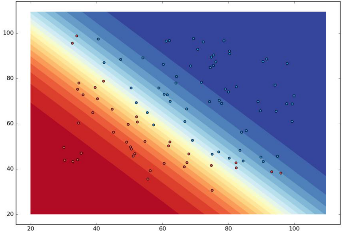


- MAP boundary, $P(y = +1 \mid x) = 1/2 \;\Leftrightarrow\; \boldsymbol{w}^T\phi(x) = 0$, is linear w.r.t. $\phi(x)$.

Binary Logistic Regression

- In two dimensions, i.e., $\boldsymbol{w}$, $\phi(x) \in \mathbb{R}^2$





- MAP boundary, $P(y = +1 \mid x) = 1/2 \;\Leftrightarrow\; \boldsymbol{w}^T \phi(x) = 0$, is linear w.r.t. $\phi(x)$.
- Some other threshold, $P(y = +1 \mid x) = \tau \;\Leftrightarrow\; \boldsymbol{w}^T \phi(x) = \log(\frac{\tau}{1-\tau})$; linear w.r.t. $\phi(x)$.

Multinomial Logistic Regression

- Recall $\boldsymbol{W} = [\boldsymbol{w}_1, ..., \boldsymbol{w}_K] \in \mathbb{R}^{K \times D}$ and $P_{\boldsymbol{W}}(y|x) = \frac{\exp(\boldsymbol{w}_y{}^T \phi(x))}{\sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x))}$

- How do we learn weights $\boldsymbol{W}$?

Multinomial Logistic Regression

- Recall $\boldsymbol{W} = [\boldsymbol{w}_1, \dots, \boldsymbol{w}_K] \in \mathbb{R}^{K \times D}$ and $P_{\boldsymbol{W}}(y|x) = \frac{\exp(\boldsymbol{w}_y{}^T \phi(x))}{\sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x))}$

- How do we learn weights $\boldsymbol{W}$?

- Maximize the conditional log-likelihood, given training data:

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

Multinomial Logistic Regression

- Recall $\boldsymbol{W} = [\boldsymbol{w}_1, ..., \boldsymbol{w}_K] \in \mathbb{R}^{K \times D}$ and $P_{\boldsymbol{W}}(y|x) = \frac{\exp(\boldsymbol{w}_y{}^T \phi(x))}{\sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x))}$

- How do we learn weights $\boldsymbol{W}$?

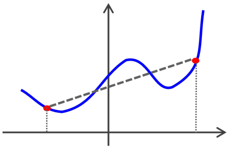
- Maximize the conditional log-likelihood, given training data:

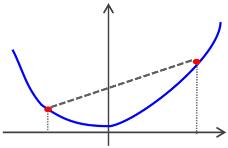
$$\begin{aligned} \widehat{\boldsymbol{W}} &= \arg\max_{\boldsymbol{W}} \log \left(\prod_{t=1}^{N} P_{\boldsymbol{W}}(y_t|x_t) \right) = \arg\min_{\boldsymbol{W}} -\sum_{t=1}^{N} \log P_{\boldsymbol{W}}(y_t|x_t) = \\ &= \arg\min_{\boldsymbol{W}} \sum_{t=1}^{N} \left(\log \sum_{y'_t} \exp(\boldsymbol{w}_{y'_t}{}^T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

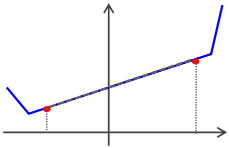
- $\widehat{\boldsymbol{W}}$ is set to assign as much probability as possible to the correct labels!

Logistic Regression

- This objective function is strictly convex

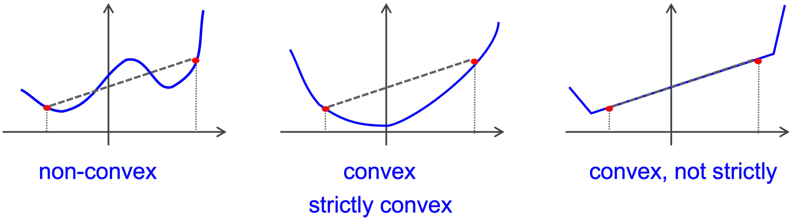

non-convex


convex
strictly convex


convex, not strictly

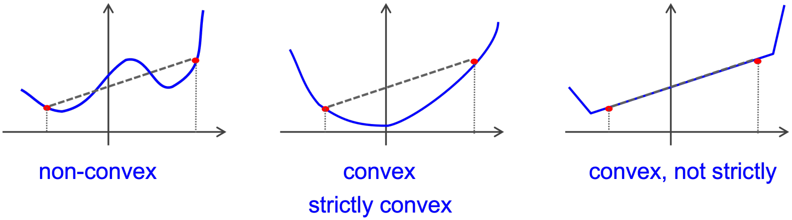
Logistic Regression

- This objective function is strictly convex



- Proof left as exercise! (hint, compute second derivatives, *i.e.*, Hessian)

- This objective function is strictly convex

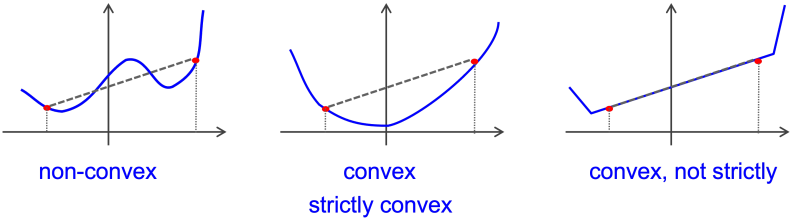


- Proof left as exercise! (hint, compute second derivatives, *i.e.*, Hessian)

- Therefore any local minimum is a global minimum

Logistic Regression

- This objective function is strictly convex



- Proof left as exercise! (hint, compute second derivatives, *i.e.*, Hessian)
- Therefore any local minimum is a global minimum
- No closed form solution, but many numerical techniques
  - ✓ Gradient methods (gradient descent, conjugate gradient)
  - ✓ Quasi-Newton methods (L-BFGS, ...)

Recap: Gradient Descent

- Goal: minimize $f : \mathbb{R}^d \to \mathbb{R}$, for differentiable objective function f

Recap: Gradient Descent

- Goal: minimize $f : \mathbb{R}^d \to \mathbb{R}$, for differentiable objective function f
- Take small steps in the negative gradient direction until a stopping criterion is met:

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

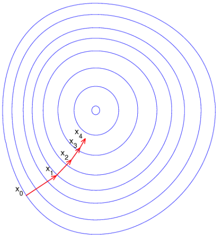
Recap: Gradient Descent

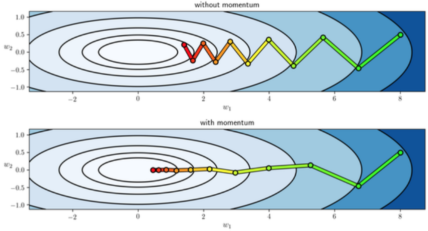
- Goal: minimize $f : \mathbb{R}^d \to \mathbb{R}$, for differentiable objective function f
- Take small steps in the negative gradient direction until a stopping criterion is met:
$$x^{(t+1)} \leftarrow x^{(t)} - \eta_{(t)} \nabla f(x^{(t)})$$
- Choosing the step-size: crucial for convergence and performance.

- Goal: minimize $f : \mathbb{R}^d \to \mathbb{R}$, for differentiable objective function f
- Take small steps in the negative gradient direction until a stopping criterion is met:

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

- Choosing the step-size: crucial for convergence and performance.
- GD may work well, or not so well. There are many ways to improve it.





Gradient Descent

- Objective function in logistic regression:

$$\sum_{t=1}^{N} L(\boldsymbol{W};(x_t,y_t)) = \sum_{t=1}^{N}\left(\log\sum_{y'}\exp({\boldsymbol{w}_{y'}}^T\phi(x)) - {\boldsymbol{w}_y}^T\phi(x)\right)$$

Gradient Descent

- Objective function in logistic regression:

$$\sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t)) = \sum_{t=1}^{N} \left(\log \sum_{y'} \exp({\boldsymbol{w}_{y'}}^T \phi(x)) - {\boldsymbol{w}_y}^T \phi(x) \right)$$

- Gradient descent:

  ✓ Set $\boldsymbol{W}^{(0)} = 0$

  ✓ Iterate until convergence (for suitable stepsize η_k):

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

Gradient Descent

- Objective function in logistic regression:

$$\sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t)) = \sum_{t=1}^{N} \Big(\log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T} \phi(x)) - \boldsymbol{w}_{y}{}^{T} \phi(x) \Big)$$

- Gradient descent:
  - ✓ Set $\boldsymbol{W}^{(0)} = 0$
  - ✓ Iterate until convergence (for suitable stepsize η_k):

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

- $\nabla_{\boldsymbol{W}} L(\boldsymbol{W}^{(k)})$ is gradient of w.r.t. $\boldsymbol{W}$, computed at $\boldsymbol{W}^{(k)}$

Gradient Descent

- Objective function in logistic regression:

$$\sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t)) = \sum_{t=1}^{N} \left(\log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x)) - \boldsymbol{w}_y{}^T \phi(x) \right)$$

- Gradient descent:
  - ✓ Set $\boldsymbol{W}^{(0)} = 0$
  - ✓ Iterate until convergence (for suitable stepsize η_k):

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

- $\nabla_{\boldsymbol{W}} L(\boldsymbol{W}^{(k)})$ is gradient of w.r.t. $\boldsymbol{W}$, computed at $\boldsymbol{W}^{(k)}$
- L convex $\Rightarrow$ gradient descent converges to global optimum

Stochastic Gradient Descent

- Stochastic approximation of the gradient (more frequent updates, convenient with large datasets)

Stochastic Gradient Descent

- Stochastic approximation of the gradient (more frequent updates, convenient with large datasets)

- Set $\boldsymbol{W}^{(0)} = 0$ and iterate until convergence:
  - ✓ Pick (x_t, y_t) randomly
  - ✓ Update $\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(k)} - \eta_k \nabla_{\boldsymbol{W}} L(\boldsymbol{W}^{(k)}; (x_t, y_t))$

Stochastic Gradient Descent

- Stochastic approximation of the gradient (more frequent updates, convenient with large datasets)
- Set $\boldsymbol{W}^{(0)} = 0$ and 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.* approximate the gradient with noisy, unbiased, version using a single sample

Stochastic Gradient Descent

- Stochastic approximation of the gradient (more frequent updates, convenient with large datasets)

- Set $\boldsymbol{W}^{(0)} = 0$ and 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.* approximate the gradient with noisy, unbiased, version using a single sample

- Variants exist in-between batch and stochastic: mini-batches

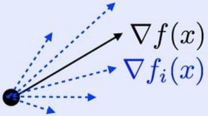
Stochastic Gradient Descent

- Stochastic approximation of the gradient (more frequent updates, convenient with large datasets)
- Set $\boldsymbol{W}^{(0)} = 0$ and 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.* approximate the gradient with noisy, unbiased, version using a single sample
- Variants exist in-between batch and stochastic: mini-batches
- All guaranteed to find the optimal $\boldsymbol{W}$ (for suitable step sizes)

SGD: Visual Summary

Finite sums

$$f(x) \overset{\text{def.}}{=} \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$
$$\nabla f(x) = \frac{1}{n}\sum_i \nabla f_i(x)$$

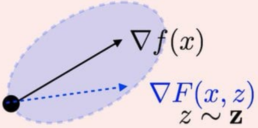


Draw $i \in \{1, \dots, n\}$ uniformly.

$$x_{k+1} = x_k - \tau_k \nabla f_i(x_k)$$

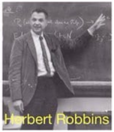
Expectation

$$f(x) \overset{\text{def.}}{=} \mathbb{E}_{\mathbf{z}}(f(x, \mathbf{z}))$$
$$\nabla f(x) = \mathbb{E}_{\mathbf{z}}(\nabla F(x, \mathbf{z}))$$



Draw $z \sim \mathbf{z}$

$$x_{k+1} = x_k - \tau_k \nabla F(x, z)$$



Theorem: If f is strongly convex and $\tau_k \sim 1/k$,

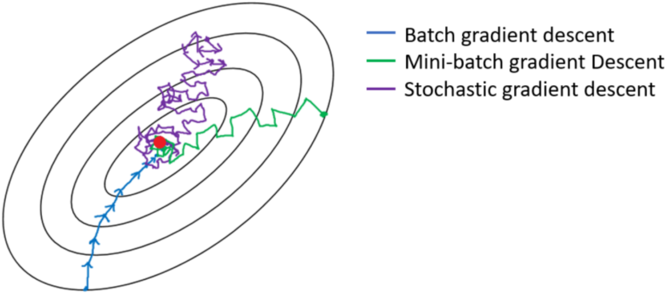
$$\mathbb{E}(\|x_k - x^\star\|^2) = O(1/k)$$

Figure by Gabriel Peyre. Highly recommended: twitter.com/gabrielpeyre

Batch, Stochastic, and Minibatch Gradient Descent

- Minibatch: instead of single sample, sample subset $B \subset \{1, \ldots, N\}$.
- Use average gradient on minibatch:

$$\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(k)} - \eta_k \frac{1}{|B|} \sum_{t \in B} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}^{(k)}; (x_t, y_t))$$



Computing the Gradient

- All this requires computing $\nabla_{\boldsymbol{W}} L(\boldsymbol{W}; (x_t, y_t))$, where

$$L(\boldsymbol{W}; (x, y)) = \log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x)) - \boldsymbol{w}_y{}^T \phi(x)$$

Computing the Gradient

- All this requires computing $\nabla_{\boldsymbol{W}} L(\boldsymbol{W}; (x_t, y_t))$, where

$$L(\boldsymbol{W}; (x, y)) = \log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x)) - \boldsymbol{w}_y{}^T \phi(x)$$

- Some reminders:

✓ $\nabla_{\boldsymbol{W}} \log F(\boldsymbol{W}) = \frac{1}{F(\boldsymbol{W})} \nabla_{\boldsymbol{W}} F(\boldsymbol{W})$

✓ $\nabla_{\boldsymbol{W}} \exp F(\boldsymbol{W}) = \exp(F(\boldsymbol{W})) \nabla_{\boldsymbol{W}} F(\boldsymbol{W})$

Computing the Gradient

- All this requires computing $\nabla_{\boldsymbol{W}} L(\boldsymbol{W};(x_t, y_t))$, where

$$L(\boldsymbol{W};(x,y)) = \log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x)) - \boldsymbol{w}_y{}^T \phi(x)$$

- Some reminders:

  ✓ $\nabla_{\boldsymbol{W}} \log F(\boldsymbol{W}) = \frac{1}{F(\boldsymbol{W})} \nabla_{\boldsymbol{W}} F(\boldsymbol{W})$

  ✓ $\nabla_{\boldsymbol{W}} \exp F(\boldsymbol{W}) = \exp(F(\boldsymbol{W})) \nabla_{\boldsymbol{W}} F(\boldsymbol{W})$

- One-hot vector representation of class y:

$$\boldsymbol{e}_y = [0, \ldots, 0, \underbrace{1}_{y}, 0, \ldots, 0]^\top \in \{0,1\}^K, \text{ such that } 1^T \boldsymbol{e}_y = 1$$

Computing the Gradient: Step by Step

$$\nabla_{\boldsymbol{W}} L(\boldsymbol{W}; (x, y)) \quad = \quad \nabla_{\boldsymbol{W}} \left(\log \sum_{y'} \exp({\boldsymbol{w}_{y'}}^T \phi(x)) - {\boldsymbol{w}_y}^T \phi(x) \right)$$

Computing the Gradient: Step by Step

$$
\begin{aligned}
\nabla_{\boldsymbol{W}} L(\boldsymbol{W};(x,y)) &= \nabla_{\boldsymbol{W}} \left(\log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x)) - \boldsymbol{w}_y{}^T \phi(x) \right) \\
&= \nabla_{\boldsymbol{W}} \log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x)) - \nabla_{\boldsymbol{W}} \boldsymbol{w}_y{}^T \phi(x)
\end{aligned}
$$

Computing the Gradient: Step by Step

$$
\begin{aligned}
\nabla_{\boldsymbol{W}} L(\boldsymbol{W};(x,y)) &= \nabla_{\boldsymbol{W}} \left(\log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T} \phi(x)) - \boldsymbol{w}_{y}{}^{T} \phi(x) \right) \\
&= \nabla_{\boldsymbol{W}} \log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T} \phi(x)) - \nabla_{\boldsymbol{W}} \boldsymbol{w}_{y}{}^{T} \phi(x) \\
&= \frac{1}{\sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T} \phi(x))} \sum_{y'} \nabla_{\boldsymbol{W}} \exp(\boldsymbol{w}_{y'}{}^{T} \phi(x)) - \boldsymbol{e}_{y} \phi(x)^{\top}
\end{aligned}
$$

Computing the Gradient: Step by Step

$$
\begin{aligned}
\nabla_{\boldsymbol{W}} L(\boldsymbol{W};(x,y)) &= \nabla_{\boldsymbol{W}} \left(\log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T}\phi(x)) - \boldsymbol{w}_{y}{}^{T}\phi(x) \right) \\
&= \nabla_{\boldsymbol{W}} \log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T}\phi(x)) - \nabla_{\boldsymbol{W}} \boldsymbol{w}_{y}{}^{T}\phi(x) \\
&= \frac{1}{\sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T}\phi(x))} \sum_{y'} \nabla_{\boldsymbol{W}} \exp(\boldsymbol{w}_{y'}{}^{T}\phi(x)) - \boldsymbol{e}_{y}\phi(x)^{\top} \\
&= \frac{1}{Z_x} \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T}\phi(x)) \nabla_{\boldsymbol{W}} \boldsymbol{w}_{y'}{}^{T}\phi(x) - \boldsymbol{e}_{y}\phi(x)^{\top}
\end{aligned}
$$

Computing the Gradient: Step by Step

$$
\begin{aligned}
\nabla_{\boldsymbol{W}} L(\boldsymbol{W};(x,y)) &= \nabla_{\boldsymbol{W}} \left(\log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T}\phi(x)) - \boldsymbol{w}_{y}{}^{T}\phi(x) \right) \\
&= \nabla_{\boldsymbol{W}} \log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T}\phi(x)) - \nabla_{\boldsymbol{W}} \boldsymbol{w}_{y}{}^{T}\phi(x) \\
&= \frac{1}{\sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T}\phi(x))} \sum_{y'} \nabla_{\boldsymbol{W}} \exp(\boldsymbol{w}_{y'}{}^{T}\phi(x)) - \boldsymbol{e}_{y}\boldsymbol{\phi}(x)^{\top} \\
&= \frac{1}{Z_x} \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^{T}\phi(x)) \nabla_{\boldsymbol{W}} \boldsymbol{w}_{y'}{}^{T}\phi(x) - \boldsymbol{e}_{y}\boldsymbol{\phi}(x)^{\top} \\
&= \sum_{y'} \frac{\exp(\boldsymbol{w}_{y'}{}^{T}\phi(x))}{Z_x} \boldsymbol{e}_{y'}\boldsymbol{\phi}(x)^{\top} - \boldsymbol{e}_{y}\boldsymbol{\phi}(x)^{\top}
\end{aligned}
$$

Computing the Gradient: Step by Step

$$\begin{aligned}
\nabla_{\boldsymbol{W}} L(\boldsymbol{W};(x,y)) &= \nabla_{\boldsymbol{W}} \left(\log \sum_{y'} \exp({\boldsymbol{w}_{y'}}^T \phi(x)) - {\boldsymbol{w}_y}^T \phi(x) \right) \\
&= \nabla_{\boldsymbol{W}} \log \sum_{y'} \exp({\boldsymbol{w}_{y'}}^T \phi(x)) - \nabla_{\boldsymbol{W}} {\boldsymbol{w}_y}^T \phi(x) \\
&= \frac{1}{\sum_{y'} \exp({\boldsymbol{w}_{y'}}^T \phi(x))} \sum_{y'} \nabla_{\boldsymbol{W}} \exp({\boldsymbol{w}_{y'}}^T \phi(x)) - \boldsymbol{e}_y \phi(x)^\top \\
&= \frac{1}{Z_x} \sum_{y'} \exp({\boldsymbol{w}_{y'}}^T \phi(x)) \nabla_{\boldsymbol{W}} {\boldsymbol{w}_{y'}}^T \phi(x) - \boldsymbol{e}_y \phi(x)^\top \\
&= \sum_{y'} \frac{\exp({\boldsymbol{w}_{y'}}^T \phi(x))}{Z_x} \boldsymbol{e}_{y'} \phi(x)^\top - \boldsymbol{e}_y \phi(x)^\top \\
&= \sum_{y'} P_{\boldsymbol{W}}(y'|x) \boldsymbol{e}_{y'} \phi(x)^\top - \boldsymbol{e}_y \phi(x)^\top
\end{aligned}$$

Computing the Gradient: Step by Step

$$
\begin{aligned}
\nabla_{\boldsymbol{W}} L(\boldsymbol{W};(x,y)) &= \nabla_{\boldsymbol{W}} \left(\log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x)) - \boldsymbol{w}_y{}^T \phi(x) \right) \\
&= \nabla_{\boldsymbol{W}} \log \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x)) - \nabla_{\boldsymbol{W}} \boldsymbol{w}_y{}^T \phi(x) \\
&= \frac{1}{\sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x))} \sum_{y'} \nabla_{\boldsymbol{W}} \exp(\boldsymbol{w}_{y'}{}^T \phi(x)) - \boldsymbol{e}_y \phi(x)^\top \\
&= \frac{1}{Z_x} \sum_{y'} \exp(\boldsymbol{w}_{y'}{}^T \phi(x)) \nabla_{\boldsymbol{W}} \boldsymbol{w}_{y'}{}^T \phi(x) - \boldsymbol{e}_y \phi(x)^\top \\
&= \sum_{y'} \frac{\exp(\boldsymbol{w}_{y'}{}^T \phi(x))}{Z_x} \boldsymbol{e}_{y'} \phi(x)^\top - \boldsymbol{e}_y \phi(x)^\top \\
&= \sum_{y'} P_{\boldsymbol{W}}(y'|x) \boldsymbol{e}_{y'} \phi(x)^\top - \boldsymbol{e}_y \phi(x)^\top \\
&= \left(\begin{bmatrix} \vdots \\ P_{\boldsymbol{W}}(y'|x) \\ \vdots \end{bmatrix} - \boldsymbol{e}_y \right) \phi(x)^\top
\end{aligned}
$$

Logistic Regression Summary

- Conditional class probabilities:

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

Logistic Regression Summary

- Conditional class probabilities:

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

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

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

Logistic Regression Summary

- Conditional class probabilities:

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

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

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

- Gradient can be computed

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

thus (S)GD (or any gradient-based algorithm) can be used.

The Story So Far

- Logistic regression is discriminative: maximizes conditional likelihood
  - ✓ also called log-linear model and max-entropy classifier
  - ✓ no closed form solution.
  - ✓ stochastic gradient updates (SGD):

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

The Story So Far

- Logistic regression is discriminative: maximizes conditional likelihood
  - ✓ also called log-linear model and max-entropy classifier
  - ✓ no closed form solution.
  - ✓ stochastic gradient updates (SGD):

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

- Perceptron is a discriminative, non-probabilistic classifier
  - ✓ perceptron updates:

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

The Story So Far

- Logistic regression is discriminative: maximizes conditional likelihood
  - ✓ also called log-linear model and max-entropy classifier
  - ✓ no closed form solution.
  - ✓ stochastic gradient updates (SGD):

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

- Perceptron is a discriminative, non-probabilistic classifier
  - ✓ perceptron updates:

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

- Logistic regression SGD updates and perceptron updates look similar!

Outline

Maximizing Margin

- Let $\gamma > 0$ denote the margin, and set the goal of maximizing it

$$\max_{\boldsymbol{U}} \ \gamma$$

subject to

$$\|\boldsymbol{U}\| = 1$$

$$\boldsymbol{u}_{y_t}^T \phi(x_t) - \boldsymbol{u}_{y'}^T \phi(x_t) \geq \gamma$$

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

Maximizing Margin

- Let $\gamma > 0$ denote the margin, and set the goal of maximizing it

$$\max_{\boldsymbol{U}} \ \gamma$$

subject to

$$\|\boldsymbol{U}\| = 1$$

$$\boldsymbol{u}_{y_t}^T \phi(x_t) - \boldsymbol{u}_{y'}^T \phi(x_t) \geq \gamma$$

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

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

Maximizing Margin

- Let $\gamma > 0$ denote the margin, and set the goal of maximizing it

$$\max_{\boldsymbol{U}} \ \gamma$$

subject to

$$\|\boldsymbol{U}\| = 1$$

$$\boldsymbol{u}_{y_t}^T \phi(x_t) - \boldsymbol{u}_{y'}^T \phi(x_t) \geq \gamma$$

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

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

- Fix $||\boldsymbol{U}|| = 1$ since increasing $\|\boldsymbol{U}\|$ trivially produces larger margin

Maximum Margin $\Leftrightarrow$ Minimum Norm

Max Margin:

$$\max_{\boldsymbol{U}} \quad \gamma$$

subject to

$$\|\boldsymbol{U}\| = 1$$

$$\boldsymbol{u}_{y_t}^T \phi(x_t) - \boldsymbol{u}_{y'}^T \phi(x_t) \geq \gamma$$

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

$\Leftrightarrow$

Min Norm:

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

such that:

$$\boldsymbol{w}_{y_t}^T \phi(x_t) - \boldsymbol{w}_{y'}^T \phi(x_t) \geq 1$$

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

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

Maximum Margin $\Leftrightarrow$ Minimum Norm

Max Margin:

$$\max_{\boldsymbol{U}} \quad \gamma$$

subject to

$$\|\boldsymbol{U}\| = 1$$

$$\boldsymbol{u}_{y_t}^T \phi(x_t) - \boldsymbol{u}_{y'}^T \phi(x_t) \geq \gamma$$

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

$\Leftrightarrow$

Min Norm:

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

such that:

$$\boldsymbol{w}_{y_t}^T \phi(x_t) - \boldsymbol{w}_{y'}^T \phi(x_t) \geq 1$$

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

- Instead of fixing $||\boldsymbol{U}||$ we fix the margin to 1
- Make substitution $\boldsymbol{W} = \frac{\boldsymbol{U}}{\gamma}$; then we have $\|\boldsymbol{W}\| = \frac{\|\boldsymbol{U}\|}{\gamma} = \frac{1}{\gamma}$.
- Quadratic programming (QP) problem: well known convex problem, for which there are several techniques.

Support Vector Machines

- What if data is not separable?

Support Vector Machines

- What if data is not separable? Introduce and penalize slacks

Support Vector Machines

- What if data is not separable? Introduce and penalize slacks
- Slacks allow (penalized) violation of the margin constraints

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

subject to

$$\boldsymbol{w}_{y_t}^T\phi(x_t) - \boldsymbol{w}_{y'}^T\phi(x_t) \geq 1 - \xi_t \text{ and } \xi_t \geq 0$$

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

Support Vector Machines

- What if data is not separable? Introduce and penalize slacks

- Slacks allow (penalized) violation of the margin constraints

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

subject to

$$\boldsymbol{w}_{y_t}^T\phi(x_t) - \boldsymbol{w}_{y'}^T\phi(x_t) \geq 1 - \xi_t \text{ and } \xi_t \geq 0$$

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

- Larger C: more examples correctly classified, but smaller margin.

Support Vector Machines

- What if data is not separable? Introduce and penalize slacks

- Slacks allow (penalized) violation of the margin constraints

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

subject to

$$\boldsymbol{w}_{y_t}^T\phi(x_t) - \boldsymbol{w}_{y'}^T\phi(x_t) \geq 1 - \xi_t \text{ and } \xi_t \geq 0$$

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

- Larger C: more examples correctly classified, but smaller margin.

- If data is separable, optimal solution has $\xi_i = 0$, $\forall i$

Support Vector Machines: Hinge Loss View

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

subject to

$$\boldsymbol{w}_{y_t}^T\phi(x_t) - \boldsymbol{w}_{y'}^T\phi(x_t) \geq 1 - \xi_t \quad \forall y' \neq y_t$$

Support Vector Machines: Hinge Loss View

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

subject to

$$\boldsymbol{w}_{y_t}^T \phi(x_t) - \max_{y' \neq y_t} \ \boldsymbol{w}_{y'}^T \phi(x_t) \geq 1 - \xi_t$$

Support Vector Machines: Hinge Loss View

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

subject to

$$\xi_t \geq 1 + \max_{y' \neq y_t} \; \boldsymbol{w}_{y'}^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t)$$

Support Vector Machines: Hinge Loss View

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

subject to

$$\xi_t \geq 1 + \max_{y' \neq y_t} \; \boldsymbol{w}_{y'}^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t)$$

Support Vector Machines: Hinge Loss View

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

subject to

$$\xi_t \geq 1 + \max_{y' \neq y_t} \; \boldsymbol{w}_{y'}^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t)$$

- If $\boldsymbol{W}$ classifies (x_t, y_t) with margin 1, penalty $\xi_t = 0$

Support Vector Machines: Hinge Loss View

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

subject to

$$\xi_t \geq 1 + \max_{y' \neq y_t} \; \boldsymbol{w}_{y'}^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t)$$

- If $\boldsymbol{W}$ classifies (x_t, y_t) with margin 1, penalty $\xi_t = 0$
- Otherwise penalty/slack $\xi_t = 1 + \max_{y' \neq y_t} \; \boldsymbol{w}_{y'}^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t)$

Support Vector Machines: Hinge Loss View

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

subject to

$$\xi_t \geq 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'}^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t)$$

- If $\boldsymbol{W}$ classifies (x_t, y_t) with margin 1, penalty $\xi_t = 0$
- Otherwise penalty/slack $\xi_t = 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'}^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t)$
- Hinge loss:

$$L(\boldsymbol{W}; (x_t, y_t)) = \max \left(0, 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'}^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t)\right)$$

Support Vector Machines: Hinge Loss View

- SVM QP formulation:

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

subject to

$$\xi_t \geq 1 + \max_{y' \neq y_t} \ \boldsymbol{w}_{y'}^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t), \quad \text{for } t = 1, \ldots, N$$

Support Vector Machines: Hinge Loss View

- SVM QP formulation:

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

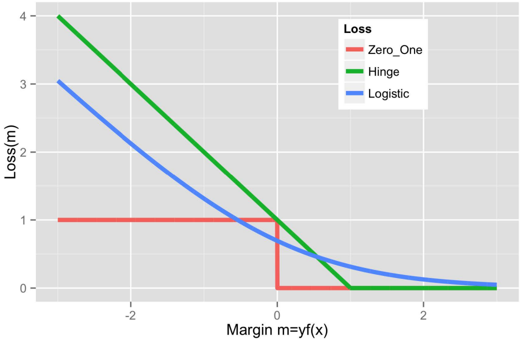
subject to

$$\xi_t \geq 1 + \max_{y' \neq y_t} \boldsymbol{w}_{y'}^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t), \quad \text{for } t = 1, \ldots, N$$

- Hinge loss equivalent:

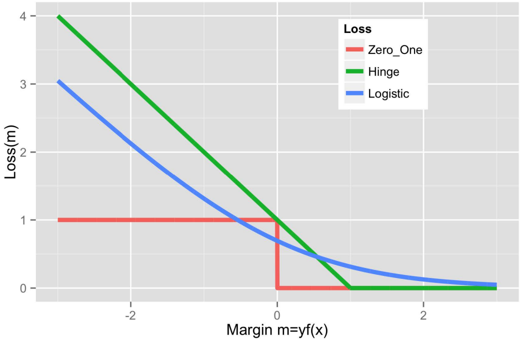
$$\boldsymbol{W} = \arg\min_{\boldsymbol{W}} \left(\sum_{t=1}^{N} \underbrace{\max\left(0, 1 - \overbrace{\left(\boldsymbol{w}_{y_t}^T \phi(x_t) - \max_{y' \neq y_t} \boldsymbol{w}_{y'}^T \phi(x_t)\right)}^{\text{margin of sample } t}\right)}_{L(\boldsymbol{W};(x_t,y_t))} \right) + \frac{\lambda}{2}||\boldsymbol{W}||^2$$

Hinge Loss



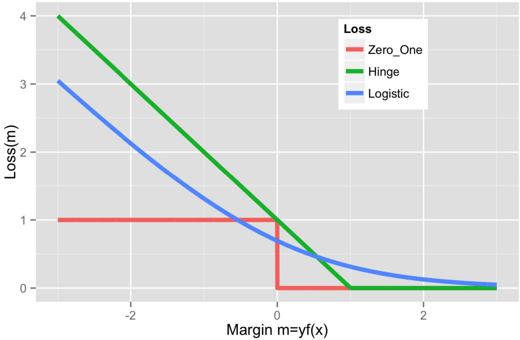
- Hinge: $h(u) = \max\{0, 1-u\}$: piecewise linear, not everywhere differentiable.

Hinge Loss



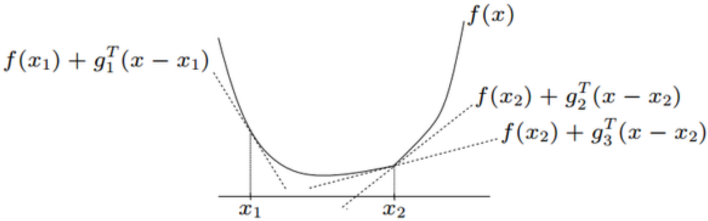
- Hinge: $h(u) = \max\{0, 1 - u\}$: piecewise linear, not everywhere differentiable.
- Cannot use gradient descent

Hinge Loss



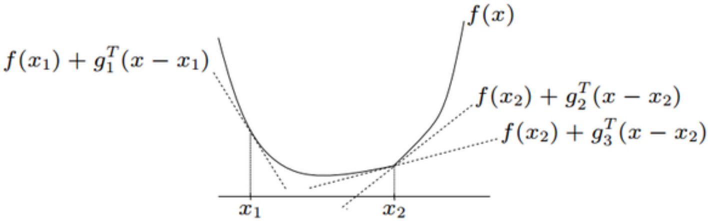
- Hinge: $h(u) = \max\{0, 1 - u\}$: piecewise linear, not everywhere differentiable.
- Cannot use gradient descent
- But can use subgradient descent (almost the same)!

Subgradients



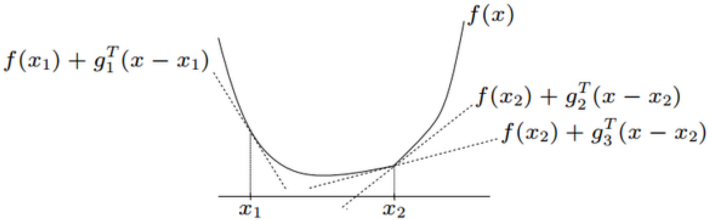
- Defined for convex functions $f : \mathbb{R}^D \to \mathbb{R}$

Subgradients



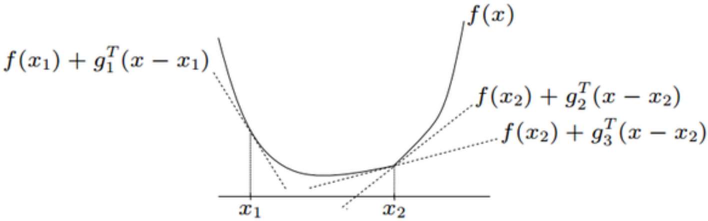
- Defined for convex functions $f : \mathbb{R}^D \to \mathbb{R}$
- Generalizes the notion of gradient: in points where f is differentiable, there is a single subgradient which equals the gradient.

Subgradients



- 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.
- At points where f is non-differentiable, there are infinitely many subgradients (an interval for $D = 1$).

Subgradients



- 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.
- At points where f is non-differentiable, there are infinitely many subgradients (an interval for $D = 1$).
- For $D = 1$ (figure above), a subgradient at x_2 is the slope of any tangent that stays below the function.

Subgradients: Hinge Function

- Hinge: $h(u) = \max\{0, 1 - u\}$

Subgradients: Hinge Function

- Hinge: $h(u) = \max\{0, 1 - u\}$
- Subgradients:
  - ✓ For $u < 1$, $\tilde{\nabla}_u h(u) = -1$
  - ✓ For $u > 1$, $\tilde{\nabla}_u h(u) = 0$
  - ✓ For $u = 1$, $\tilde{\nabla}_u h(u) =$ any number in $[-1, 0]$.

Subgradients: Hinge Function

- Hinge: $h(u) = \max\{0, 1 - u\}$
- Subgradients:
  - ✓ For $u < 1$, $\tilde{\nabla}_u h(u) = -1$
  - ✓ For $u > 1$, $\tilde{\nabla}_u h(u) = 0$
  - ✓ For $u = 1$, $\tilde{\nabla}_u h(u) =$ any number in $[-1, 0]$.
- Can take a subgradient at $u = 1$ to be 0

Subgradients: Hinge Function

- Hinge: $h(u) = \max\{0, 1 - u\}$
- Subgradients:
  - ✓ For $u < 1$, $\tilde{\nabla}_u h(u) = -1$
  - ✓ For $u > 1$, $\tilde{\nabla}_u h(u) = 0$
  - ✓ For $u = 1$, $\tilde{\nabla}_u h(u) =$ any number in $[-1, 0]$.
- Can take a subgradient at $u = 1$ to be 0
- For some $f(x) = h(g(x))$, if g is differentiable, a valid choice is thus

$$\tilde{\nabla} f(x) = \begin{cases} 0, & \text{if } g(x) \geq 1 \\ -\nabla g(x), & \text{if } g(x) < 1 \end{cases}$$

Perceptron and Hinge-Loss

- SVM subgradient update (ignoring $\|\boldsymbol{W}\|^2$ term):

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

Perceptron and Hinge-Loss

- SVM subgradient update (ignoring $\|\boldsymbol{W}\|^2$ term):

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

- Perceptron update is similar (but not equal):

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

where $\eta = 1$

Perceptron and Hinge-Loss

- SVM subgradient update (ignoring $\|\boldsymbol{W}\|^2$ term):

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

- Perceptron update is similar (but not equal):

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

where $\eta = 1$

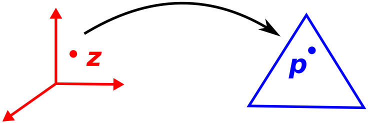
- Perceptron = SGD with zero-margin hinge-loss:

$$\max\left(0, \max_{y \neq y_t} \boldsymbol{w}_y^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t)\right) = \mathrm{ReLU}(\max_{y \neq y_t} \boldsymbol{w}_y^T \phi(x_t) - \boldsymbol{w}_{y_t}^T \phi(x_t))$$

Outline

Obtaining Probabilities

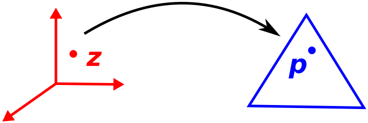
- Mapping from score vector $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{Y}|}$ to probability distribution over $\mathcal{Y}$



$$\underbrace{\Delta_{K-1} = \{\mathsf{v} \in \mathbb{R}_+^K : \sum_i v_i = 1\}}_{\text{probability simplex}}$$

Obtaining Probabilities

- Mapping from score vector $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{Y}|}$ to probability distribution over $\mathcal{Y}$

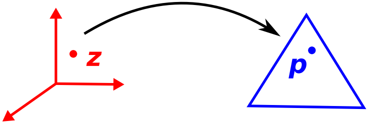


$$\Delta_{K-1} = \underbrace{\{\mathsf{v} \in \mathbb{R}_+^K : \sum_i v_i = 1\}}_{\text{probability simplex}}$$

- Any such mapping $\boldsymbol{\rho} : \mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$ should satisfy:
  - ✓ for any $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{Y}|}$ and $\alpha \in \mathbb{R}$, $\boldsymbol{\rho}(\boldsymbol{z} + \alpha) = \boldsymbol{\rho}(\boldsymbol{z})$

Obtaining Probabilities

- Mapping from score vector $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{Y}|}$ to probability distribution over $\mathcal{Y}$

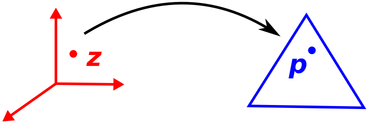


$$\Delta_{K-1} = \underbrace{\{\mathsf{v} \in \mathbb{R}_+^K : \sum_i v_i = 1\}}_{\text{probability simplex}}$$

- Any such mapping $\boldsymbol{\rho} : \mathbb{R}^{|\mathcal{Y}|} \rightarrow \Delta_{|\mathcal{Y}|-1}$ should satisfy:
  - ✓ for any $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{Y}|}$ and $\alpha \in \mathbb{R}$, $\boldsymbol{\rho}(\boldsymbol{z} + \alpha) = \boldsymbol{\rho}(\boldsymbol{z})$
  - ✓ permutation equivariance: P, $\boldsymbol{\rho}(\mathsf{P}\boldsymbol{z}) = \mathsf{P}\boldsymbol{\rho}(\boldsymbol{z})$, $\forall$ permutation matrix P

Obtaining Probabilities

- Mapping from score vector $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{Y}|}$ to probability distribution over $\mathcal{Y}$

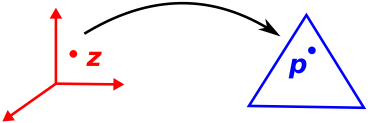


$$\Delta_{K-1} = \underbrace{\{\mathsf{v} \in \mathbb{R}^K_+ : \sum_i v_i = 1\}}_{\text{probability simplex}}$$

- Any such mapping $\boldsymbol{\rho} : \mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$ should satisfy:
  - ✓ for any $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{Y}|}$ and $\alpha \in \mathbb{R}$, $\boldsymbol{\rho}(\boldsymbol{z} + \alpha) = \boldsymbol{\rho}(\boldsymbol{z})$
  - ✓ permutation equivariance: P, $\boldsymbol{\rho}(\mathsf{P}\boldsymbol{z}) = \mathsf{P}\boldsymbol{\rho}(\boldsymbol{z})$, $\forall$ permutation matrix P
  - ✓ monotonicity: $z_i \geq z_j \;\Rightarrow\; \big(\boldsymbol{\rho}(\boldsymbol{z})\big)_i \geq \big(\boldsymbol{\rho}(\boldsymbol{z})\big)_j$

Obtaining Probabilities

- Mapping from score vector $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{Y}|}$ to probability distribution over $\mathcal{Y}$



$$\Delta_{K-1} = \underbrace{\{\mathsf{v} \in \mathbb{R}^K_+ : \sum_i v_i = 1\}}_{\text{probability simplex}}$$

- Any such mapping $\boldsymbol{\rho} : \mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$ should satisfy:
  - ✓ for any $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{Y}|}$ and $\alpha \in \mathbb{R}$, $\boldsymbol{\rho}(\boldsymbol{z} + \alpha) = \boldsymbol{\rho}(\boldsymbol{z})$
  - ✓ permutation equivariance: P, $\boldsymbol{\rho}(\mathsf{P}\boldsymbol{z}) = \mathsf{P}\boldsymbol{\rho}(\boldsymbol{z})$, $\forall$ permutation matrix P
  - ✓ monotonicity: $z_i \geq z_j \;\Rightarrow\; \big(\boldsymbol{\rho}(\boldsymbol{z})\big)_i \geq \big(\boldsymbol{\rho}(\boldsymbol{z})\big)_j$
- We already saw one such mapping: softmax. Next: sparsemax.

Recap: Softmax Transformation

- Classical choice is softmax : $\mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$:

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

Recap: Softmax Transformation

- Classical choice is softmax : $\mathbb{R}^{|\mathcal{Y}|} \rightarrow \Delta_{|\mathcal{Y}|-1}$:

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

- Underlies logistic regression!

Recap: Softmax Transformation

- Classical choice is softmax : $\mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$:

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

- Underlies logistic regression!

- Resulte has full support: $\big(\text{softmax}(\boldsymbol{z})\big)_i > 0, \forall \boldsymbol{z}, i \in \{1, ..., |\mathcal{Y}|\}$

Recap: Softmax Transformation

- Classical choice is softmax : $\mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$:

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

- Underlies logistic regression!

- Resulte has full support: $\big(\text{softmax}(\boldsymbol{z})\big)_i > 0, \forall \boldsymbol{z}, i \in \{1, ..., |\mathcal{Y}|\}$

- A disadvantage if a sparse distribution is desired (keeping only the most probable classes, in an adaptive way).

Recap: Softmax Transformation

- Classical choice is softmax : $\mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$:

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

- Underlies logistic regression!
- Resulte has full support: $\big(\mathrm{softmax}(\boldsymbol{z})\big)_i > 0, \forall \boldsymbol{z}, i \in \{1, ..., |\mathcal{Y}|\}$
- A disadvantage if a sparse distribution is desired (keeping only the most probable classes, in an adaptive way).
- Common workaround: threshold and renormalize.

Sparsemax (Martins and Astudillo, 2016)

- A sparse-friendly alternative is sparsemax : $\mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$.

Sparsemax (Martins and Astudillo, 2016)

- A sparse-friendly alternative is $\text{sparsemax} : \mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$.
- Key idea: Euclidean projection of $\boldsymbol{z}$ onto the probability simplex

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

Sparsemax (Martins and Astudillo, 2016)

- A sparse-friendly alternative is sparsemax : $\mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$.
- Key idea: Euclidean projection of $\boldsymbol{z}$ onto the probability simplex

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

- May be at the boundary of the simplex, in which case sparsemax($\boldsymbol{z}$) is sparse (has zeros)

Sparsemax (Martins and Astudillo, 2016)

- A sparse-friendly alternative is sparsemax : $\mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$.
- Key idea: Euclidean projection of $\boldsymbol{z}$ onto the probability simplex

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

- May be at the boundary of the simplex, in which case sparsemax($\boldsymbol{z}$) is sparse (has zeros)
- Retains many properties of softmax, namely differentiability

Sparsemax (Martins and Astudillo, 2016)

- A sparse-friendly alternative is $\text{sparsemax} : \mathbb{R}^{|\mathcal{Y}|} \rightarrow \Delta_{|\mathcal{Y}|-1}$.
- Key idea: Euclidean projection of $\boldsymbol{z}$ onto the probability simplex

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

- May be at the boundary of the simplex, in which case $\text{sparsemax}(\boldsymbol{z})$ is sparse (has zeros)
- Retains many properties of softmax, namely differentiability
- Can be computed efficiently, with cost at most $O(|\mathcal{Y}| \log |\mathcal{Y}|)$

Sparsemax (Martins and Astudillo, 2016)

- A sparse-friendly alternative is sparsemax : $\mathbb{R}^{|\mathcal{Y}|} \rightarrow \Delta_{|\mathcal{Y}|-1}$.
- Key idea: Euclidean projection of $\boldsymbol{z}$ onto the probability simplex

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

- May be at the boundary of the simplex, in which case sparsemax($\boldsymbol{z}$) is sparse (has zeros)
- Retains many properties of softmax, namely differentiability
- Can be computed efficiently, with cost at most $O(|\mathcal{Y}| \log |\mathcal{Y}|)$
- Essentially: sorting, shifting, and thresholding.

The Binary Case

- $\mathcal{Y} = \{1, 2\}$; parametrize $z = (t, 0)$

The Binary Case

- $\mathcal{Y} = \{1, 2\}$; parametrize $\boldsymbol{z} = (t, 0)$
- The binary softmax is the logistic (sigmoid) function:

$$\mathrm{softmax}_1(\boldsymbol{z}) = \frac{1}{1 + \exp(-t)}$$

The Binary Case

- $\mathcal{Y} = \{1, 2\}$; parametrize $\boldsymbol{z} = (t, 0)$
- The binary softmax is the logistic (sigmoid) function:

$$\mathsf{softmax}_1(\boldsymbol{z}) = \frac{1}{1 + \exp(-t)}$$

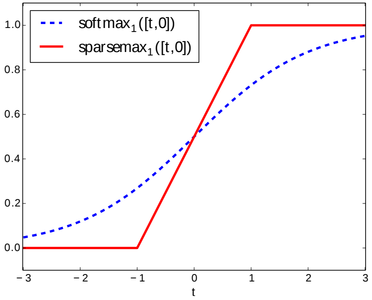
- The binary sparsemax is a "hardened" version of the sigmoid:

The Binary Case

- $\mathcal{Y} = \{1, 2\}$; parametrize $\boldsymbol{z} = (t, 0)$
- The binary softmax is the logistic (sigmoid) function:

$$\text{softmax}_1(\boldsymbol{z}) = \frac{1}{1 + \exp(-t)}$$

- The binary sparsemax is a "hardened" version of the sigmoid:

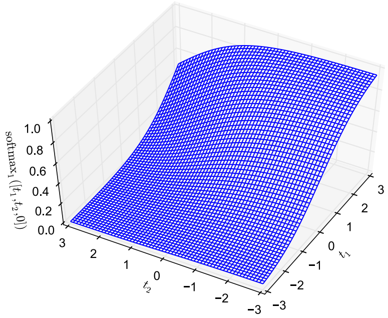


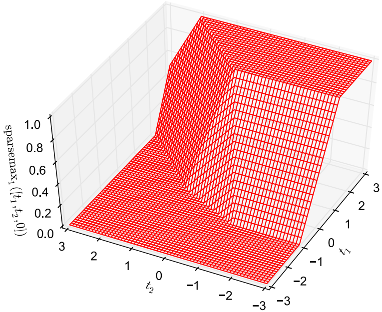
Ternary Case

- Parameterize $z = (t_1, t_2, 0)$ and plot $\mathrm{softmax}_1(z)$ and $\mathrm{sparsemax}_1(z)$ as a function of t_1 and t_2

Ternary Case

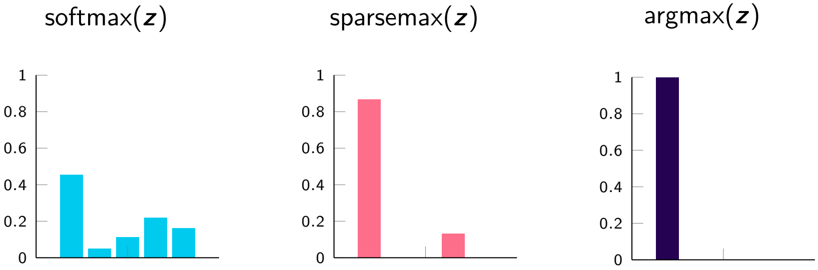
- Parameterize $z = (t_1, t_2, 0)$ and plot $\text{softmax}_1(z)$ and $\text{sparsemax}_1(z)$ as a function of t_1 and t_2
- sparsemax is piecewise linear, but asymptotically similar to softmax





Softmax, sparsemax, and argmax

- Sparsemax is in-between softmax and argmax

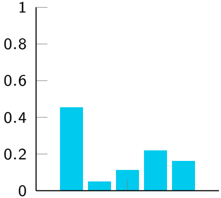


(Same $\boldsymbol{z} = [1.0716, -1.1221, -0.3288, 0.3368, 0.0425]$)

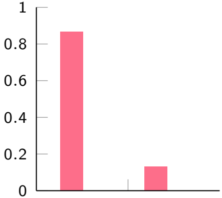
Softmax, sparsemax, and argmax

- Sparsemax is in-between softmax and argmax

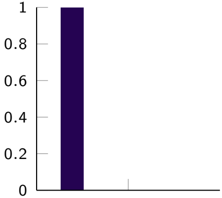
softmax($\boldsymbol{z}$)



sparsemax($\boldsymbol{z}$)



argmax($\boldsymbol{z}$)



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

- It is (it may be) **sparse**, but **differentiable**.

Temperature

- We may include a "temperature" parameter T in softmax and sparsemax:
- Scale the argument by $1/T$: $\text{softmax}(z/T)$ and $\text{sparsemax}(z/T)$

Temperature

- We may include a "temperature" parameter T in softmax and sparsemax:
- Scale the argument by $1/T$: $\text{softmax}(z/T)$ and $\text{sparsemax}(z/T)$
- Zero temperature limit:

$$\lim_{T \to 0} \text{softmax}(z/T) = \lim_{T \to 0} \text{sparsemax}(z/T) = \text{argmax}(z)$$

Temperature

- We may include a "temperature" parameter T in softmax and sparsemax:
- Scale the argument by $1/T$: $\text{softmax}(z/T)$ and $\text{sparsemax}(z/T)$
- Zero temperature limit:

$$\lim_{T \to 0} \text{softmax}(z/T) = \lim_{T \to 0} \text{sparsemax}(z/T) = \text{argmax}(z)$$

- High temperature limit:

$$\lim_{T \to \infty} \text{softmax}(z/T) = \lim_{T \to 0} \text{sparsemax}(z/T) = \left(\frac{1}{|\mathcal{Y}|}, \ldots, \frac{1}{|\mathcal{Y}|}\right)$$

Temperature

- We may include a "temperature" parameter T in softmax and sparsemax:
- Scale the argument by $1/T$: $\text{softmax}(\boldsymbol{z}/T)$ and $\text{sparsemax}(\boldsymbol{z}/T)$
- Zero temperature limit:

$$\lim_{T \to 0} \text{softmax}(\boldsymbol{z}/T) = \lim_{T \to 0} \text{sparsemax}(\boldsymbol{z}/T) = \text{argmax}(\boldsymbol{z})$$

- High temperature limit:

$$\lim_{T \to \infty} \text{softmax}(\boldsymbol{z}/T) = \lim_{T \to 0} \text{sparsemax}(\boldsymbol{z}/T) = \left(\frac{1}{|\mathcal{Y}|}, \ldots, \frac{1}{|\mathcal{Y}|}\right)$$

- The temperature controls how peaked the softmax is and how sparse the sparsemax is.

Loss Function for Sparsemax?

- The common choice for softmax:
  - ✓ the classifier estimates $P(y = c \mid x; \boldsymbol{W})$

Loss Function for Sparsemax?

- The common choice for softmax:
  - ✓ the classifier estimates $P(y = c \mid x; \boldsymbol{W})$
  - ✓ loss is the negative log-likelihood:

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

  where $z_c(x)$ is the score of class c.

Loss Function for Sparsemax?

- The common choice for softmax:
  - ✓ the classifier estimates $P(y = c \mid x; \boldsymbol{W})$
  - ✓ loss is the negative log-likelihood:

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

  where $z_c(x)$ is the score of class c.

- Loss gradient:

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

Loss Function for Sparsemax?

- The common choice for softmax:
  - ✓ the classifier estimates $P(y = c \mid x; \boldsymbol{W})$
  - ✓ loss is the negative log-likelihood:

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

  where $z_c(x)$ is the score of class c.

- Loss gradient:

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

- Not directly applicable to sparsemax: cannot compute $log(0)$

Sparsemax Loss(Martins and Astudillo, 2016)

- The natural choice for a sparsemax output layer

Sparsemax Loss(Martins and Astudillo, 2016)

- The natural choice for a sparsemax output layer
- Compute estimates $P(y \mid x; \boldsymbol{W})$ using sparsemax

Sparsemax Loss(Martins and Astudillo, 2016)

- The natural choice for a sparsemax output layer
- Compute estimates $P(y \mid x; \boldsymbol{W})$ using sparsemax
- We would like the gradient to have the form:

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

Sparsemax Loss(Martins and Astudillo, 2016)

- The natural choice for a sparsemax output layer
- Compute estimates $P(y \mid x; \boldsymbol{W})$ using sparsemax
- We would like the gradient to have the form:

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

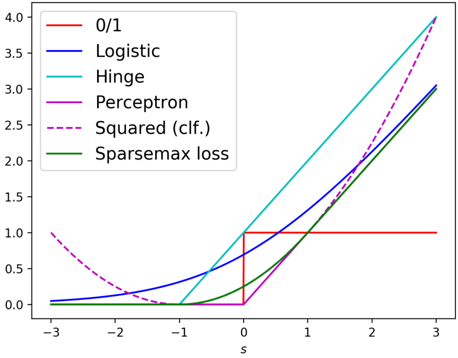
- This is achieved with the sparsemax loss:

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

where $z_y(x)$ is the score of class y.

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":

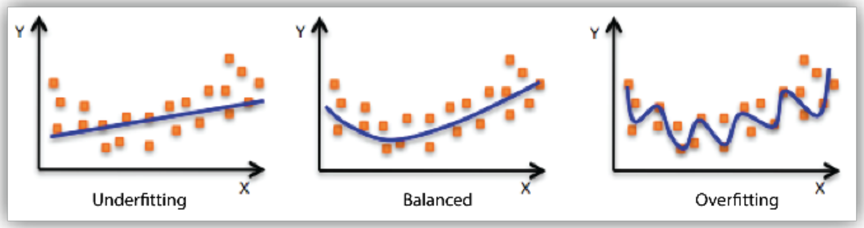


Outline

1. Regression
2. Classification
   - Perceptron
   - Logistic Regression
   - Support Vector Machines
   - Sparsemax
3. **Regularization**
4. Non-Linear Models

Overfitting

- If a model is too complex (too many parameters), there is a the risk of overfitting:



- We saw one example already with polynomial regression.

Regularization

- Regularization aims at preventing overfitting

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

$\Omega(\boldsymbol{W})$: regularization function; λ: regularization parameter.

Regularization

- Regularization aims at preventing overfitting

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

$\Omega(\boldsymbol{W})$: regularization function; λ: regularization parameter.

- ℓ_2 regularization (or Gaussian prior) promotes small weights:

$$\Omega(\boldsymbol{W}) = \frac{1}{2}\|\boldsymbol{W}\|_2^2 = \frac{1}{2}\sum_y \|\boldsymbol{w}_y\|_2^2 = \frac{1}{2}\sum_y \sum_j w_{y,j}^2$$

Regularization

- Regularization aims at preventing overfitting

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

$\Omega(\boldsymbol{W})$: regularization function; λ: regularization parameter.

- ℓ_2 regularization (or Gaussian prior) promotes small weights:

$$\Omega(\boldsymbol{W}) = \frac{1}{2}\|\boldsymbol{W}\|_2^2 = \frac{1}{2}\sum_y \|\boldsymbol{w}_y\|_2^2 = \frac{1}{2}\sum_y \sum_j w_{y,j}^2$$

- ℓ_1 regularization (Laplacian prior) promotes sparse weights!

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

Regularization

- Regularization aims at preventing overfitting

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

$\Omega(\boldsymbol{W})$: regularization function; λ: regularization parameter.

- ℓ_2 regularization (or Gaussian prior) promotes small weights:

$$\Omega(\boldsymbol{W}) = \frac{1}{2}\|\boldsymbol{W}\|_2^2 = \frac{1}{2}\sum_y \|\boldsymbol{w}_y\|_2^2 = \frac{1}{2}\sum_y \sum_j w_{y,j}^2$$

- ℓ_1 regularization (Laplacian prior) promotes sparse weights!

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

- Easy to use ℓ_2 in gradient methods, since $\nabla_{\boldsymbol{W}} \frac{1}{2}\|\boldsymbol{W}\|_2^2 = \boldsymbol{W}$.

Regularization

- Regularization aims at preventing overfitting

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

$\Omega(\boldsymbol{W})$: regularization function; λ: regularization parameter.

- ℓ_2 regularization (or Gaussian prior) promotes small weights:

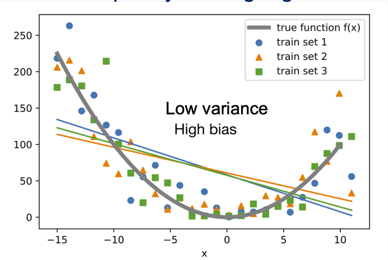
$$\Omega(\boldsymbol{W}) = \frac{1}{2}\|\boldsymbol{W}\|_2^2 = \frac{1}{2}\sum_y \|\boldsymbol{w}_y\|_2^2 = \frac{1}{2}\sum_y \sum_j w_{y,j}^2$$

- ℓ_1 regularization (Laplacian prior) promotes sparse weights!

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

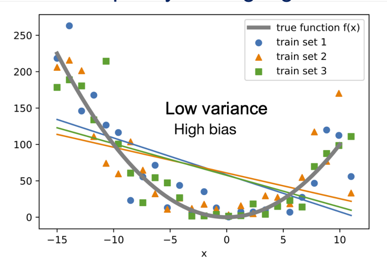
- Easy to use ℓ_2 in gradient methods, since $\nabla_{\boldsymbol{W}} \frac{1}{2}\|\boldsymbol{W}\|_2^2 = \boldsymbol{W}$.
- Not so easy to use ℓ_1 regularization.

low complexity / strong regularization

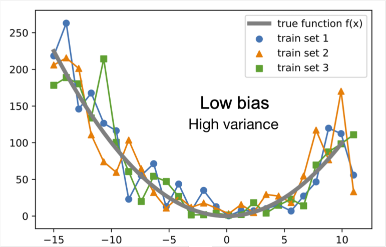


Bias, Variance, and their Tradeoff

low complexity / strong regularization

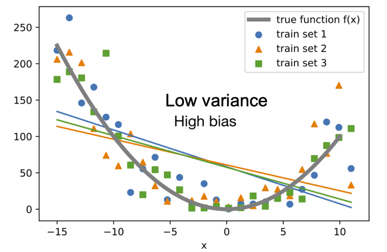


high complexity / weak regularization

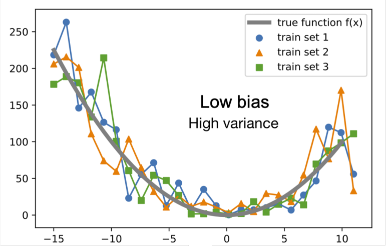


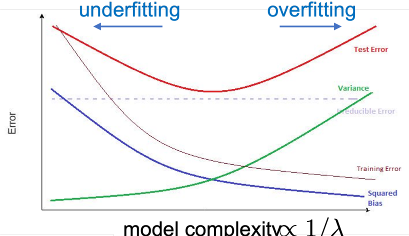
Bias, Variance, and their Tradeoff

low complexity / strong regularization



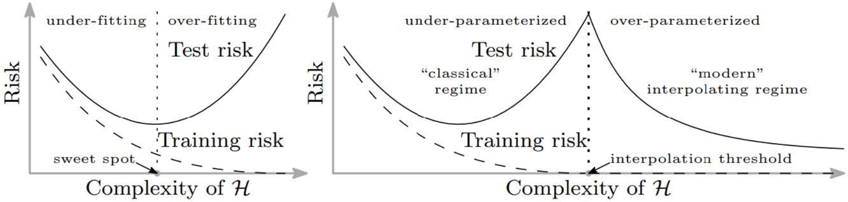
high complexity / weak regularization





model complexity $\propto 1/\lambda$

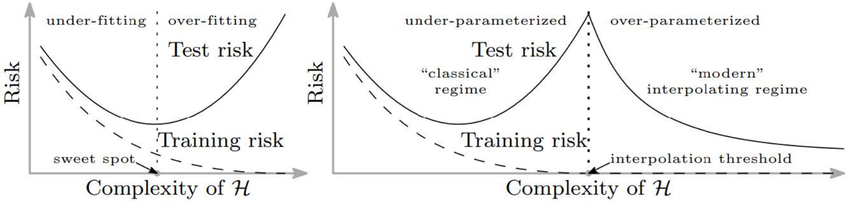
- A more modern view, compatible with large deep networks:



(a) U-shaped "bias-variance" risk curve (b) "double descent" risk curve

Double Descent

- A more modern view, compatible with large deep networks:



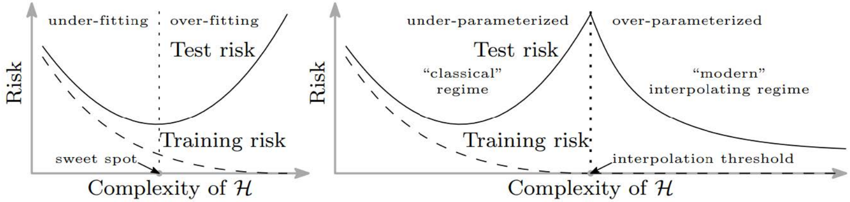
(a) U-shaped "bias-variance" risk curve

(b) "double descent" risk curve

- In the interpolating regime, use minimum-norm criterion:

$$\widehat{\boldsymbol{W}} = \arg\min_{\boldsymbol{W}} \|\boldsymbol{W}\|^2, \quad \text{subject to} \quad \overbrace{\sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t)) = 0}^{\textit{interpolation}}$$

- A more modern view, compatible with large deep networks:



(a) U-shaped "bias-variance" risk curve

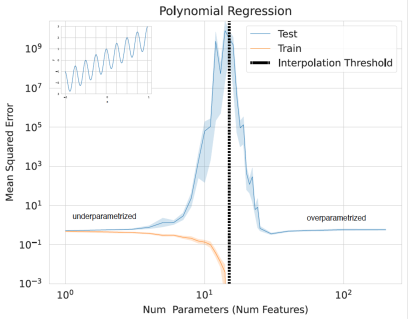
(b) "double descent" risk curve

- In the interpolating regime, use minimum-norm criterion:

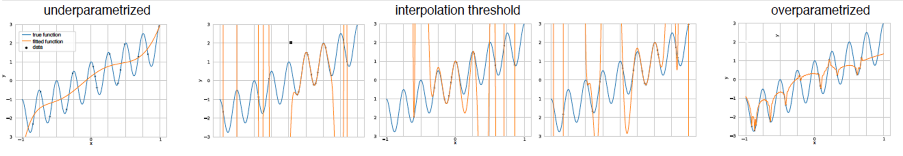
$$\widehat{\boldsymbol{W}} = \arg\min_{\boldsymbol{W}} \|\boldsymbol{W}\|^2, \quad \text{subject to} \quad \overbrace{\sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t)) = 0}^{\text{interpolation}}$$

- Active research topic, pioneered by M. Belkin (2018).

Double Descent: Intuition



Schaeffer et al, 2023
arXiv:2303.14151v1



Outline

Summary: Linear Classifiers

- We have covered:
  - ✓ Perceptron
  - ✓ Logistic and Sparsemax regression
  - ✓ Support vector machines
- All lead to convex optimization problems $\Rightarrow$ no issues with local minima/initialization
- All assume the feature map ϕ is 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!)
- Use kernel methods:
  - ✓ work implicitly in high-dimensional feature spaces
  - ✓ ... but still need to choose/design a good kernel

What If Data Are Not Linearly Separable?

- Engineer better features (often works!)
- Use kernel methods:
  - ✓ work implicitly in high-dimensional feature spaces
  - ✓ ... but still need to choose/design a good kernel
- Use one of many other methods: trees, random forests, nearest neighbors, ...

What If Data Are Not Linearly Separable?

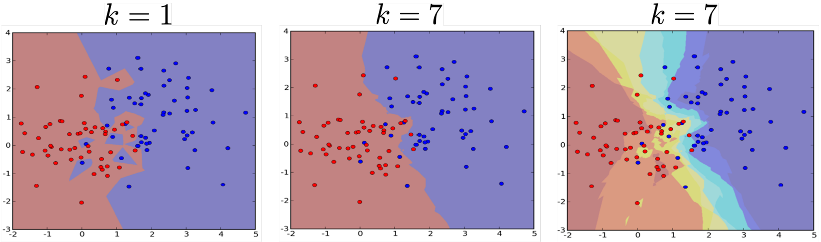
- Engineer better features (often works!)
- Use kernel methods:
  - ✓ work implicitly in high-dimensional feature spaces
  - ✓ ... but still need to choose/design a good kernel
- Use one of many other methods: trees, random forests, nearest neighbors, ...
- Use deep neural networks (tomorrow's lecture!)
  - ✓ embrace non-convexity and local minima
  - ✓ instead of engineering features/kernels, engineer the model architecture,
  - ✓ ...and use many tricks of the trade.

Nearest Neighbor Classifiers

- Instead of "training", **keep** all the data $\mathcal{D} = \{(x_i, y_i)_{i=1}^N\}$
- For a test sample x, return the majority class in the k nearest neighbors in $\{x_1, ..., x_N\}$

Nearest Neighbor Classifiers

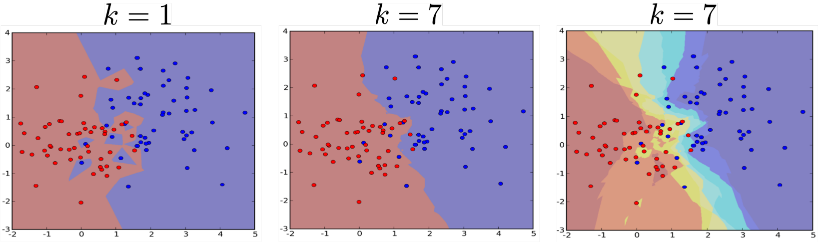
- Instead of "training", **keep** all the data $\mathcal{D} = \{(x_i, y_i)_{i=1}^N\}$
- For a test sample x, return the majority class in the k nearest neighbors in $\{x_1, ..., x_N\}$



class probability estimates

Nearest Neighbor Classifiers

- Instead of "training", **keep** all the data $\mathcal{D} = \{(x_i, y_i)_{i=1}^N\}$
- For a test sample x, return the majority class in the k nearest neighbors in $\{x_1, ..., x_N\}$

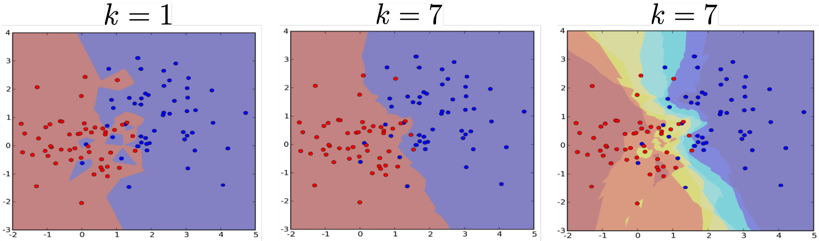


class probability estimates

- Pros: no training, easy implementation, few assumptions, intuitive, intrinsically explainable

Nearest Neighbor Classifiers

- Instead of "training", **keep** all the data $\mathcal{D} = \{(x_i, y_i)_{i=1}^N\}$
- For a test sample x, return the majority class in the k nearest neighbors in $\{x_1, ..., x_N\}$



class probability estimates

- Pros: no training, easy implementation, few assumptions, intuitive, intrinsically explainable
- Cons: store all the data, need to define distance, not top (but decent) performance, slow with large high-dim datasets (but there are tricks!)

Nearest Neighbor Classifiers: Obsolete?

"Low-Resource" Text Classification: A Parameter-Free Classification Method with Compressors

Zhiying Jiang[1,2], Matthew Y.R. Yang[1], Mikhail Tsirlin[1], Raphael Tang[1], Yiqin Dai[2] and Jimmy Lin[1]

ACL 2023, July 9-14

alternative to DNNs that's easy, lightweight, and universal in text classification: a combination of a simple compressor like *gzip* with a k-nearest-neighbor classifier. Without any training parameters, our method achieves results that are competitive with non-pretrained deep learning methods on six in-distribution datasets. It even outperforms BERT on all five OOD datasets, including four low-resource languages. Our method also excels in the few-shot setting, where labeled data are too scarce to train DNNs effectively. Code is available at

Model/Dataset	KinyarwandaNews		KirundiNews		DengueFilipino		SwahiliNews		SogouNews	
Shot#	Full	5-shot	Full	5-shot	Full	5-shot	Full	5-shot	Full	5-shot
Bi-LSTM+Attn	0.843	0.253±0.061	0.872	0.254±0.053	0.948	0.369±0.053	0.863	0.357±0.049	0.952	0.534±0.042
HAN	0.820	0.137±0.033	0.881	0.190±0.099	0.981	0.362±0.119	0.887	0.264±0.042	0.957	0.425±0.072
fastText	0.869	0.170±0.057	0.883	0.245±0.242	0.870	0.248±0.108	0.874	0.347±0.255	0.930	0.545±0.053
W2V	0.874	0.281±0.236	0.904	0.288±0.189	0.993	0.481±0.158	0.892	0.373±0.341	0.943	0.141±0.005
SentBERT	0.788	0.292±0.062	0.886	0.314±0.060	0.992	0.629±0.143	0.822	0.436±0.081	0.860	0.485±0.043
BERT	0.838	0.240±0.060	0.879	0.386±0.099	0.979	0.409±0.058	0.897	0.396±0.096	0.952	0.221±0.041
mBERT	0.835	0.229±0.066	0.874	0.324±0.071	0.983	0.465±0.048	0.906	0.558±0.169	0.953	0.282±0.060
gzip (ours)	**0.891**	**0.458±0.065**	**0.905**	**0.541±0.056**	**0.998**	**0.652±0.048**	**0.927**	**0.627±0.072**	**0.975**	**0.649±0.061**

Table 5: Test accuracy on OOD datasets with 95% confidence interval over five trials in five-shot setting.

Features vs Similarities

- Two perspectives on building machine learning systems:

Features vs Similarities

- Two perspectives on building machine learning systems:

  1. Feature-based: describe object properties via features and build models that use them.

     ✓ everything that we have seen so far, recall the feature map $\phi(x)$

Features vs Similarities

- Two perspectives on building machine learning systems:

1. Feature-based: describe object properties via features and build models that use them.
   - ✓ everything that we have seen so far, recall the feature map $\phi(x)$

2. Similarity-based: don't describe objects by their properties; rather, build systems based on **comparing** objects to each other
   - ✓ k nearest neighbors (previous slide); Gaussian processes; kernel methods (next)

Features vs Similarities

- Two perspectives on building machine learning systems:

  1. Feature-based: describe object properties via features and build models that use them.

     ✓ everything that we have seen so far, recall the feature map $\phi(x)$

  2. Similarity-based: don't describe objects by their properties; rather, build systems based on **comparing** objects to each other

     ✓ k nearest neighbors (previous slide); Gaussian processes; kernel methods (next)

- Sometimes the diference is unclear

Kernels

- Consider the set of objects $\mathcal{X}$ (no assumptions)

Kernels

- Consider the set of objects $\mathcal{X}$ (no assumptions)

- A kernel is a similarity function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ between pairs of objects.

Kernels

- Consider the set of objects $\mathcal{X}$ (no assumptions)
- A kernel is a similarity function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ between pairs of objects.
- A valid kernel is symmetric

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

and positive semi-definite (next)

Kernels

- Consider the set of objects $\mathcal{X}$ (no assumptions)
- A kernel is a similarity function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ between pairs of objects.
- A valid kernel is symmetric

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

  and positive semi-definite (next)
- Given set of objects $\{x_1, ..., x_N\}$, the Gram matrix $\boldsymbol{K}$ is the $N \times N$ matrix defined as:

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

Kernels

- Consider the set of objects $\mathcal{X}$ (no assumptions)
- A kernel is a similarity function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ between pairs of objects.
- A valid kernel is symmetric

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

  and positive semi-definite (next)
- Given set of objects $\{x_1, ..., x_N\}$, the Gram matrix $\boldsymbol{K}$ is the $N \times N$ matrix defined as:

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

- The kernel is positive semi-definite if, for all $N \in \mathbb{N}$, all sets of N objects $\{x_1, ..., x_N\} \subseteq \mathcal{X}$, and any $\mathbf{v} \in \mathbb{R}^N$

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

Kernels

- Mercer's Theorem: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \rightarrow \mathcal{H}$, such that

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

Kernels

- Mercer's Theorem: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \to \mathcal{H}$, such that

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

- A kernel corresponds to some a mapping in some **implicit** feature space!

Kernels

- Mercer's Theorem: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \to \mathcal{H}$, such that

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

- A kernel corresponds to some a mapping in some **implicit** feature space!
- Kernel trick: take a feature-based model (SVMs, logistic); replace explicit feature computations with kernel evaluations!

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

Kernels

- Mercer's Theorem: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \to \mathcal{H}$, such that

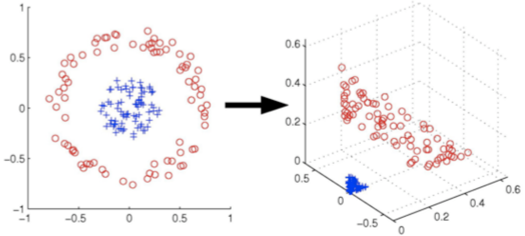
$$\kappa(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

- A kernel corresponds to some a mapping in some **implicit** feature space!
- Kernel trick: take a feature-based model (SVMs, logistic); replace explicit feature computations with kernel evaluations!

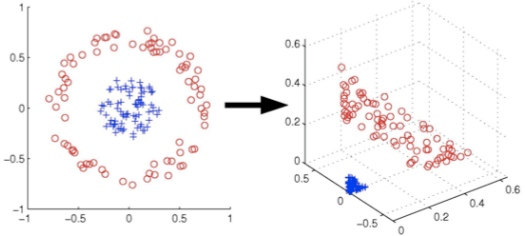
$$\boldsymbol{w}_y{}^T \phi(x) = \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} \alpha_{i,y} \kappa(x, x_i) \quad \text{for some } \alpha_{i,y} \in \mathbb{R}$$

- Extremely popular idea in the 1990-2000s!

Kernel Trick Illustration

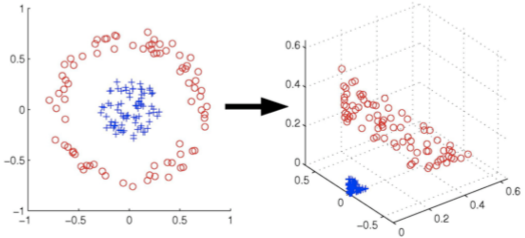


Kernel Trick Illustration



- Take $\mathcal{X} = \mathbb{R}^2$; feature map: $\phi([x_1, x_2]) = [x_1^2, \sqrt{2} x_1 x_2, x_2^2] \in \mathbb{R}^3$

$$
\begin{aligned}
\phi(x) \cdot \phi(z) &= [x_1^2, \sqrt{2} x_1 x_2, x_2^2] \cdot [z_1^2, \sqrt{2} z_1 z_2, z_2^2] \\
&= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\
&= ([x_1 \, x_2] \cdot [z_1, z_2])^2 \\
&= \kappa(x, z)
\end{aligned}
$$



- Take $\mathcal{X} = \mathbb{R}^2$; feature map: $\phi([x_1, x_2]) = [x_1^2, \sqrt{2}x_1\, x_2, x_2^2] \in \mathbb{R}^3$

$$\begin{aligned}
\phi(x)\cdot\phi(z) &= [x_1^2, \sqrt{2}x_1\, x_2, x_2^2]\cdot[z_1^2, \sqrt{2}z_1\, z_2, z_2^2] \\
&= x_1^2\, z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2\, z_2^2 \\
&= ([x_1\, x_2]\cdot[z_1, z_2])^2 \\
&= \kappa(x, z)
\end{aligned}$$

- The inner product in $\mathbb{R}^3$ is a function of the inner product in $\mathbb{R}^2$

Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space

Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space

- Computing a non-linear kernel is often better computationally than calculating the corresponding dot product in the high dimension feature space

Kernels = Tractable Non-Linearity

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

Kernels = Tractable Non-Linearity

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

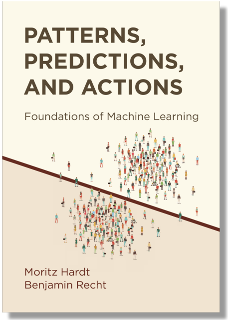
Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space
- Computing a non-linear kernel is often better computationally than calculating the corresponding dot product in the high dimension feature space
- Many models can be "kernelized" – learning algorithms generally solve the dual optimization problem (also convex)
- Drawback: quadratic dependency on dataset size
- Kernels decouple the learning algorithm (e.g., logistic, SVM) from the nature of the data: strings, images, sets, signals, graphs, probability distributions, ...

Conclusions

- Linear models are a broad class including the well-known perceptron, 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 is useful if training datasets are large
- However, linear models rely on specification of feature representations
- **Tomorrow:** methods that learn internal representations

Recommended Books



https://mlstory.org/

Learning Theory from First Principles

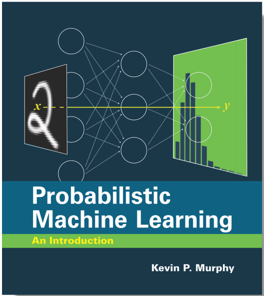
DRAFT

April 19, 2023

Francis Bach

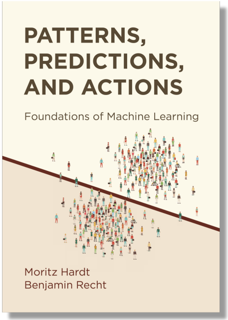
francis.bach@inria.fr

https://www.di.ens.fr/~fbach/ltfp_book.pdf

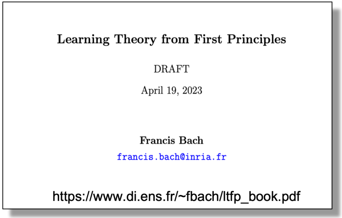


https://probml.github.io/pml-book/book1.html

Recommended Books



https://mlstory.org/


Learning Theory from First Principles

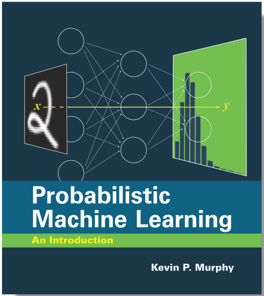
DRAFT

April 19, 2023

Francis Bach

francis.bach@inria.fr

https://www.di.ens.fr/~fbach/ltfp_book.pdf



https://probml.github.io/pml-book/book1.html

Thank you! Questions?