Lecture 2: Linear Models I

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Today's Roadmap

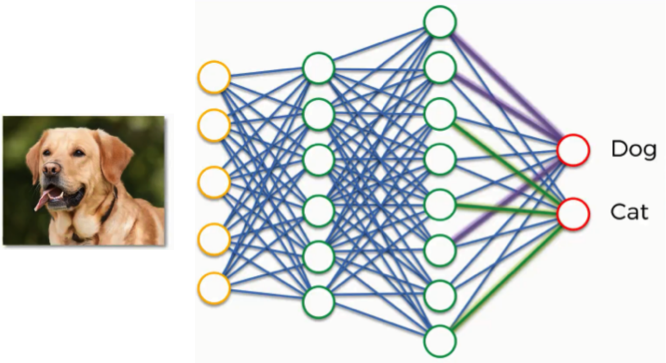
- Linear regression.
- Binary and multi-class linear classification.
- Linear classifiers: perceptron.

Why Linear Classifiers?

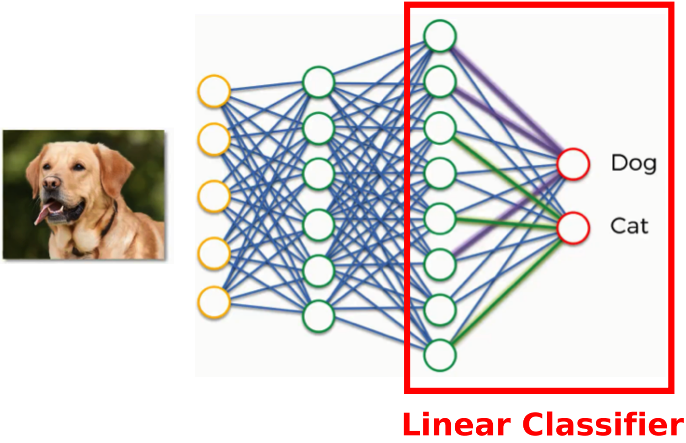
We know the course title promised "deep", but...

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

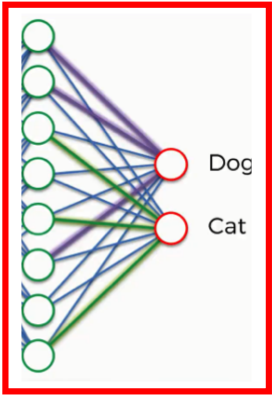
Linear Classifiers and Neural Networks



Linear Classifiers and Neural Networks



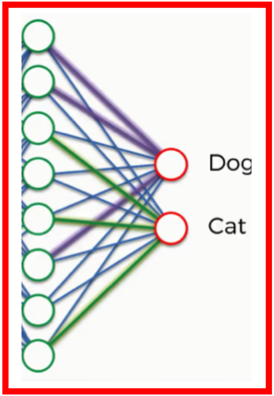
Linear Classifiers and Neural Networks



Linear Classifier

Linear Classifiers and Neural Networks

Handcrafted Features



Linear Classifier

Outline

1. Feature Representations
2. Linear Regression
3. Binary Classification and the Perceptron

Feature Representations

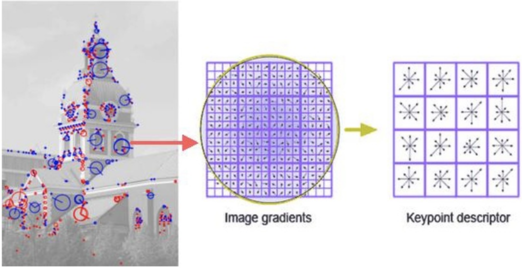
Feature engineering is an important step in linear classifiers:

- Bag-of-words features for text, also lemmas, parts-of-speech, ...

Feature Representations

Feature engineering is an important step in linear classifiers:

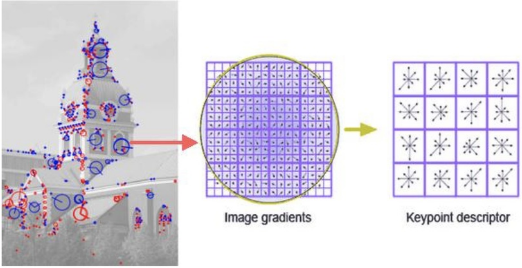
- Bag-of-words features for text, also lemmas, parts-of-speech, ...
- SIFT features and wavelet representations in computer vision.



Feature Representations

Feature engineering is an important step in linear classifiers:

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



- Other categorical, Boolean, and continuous features.

Feature Representations

Representing information about x:

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

Feature Representations

Representing information about x:

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

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

Feature Representations

Representing information about x:

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

- $\phi(x)$ is a (maybe high-dimensional) **feature vector**.
- $\phi(x)$ may include Boolean, categorical, and continuous features.

Feature Representations

Representing information about x:

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

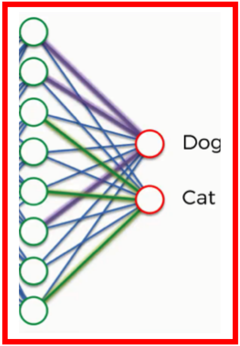
- $\phi(x)$ is a (maybe high-dimensional) **feature vector**.
- $\phi(x)$ may include Boolean, categorical, and continuous features.
- Categorical features can be reduced to a one-hot binary vector.

$\phi(x) \in \{1, 2, ..., K\}$ same information $\phi'(x) = [0, ..., 0, 1, 0, ..., 0] \in \{0, 1\}^K$

with one and only one 1.

Example: Continuous Features

Handcrafted Features



Linear Classifier

Feature Engineering and NLP Pipelines

Classical NLP pipelines: stacking together several linear classifiers.

Each output is used to handcraft features for subsequent classifiers.

Feature Engineering and NLP Pipelines

Classical NLP pipelines: stacking together several linear classifiers.

Each output is used to handcraft features for subsequent classifiers.

Examples of features:

- Word occurrences: binary features (word occurs or not in document).

Feature Engineering and NLP Pipelines

Classical NLP pipelines: stacking together several linear classifiers.

Each output is used to handcraft features for subsequent classifiers.

Examples of features:

- Word occurrences: binary features (word occurs or not in document).
- Word counts: numeric features, how many times a word occurs.

Feature Engineering and NLP Pipelines

Classical NLP pipelines: stacking together several linear classifiers.

Each output is used to handcraft features for subsequent classifiers.

Examples of features:

- Word occurrences: binary features (word occurs or not in document).
- Word counts: numeric features, how many times a word occurs.
- POS tags: classify words as nouns, verbs, adjectives, ...

Feature Engineering and NLP Pipelines

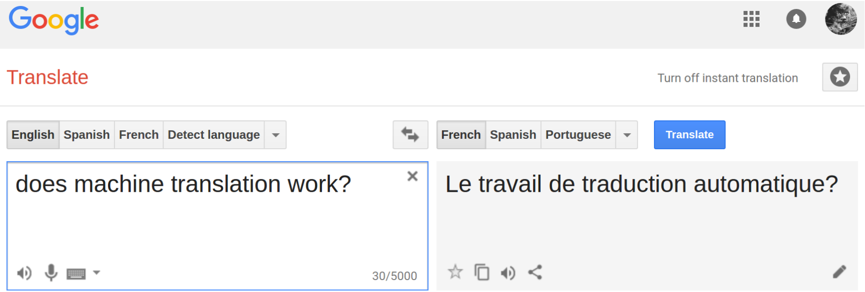
Classical NLP pipelines: stacking together several linear classifiers.

Each output is used to handcraft features for subsequent classifiers.

Examples of features:

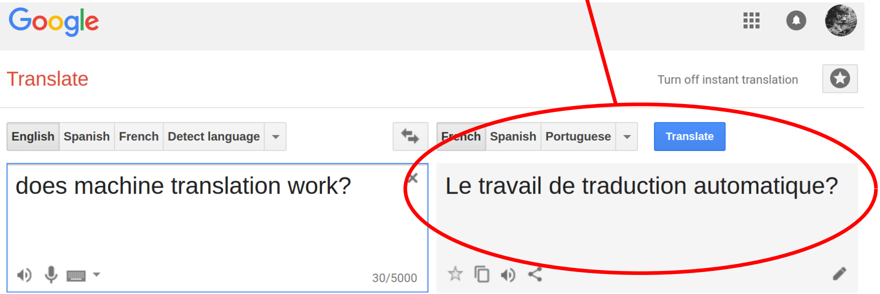
- Word occurrences: binary features (word occurs or not in document).
- Word counts: numeric features, how many times a word occurs.
- POS tags: classify words as nouns, verbs, adjectives, ...
- Spell check: misspellings counts for spam detection.

Example: Translation Quality Estimation

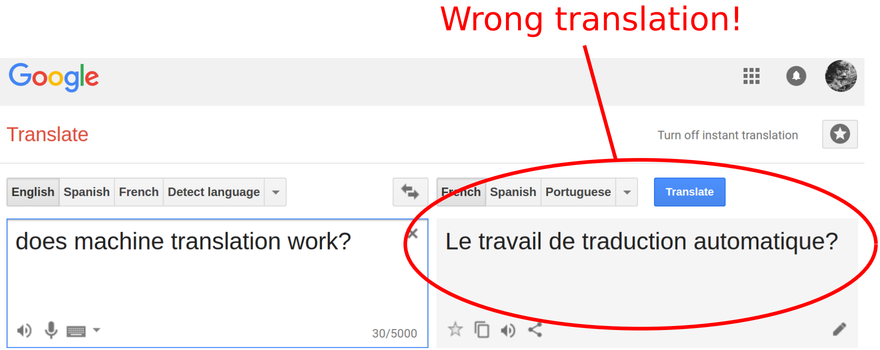


Example: Translation Quality Estimation

Wrong translation!



Example: Translation Quality Estimation



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

Example: Translation Quality Estimation

Hand-crafted features:

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

Representation Learning

Feature engineering is an "**art**" and can be very time-consuming

...but it's a good way of encoding **prior knowledge**, still widely used in practice (especially with "small data")

Representation Learning

Feature engineering is an "**art**" and can be very time-consuming

...but it's a good way of encoding **prior knowledge**, still widely used in practice (especially with "small data")

Alternative to feature engineering: **representation learning**

Representation Learning

Feature engineering is an "**art**" and can be very time-consuming

...but it's a good way of encoding **prior knowledge**, still widely used in practice (especially with "small data")

Alternative to feature engineering: **representation learning**

Neural networks will alleviate this (later in the course)!

Outline

1. Feature Representations

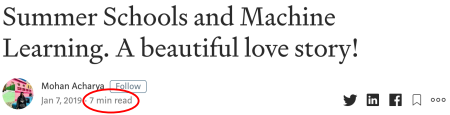
2. Linear Regression

3. Binary Classification and the Perceptron

Regression

Output space $\mathcal{Y}$ is continuous (*e.g.*, $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = [0, 1]$)

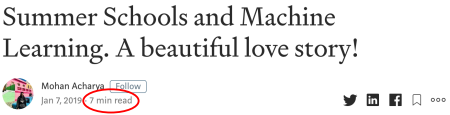
Example: given an article, how much time a user spends reading it?



Regression

Output space $\mathcal{Y}$ is continuous (*e.g.*, $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = [0, 1]$)

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

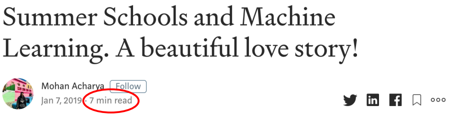


- Input x is number of words of the article
- Output $\hat{y}$ is the predicted reading time (minutes)

Regression

Output space $\mathcal{Y}$ is continuous (*e.g.*, $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = [0, 1]$)

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

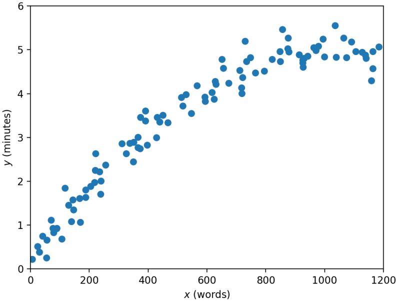


- Input x is number of words of the article
- Output $\hat{y}$ is the predicted reading time (minutes)

How to define the predictive model $\hat{y} = h(x)$?

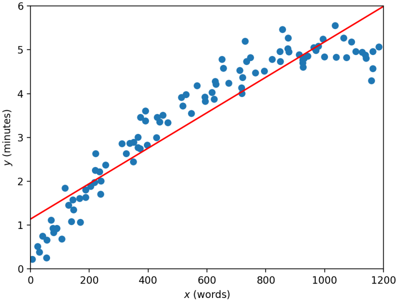
Linear Regression

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



Linear Regression

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



Least squares method: fit w and b on the training set by solving

$$\min_{w,\, b} \sum_{n=1}^{N} \Big(y_n - \underbrace{(w\, x_n + b)}_{\hat{y}_n}\Big)^2$$

Linear Regression

Often, linear dependency of $\widehat{y}$ on x is a bad/simplistic assumption

More general model: $\widehat{y} = w^T \phi(x)$, where $\phi(x)$ is a feature vector

- e.g. $\phi(x) = [1, x, x^2, \ldots, x^D]^T \in \mathbb{R}^{D+1}$ (monomials degree $\leq D$)
- the bias term b is captured by the constant feature $\phi_0(x) = 1$

Linear Regression

Often, linear dependency of $\widehat{y}$ on x is a bad/simplistic assumption

More general model: $\widehat{y} = w^T \phi(x)$, where $\phi(x)$ is a feature vector

- e.g. $\phi(x) = [1, x, x^2, \ldots, x^D]^T \in \mathbb{R}^{D+1}$ (monomials degree $\leq D$)
- the bias term b is captured by the constant feature $\phi_0(x) = 1$

Fit/learn w by solving $\min_{w \in \mathbb{R}^{D+1}} \sum_{n=1}^{N} \big(y_n - w^T \phi(x_n)\big)^2$

Linear Regression

Often, linear dependency of $\widehat{y}$ on x is a bad/simplistic assumption

More general model: $\widehat{y} = w^T \phi(x)$, where $\phi(x)$ is a feature vector

- e.g. $\phi(x) = [1, x, x^2, \ldots, x^D]^T \in \mathbb{R}^{D+1}$ (monomials degree $\leq D$)
- the bias term b is captured by the constant feature $\phi_0(x) = 1$

Fit/learn w by solving $\min_{w \in \mathbb{R}^{D+1}} \sum_{n=1}^{N} \left(y_n - w^T \phi(x_n)\right)^2$

- Closed-form solution:

$$\hat{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \text{ with } \boldsymbol{X} = \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_n)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix}, \; \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \\ \vdots \\ y_N \end{bmatrix}.$$

Linear Regression

Often, linear dependency of $\widehat{y}$ on x is a bad/simplistic assumption

More general model: $\widehat{y} = w^T\phi(x)$, where $\phi(x)$ is a feature vector

- e.g. $\phi(x) = [1, x, x^2, \ldots, x^D]^T \in \mathbb{R}^{D+1}$ (monomials degree $\leq D$)
- the bias term b is captured by the constant feature $\phi_0(x) = 1$

Fit/learn w by solving $\min_{w \in \mathbb{R}^{D+1}} \sum_{n=1}^{N} \left(y_n - w^T\phi(x_n)\right)^2$

- Closed-form solution:

$$\hat{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}, \text{ with } \boldsymbol{X} = \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_n)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix}, \; \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \\ \vdots \\ y_N \end{bmatrix}.$$

Still called linear regression – linear w.r.t. the model parameters w.

One-Slide Proof

Write the objective function in matrix-vector notation:

$$\sum_{n=1}^{N} \big(y_n - w^T \phi(x_n)\big)^2 = \|\boldsymbol{y} - \boldsymbol{X} w\|^2$$

One-Slide Proof

Write the objective function in matrix-vector notation:

$$\sum_{n=1}^{N}\left(y_n - w^T\phi(x_n)\right)^2 = \|\boldsymbol{y} - \boldsymbol{X}w\|^2$$

Equate the gradient to zero and solve the resulting equation:

$$\begin{aligned}
0 &= \nabla_w \|\boldsymbol{y} - \boldsymbol{X}w\|^2 \\
&= \nabla_w \left(w^T\boldsymbol{X}^T\boldsymbol{X}w - 2w^T\boldsymbol{X}^T\boldsymbol{y} + \|\boldsymbol{y}\|^2\right) \\
&= 2\boldsymbol{X}^T\boldsymbol{X}w - 2\boldsymbol{X}^T\boldsymbol{y}
\end{aligned}$$

One-Slide Proof

Write the objective function in matrix-vector notation:

$$\sum_{n=1}^{N} \left(y_n - w^T \phi(x_n)\right)^2 = \|\boldsymbol{y} - \boldsymbol{X} w\|^2$$

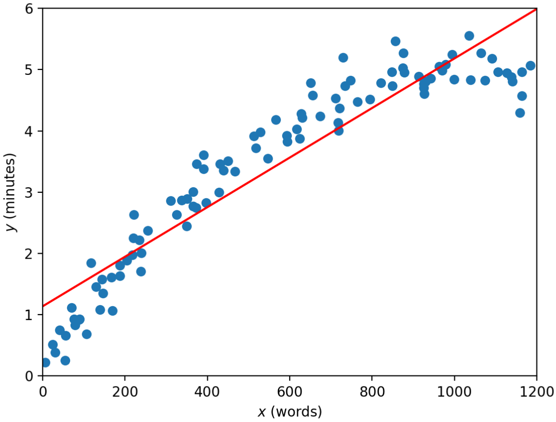
Equate the gradient to zero and solve the resulting equation:

$$\begin{aligned} 0 &= \nabla_w \|\boldsymbol{y} - \boldsymbol{X} w\|^2 \\ &= \nabla_w \left(w^T \boldsymbol{X}^T \boldsymbol{X} w - 2 w^T \boldsymbol{X}^T \boldsymbol{y} + \|\boldsymbol{y}\|^2 \right) \\ &= 2\boldsymbol{X}^T \boldsymbol{X} w - 2\boldsymbol{X}^T \boldsymbol{y} \end{aligned}$$

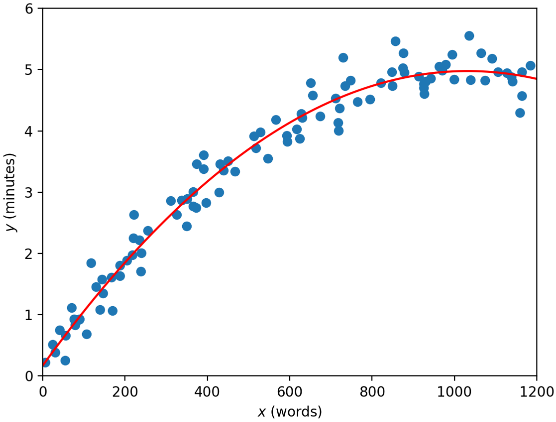
Therefore

$$\hat{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}.$$

Linear Regression ($D = 1$)



Linear Regression ($D = 2$)



Squared Loss Function

Linear regression with least squares criterion corresponds to a loss function

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

This is called the squared loss.

Squared Loss Function

Linear regression with least squares criterion corresponds to a loss function

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

This is called the squared loss.

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

$$\hat{w} = \arg\min_w \sum_{n=1}^{N} L(y_n - \hat{y}_n) = \arg\min_w \frac{1}{2} \sum_{n=1}^{N} \big(y_n - w^T \phi(x)\big)^2$$

(the factor $1/2$ is irrelevant but convenient)

Squared Loss Function

Linear regression with least squares criterion corresponds to a loss function

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

This is called the squared loss.

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

$$\hat{w} = \arg\min_w \sum_{n=1}^{N} L(y_n - \hat{y}_n) = \arg\min_w \frac{1}{2} \sum_{n=1}^{N} \big(y_n - w^T \phi(x)\big)^2$$

(the factor $1/2$ is irrelevant but convenient)

More later.

Least Squares (LS): Probabilistic Interpretation

Assume the data is generated stochastically as

$$y_n = w_*^T \phi(x_n) + z_n$$

where:

- ✓ w_* is the "true" model parameter vector;
- ✓ $z_n \sim \mathcal{N}(0, \sigma^2)$ are independent Gaussian noise samples, with zero mean variance σ^2.

Consequently,

$$y_n \sim \mathcal{N}(w_*^T \phi(x_n), \sigma^2).$$

Then $\hat{w}$ given by LS is the maximum likelihood estimate under this model.

One-Slide Proof

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

$$\hat{w}_{\mathrm{MLE}} = \arg\max_{w} P(\boldsymbol{y} \mid w)$$

One-Slide Proof

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

$$
\begin{aligned}
\hat{w}_{\text{MLE}} &= \arg\max_{w} P(\boldsymbol{y} \mid w) \\
&= \arg\max_{w} \log \prod_{n=1}^{N} P(y_n \mid w) \qquad \text{(independent samples)}
\end{aligned}
$$

One-Slide Proof

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

$$
\begin{aligned}
\hat{w}_{\text{MLE}} &= \arg\max_w P(\boldsymbol{y} \mid w) \\
&= \arg\max_w \log \prod_{n=1}^{N} P(y_n \mid w) && \text{(independent samples)} \\
&= \arg\max_w \sum_{n=1}^{N} \log P(y_n \mid w) && (\log \prod = \sum \log)
\end{aligned}
$$

One-Slide Proof

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

$$\begin{aligned}
\hat{w}_{\text{MLE}} &= \arg\max_w P(\boldsymbol{y} \mid w) \\
&= \arg\max_w \log \prod_{n=1}^{N} P(y_n \mid w) && \text{(independent samples)} \\
&= \arg\max_w \sum_{n=1}^{N} \log P(y_n \mid w) && \left(\log \prod = \sum \log\right) \\
&= \arg\max_w \sum_{n=1}^{N} -\frac{(y_n - w^T \phi(x_n))^2}{2\sigma^2} - \underbrace{\log(\sqrt{2\pi\sigma^2})}_{\text{constant}} && (\log \mathcal{N})
\end{aligned}$$

One-Slide Proof

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

$$
\begin{aligned}
\hat{w}_{\text{MLE}} &= \arg\max_w P(\boldsymbol{y} \mid w) \\
&= \arg\max_w \log \prod_{n=1}^{N} P(y_n \mid w) && \text{(independent samples)} \\
&= \arg\max_w \sum_{n=1}^{N} \log P(y_n \mid w) && (\log \textstyle\prod = \sum \log) \\
&= \arg\max_w \sum_{n=1}^{N} -\frac{(y_n - w^T\phi(x_n))^2}{2\sigma^2} - \underbrace{\log(\sqrt{2\pi\sigma^2})}_{\text{constant}} && (\log \mathcal{N}) \\
&= \arg\min_w \sum_{n=1}^{N} (y_n - w^T\phi(x_n))^2
\end{aligned}
$$

One-Slide Proof

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

$$
\begin{aligned}
\hat{w}_{\text{MLE}} &= \arg\max_w P(\boldsymbol{y} \mid w) \\
&= \arg\max_w \log \prod_{n=1}^{N} P(y_n \mid w) && \text{(independent samples)} \\
&= \arg\max_w \sum_{n=1}^{N} \log P(y_n \mid w) && (\log \textstyle\prod = \sum \log) \\
&= \arg\max_w \sum_{n=1}^{N} -\frac{(y_n - w^T \phi(x_n))^2}{2\sigma^2} - \underbrace{\log(\sqrt{2\pi\sigma^2})}_{\text{constant}} && (\log \mathcal{N}) \\
&= \arg\min_w \sum_{n=1}^{N} (y_n - w^T \phi(x_n))^2
\end{aligned}
$$

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

Other Regression Losses

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

Other Regression Losses

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

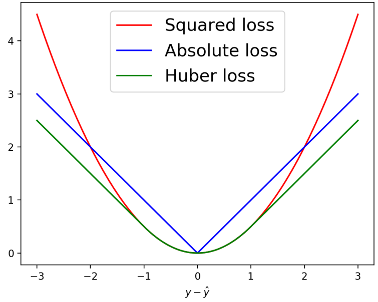
Absolute error loss: $L(y, \widehat{y}) = |y - \widehat{y}|$.

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

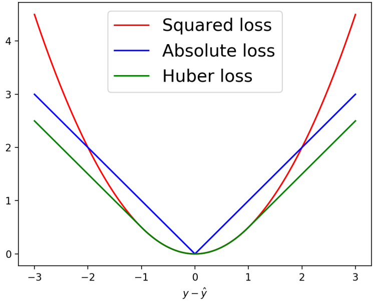


Other Regression Losses

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

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

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



Quizz: which of these are convex; and strictly convex?

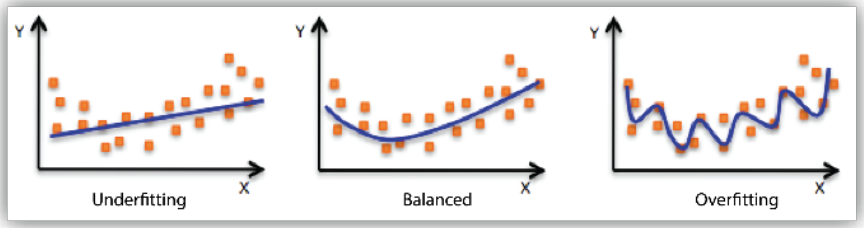
Overfitting and Underfitting

We saw earlier an example of underfitting (slide 15)

Overfitting and Underfitting

We saw earlier an example of underfitting (slide 15)

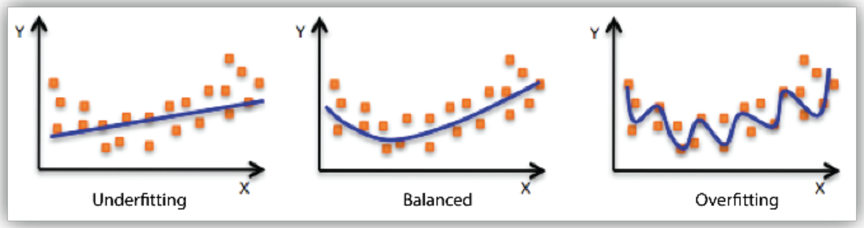
However, if the model is too complex (too many parameters) and/or the data is scarce, we run the risk of overfitting



Overfitting and Underfitting

We saw earlier an example of underfitting (slide 15)

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



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

Maximum A Posteriori

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

Maximum A Posteriori

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

A criterion to estimate w_* is the maximum a posteriori (MAP):

$$\hat{w}_{\text{MAP}} = \arg\max_{w} P(w \mid \boldsymbol{y}) = P(\boldsymbol{y}|w)\, P(w)/P(\boldsymbol{y}) \qquad (\textit{Bayes})$$

Maximum A Posteriori

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

A criterion to estimate w_* is the maximum a posteriori (MAP):

$$\begin{aligned}
\hat{w}_{\text{MAP}} &= \arg\max_w P(w \mid \boldsymbol{y}) \;=\; P(\boldsymbol{y}|w)\, P(w)/P(\boldsymbol{y}) \qquad (\textit{Bayes}) \\
&= \arg\max_w P(w) \prod_{n=1}^{N} P(y_n \mid w)
\end{aligned}$$

Maximum A Posteriori

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

A criterion to estimate w_* is the maximum a posteriori (MAP):

$$\begin{aligned}
\hat{w}_{\text{MAP}} &= \arg\max_w P(w \mid \boldsymbol{y}) \;=\; P(\boldsymbol{y}|w)\, P(w)/P(\boldsymbol{y}) \qquad \textit{(Bayes)} \\
&= \arg\max_w P(w) \prod_{n=1}^{N} P(y_n \mid w) \\
&= \arg\max_w \log P(w) + \sum_{n=1}^{N} \log P(y_n \mid w)
\end{aligned}$$

Maximum A Posteriori

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

A criterion to estimate w_* is the maximum a posteriori (MAP):

$$
\begin{aligned}
\hat{w}_{\text{MAP}} &= \arg\max_w P(w \mid \boldsymbol{y}) \;=\; P(\boldsymbol{y}|w)\, P(w)/P(\boldsymbol{y}) \qquad (\textit{Bayes}) \\
&= \arg\max_w P(w) \prod_{n=1}^{N} P(y_n \mid w) \\
&= \arg\max_w \log P(w) + \sum_{n=1}^{N} \log P(y_n \mid w) \\
&= \arg\max_w -\frac{\|w\|^2}{2\tau^2} - \sum_{n=1}^{N} \frac{(y_n - w^T \phi(x_n))^2}{2\sigma^2} + \text{constant}
\end{aligned}
$$

Maximum A Posteriori

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

A criterion to estimate w_* is the maximum a posteriori (MAP):

$$
\begin{aligned}
\hat{w}_{\text{MAP}} &= \arg\max_w P(w \mid \boldsymbol{y}) \;=\; P(\boldsymbol{y}|w)\, P(w)/P(\boldsymbol{y}) \qquad (\textit{Bayes}) \\
&= \arg\max_w P(w) \prod_{n=1}^{N} P(y_n \mid w) \\
&= \arg\max_w \log P(w) + \sum_{n=1}^{N} \log P(y_n \mid w) \\
&= \arg\max_w -\frac{\|w\|^2}{2\tau^2} - \sum_{n=1}^{N} \frac{(y_n - w^T \phi(x_n))^2}{2\sigma^2} + \text{constant} \\
&= \arg\min_w \lambda \underbrace{\|w\|^2}_{\text{regularizer}} + \underbrace{\sum_{n=1}^{N} (y_n - w^T \phi(x_n))^2}_{\text{loss}}
\end{aligned}
$$

where $\lambda = \sigma^2/\tau^2$ is the so-called regularization constant

Maximum A Posteriori

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

A criterion to estimate w_* is the maximum a posteriori (MAP):

$$
\begin{aligned}
\hat{w}_{\text{MAP}} &= \arg\max_w P(w \mid \boldsymbol{y}) \;=\; P(\boldsymbol{y}|w)\, P(w)/P(\boldsymbol{y}) \qquad (\textit{Bayes}) \\
&= \arg\max_w P(w) \prod_{n=1}^{N} P(y_n \mid w) \\
&= \arg\max_w \log P(w) + \sum_{n=1}^{N} \log P(y_n \mid w) \\
&= \arg\max_w -\frac{\|w\|^2}{2\tau^2} - \sum_{n=1}^{N} \frac{(y_n - w^T \phi(x_n))^2}{2\sigma^2} + \text{constant} \\
&= \arg\min_w \lambda \underbrace{\|w\|^2}_{\text{regularizer}} + \underbrace{\sum_{n=1}^{N} (y_n - w^T \phi(x_n))^2}_{\text{loss}}
\end{aligned}
$$

where $\lambda = \sigma^2/\tau^2$ is the so-called regularization constant

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

Outline

1. Feature Representations

2. Linear Regression

3. Binary Classification and the Perceptron

Binary Classification

Before multi-class, we start with simpler case of binary classification

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

Binary Classification

Before multi-class, we start with simpler case of binary classification

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

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

Binary Classification

Before multi-class, we start with simpler case of 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 $+1$ (true) or -1 (fake)

Binary Classification

Before multi-class, we start with simpler case of 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 $+1$ (true) or -1 (fake)

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

Linear Classifier

Defined by

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

Linear Classifier

Defined by

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

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

Linear Classifier

Defined by

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

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

Different from regression: sign function converts from continuous to binary

Linear Classifier

Defined by

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

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

Different from regression: sign function converts from continuous to binary

The decision boundary is a hyperplane (w.r.t. $\phi(x)$, not x)

$$w^T\phi(x) + b = 0$$

Linear Classifier

Defined by

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

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

Different from regression: sign function converts from continuous to binary

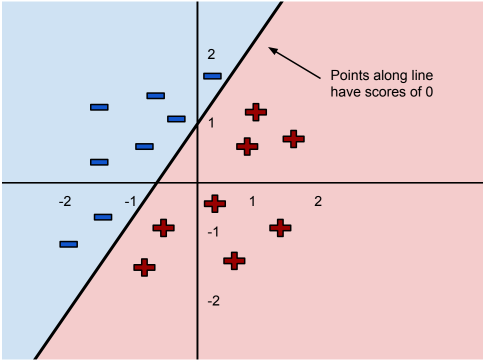
The decision boundary is a hyperplane (w.r.t. $\phi(x)$, not x)

$$w^T \phi(x) + b = 0$$

Thus also called a "hyperplane classifier."

Linear Classifier

(w, b) defines a hyperplane that splits the space into two half spaces:

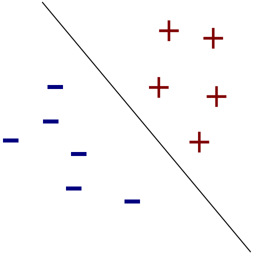


How to learn this hyperplane from the training data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$?

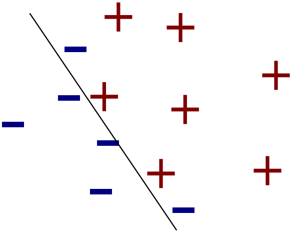
Linear Separability

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

Separable



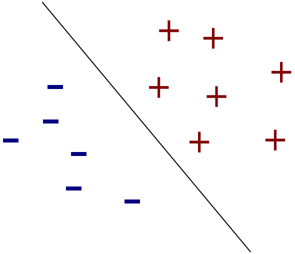
Not separable



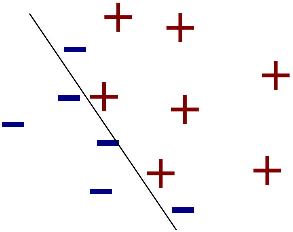
Linear Separability

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

Separable



Not separable



Next: an algorithm that finds such an hyperplane if it exists.

Linear Classifier: No Bias Term

It is common to present linear classifiers without the bias term b:

$$\widehat{y} = \text{sign}(w^T \phi(x) + b)$$

If $b = 0$, the decision hyperplane passes through the origin

Linear Classifier: No Bias Term

It is common to present linear classifiers without the bias term b:

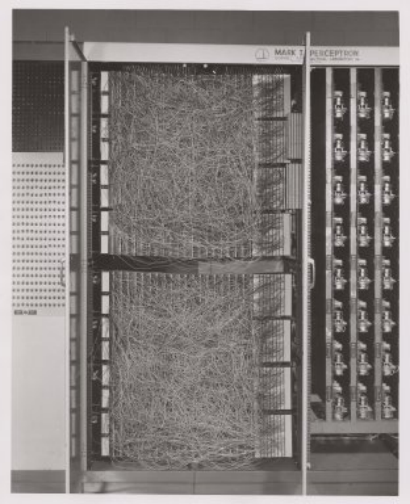
$$\widehat{y} = \text{sign}(w^T \phi(x) + b)$$

If $b = 0$, the decision hyperplane passes through the origin

We can always do this without loss of generality:

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

Perceptron (Rosenblatt, 1958)



(Source: Wikipedia)

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

Perceptron in the News...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

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

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

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

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

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

Without Human Controls

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

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

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

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

1958 New York Times...

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

Learns by Doing

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

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

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

Perceptron in the News...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

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

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

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

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

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

Without Human Controls

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

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

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

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

1958 New York Times...

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

Learns by Doing

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

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

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

Perceptron Algorithm

Online algorithm: process one data point x_n at each round

1. Apply current model to x_n, get the corresponding prediction
2. If prediction is correct, do nothing
3. If it is wrong, correct w by adding/subtracting feature vector $\phi(x_i)$

Perceptron Algorithm

input: labeled data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$
initialize $w^{(0)} = 0$
initialize $k = 0$ (number of mistakes)
repeat
  get new training example (x_n, y_n)
  predict $\widehat{y}_n = \text{sign}((w^{(k)})^T \phi(x_n))$
  if $\widehat{y}_n \neq y_n$ **then**
    update $w^{(k+1)} = w^{(k)} + y_n\, \phi(x_n)$
    increment k
  end if
until maximum number of epochs
output: model weights $w^{(k)}$

Perceptron's Mistake Bound

Definitions:

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

$$y_n \, u^T \phi(x_n) \geq \gamma, \quad \forall n.$$

Perceptron's Mistake Bound

Definitions:

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

$$y_n \; u^T \phi(x_n) \geq \gamma, \quad \forall n.$$

- Radius of the data: $R = \max_n \|\phi(x_n)\|$.

Perceptron's Mistake Bound

Definitions:

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

$$y_n \; u^T \phi(x_n) \geq \gamma, \quad \forall n.$$

- Radius of the data: $R = \max_n \|\phi(x_n)\|$.

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 $w^{(k+1)} = w^{(k)} + y_n \phi(x_n)$.

One-Slide Proof

Recall that $w^{(k+1)} = w^{(k)} + y_n \phi(x_n)$.

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

$$
\begin{aligned}
u^T w^{(k)} &= u^T w^{(k-1)} + y_n u^T \phi(x_n) \\
&\geq u^T w^{(k-1)} + \gamma \\
&\geq u^T w^{(k-2)} + \gamma + \gamma \\
&\geq k\gamma \qquad \text{(recall } w^{(0)} = 0\text{)}
\end{aligned}
$$

One-Slide Proof

Recall that $w^{(k+1)} = w^{(k)} + y_n \phi(x_n)$.

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

$$\begin{aligned} u^T w^{(k)} &= u^T w^{(k-1)} + y_n u^T \phi(x_n) \\ &\geq u^T w^{(k-1)} + \gamma \\ &\geq u^T w^{(k-2)} + \gamma + \gamma \\ &\geq k\gamma \qquad \text{(recall } w^{(0)} = 0\text{)} \end{aligned}$$

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

One-Slide Proof

Recall that $w^{(k+1)} = w^{(k)} + y_n \phi(x_n)$.

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

$$
\begin{aligned}
u^T w^{(k)} &= u^T w^{(k-1)} + y_n u^T \phi(x_n) \\
&\geq u^T w^{(k-1)} + \gamma \\
&\geq u^T w^{(k-2)} + \gamma + \gamma \\
&\geq k\gamma \qquad \text{(recall } w^{(0)} = 0\text{)}
\end{aligned}
$$

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

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

$$
\begin{aligned}
\|w^{(k)}\|^2 &= \|w^{(k-1)}\|^2 + \|\phi(x_n)\|^2 + 2 y_n \overbrace{(w^{(k)})^T \phi(x_n)}^{\leq 0, \text{ if update}} \\
&\leq \|w^{(k-1)}\|^2 + R^2 \\
&\leq k R^2.
\end{aligned}
$$

One-Slide Proof

Recall that $w^{(k+1)} = w^{(k)} + y_n \phi(x_n)$.

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

$$\begin{aligned} u^T w^{(k)} &= u^T w^{(k-1)} + y_n u^T \phi(x_n) \\ &\geq u^T w^{(k-1)} + \gamma \\ &\geq u^T w^{(k-2)} + \gamma + \gamma \\ &\geq k\gamma \qquad \text{(recall } w^{(0)} = 0\text{)} \end{aligned}$$

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

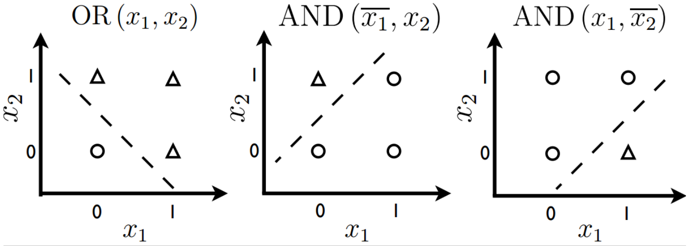
- **Upper bound on** $\|w^{(k)}\|$:

$$\begin{aligned} \|w^{(k)}\|^2 &= \|w^{(k-1)}\|^2 + \|\phi(x_n)\|^2 + 2 y_n \overbrace{(w^{(k)})^T \phi(x_n)}^{\leq 0, \text{ if update}} \\ &\leq \|w^{(k-1)}\|^2 + R^2 \\ &\leq k R^2. \end{aligned}$$

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

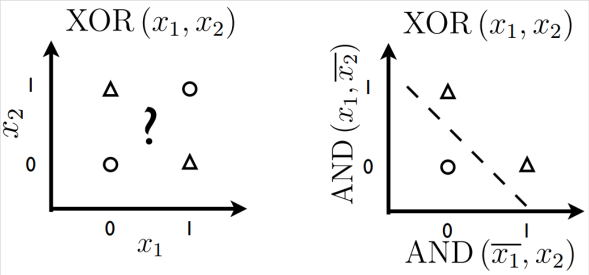
What a Simple Perceptron Can and Can't Do

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



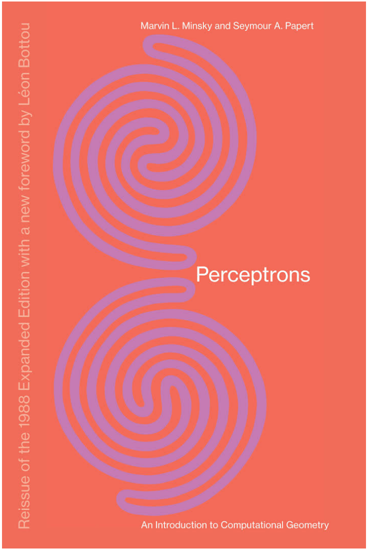
What a Simple Perceptron Can and Can't Do

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



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

Limitations of the Perceptron



Minsky and Papert (1969):

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

More later in the neural networks' lecture!

Multi-Class Classification

Assume a multi-class problem, with $|\mathcal{Y}| \geq 2$ labels (classes).

Several strategies to reduce to binary classification:

Multi-Class Classification

Assume a multi-class problem, with $|\mathcal{Y}| \geq 2$ labels (classes).

Several strategies to reduce to binary classification:

- One-vs-all: train one binary classifier per class, using all others as negative examples; pick the class with the highest score.

Multi-Class Classification

Assume a multi-class problem, with $|\mathcal{Y}| \geq 2$ labels (classes).

Several strategies to reduce to binary classification:

- One-vs-all: train one binary classifier per class, using all others as negative examples; pick the class with the highest score.
- One-vs-one: another strategy is to train pairwise classifiers and use majority voting.

Multi-Class Classification

Assume a multi-class problem, with $|\mathcal{Y}| \geq 2$ labels (classes).

Several strategies to reduce to binary classification:

- One-vs-all: train one binary classifier per class, using all others as negative examples; pick the class with the highest score.
- One-vs-one: another strategy is to train pairwise classifiers and use majority voting.
- Binary coding: use binary code for the class labels and learn a binary classifier for each bit.

Multi-Class Classification

Assume a multi-class problem, with $|\mathcal{Y}| \geq 2$ labels (classes).

Several strategies to reduce to binary classification:

- One-vs-all: train one binary classifier per class, using all others as negative examples; pick the class with the highest score.
- One-vs-one: another strategy is to train pairwise classifiers and use majority voting.
- Binary coding: use binary code for the class labels and learn a binary classifier for each bit.

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

Multi-Class Linear Classifiers

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

$$\boldsymbol{W} = \begin{bmatrix} w_1^\top \\ \vdots \\ w_{|\mathcal{Y}|}^\top \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_{|\mathcal{Y}|} \end{bmatrix}.$$

Multi-Class Linear Classifiers

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

$$\boldsymbol{W} = \begin{bmatrix} w_1^\top \\ \vdots \\ w_{|\mathcal{Y}|}^\top \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_{|\mathcal{Y}|} \end{bmatrix}.$$

- Equivalently, $|\mathcal{Y}|$ weight vectors $w_y \in \mathbb{R}^D$ and scalars $b_y \in \mathbb{R}$

Multi-Class Linear Classifiers

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

$$\boldsymbol{W} = \begin{bmatrix} w_1^\top \\ \vdots \\ w_{|\mathcal{Y}|}^\top \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_{|\mathcal{Y}|} \end{bmatrix}.$$

- Equivalently, $|\mathcal{Y}|$ weight vectors $w_y \in \mathbb{R}^D$ and scalars $b_y \in \mathbb{R}$
- The score of a particular label is based on a linear combination of features and their weights

Multi-Class Linear Classifiers

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

$$\boldsymbol{W} = \begin{bmatrix} w_1^\top \\ \vdots \\ w_{|\mathcal{Y}|}^\top \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_{|\mathcal{Y}|} \end{bmatrix}.$$

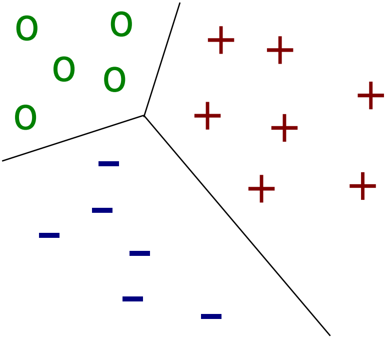
- Equivalently, $|\mathcal{Y}|$ weight vectors $w_y \in \mathbb{R}^D$ and scalars $b_y \in \mathbb{R}$
- The score of a particular label is based on a linear combination of features and their weights
- Predict the $\widehat{y}$ which maximizes this score:

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

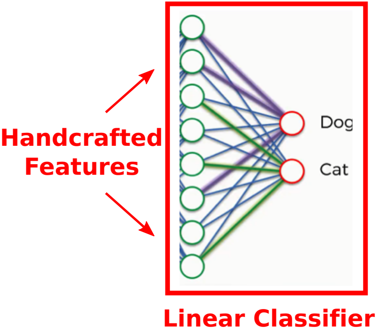
$$\equiv \arg\max(\boldsymbol{W}\phi(x) + \boldsymbol{b}) \quad \text{(compact notation)}$$

Multi-Class Linear Classifier

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



Commonly Used Notation in Neural Networks



$$\widehat{y} = \arg\max \left(\boldsymbol{W} \phi(x) + \boldsymbol{b} \right), \quad \boldsymbol{W} = \begin{bmatrix} w_1^\top \\ \vdots \\ w_{|\mathcal{Y}|}^\top \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_{|\mathcal{Y}|} \end{bmatrix}.$$

Multi-Class Recovers Binary

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

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

Multi-Class Recovers Binary

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

$$\begin{aligned}\widehat{y} &= \arg\max_{y \in \{\pm 1\}} (w_y)^T \phi(x) + b_y \\ &= \begin{cases} +1 & \text{if } (w_{+1})^T \phi(x) + b_{+1} > (w_{-1})^T \phi(x) + b_{-1} \\ -1 & \text{otherwise} \end{cases}\end{aligned}$$

Multi-Class Recovers Binary

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

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

Multi-Class Recovers Binary

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

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

That is: only half of the parameters are needed.

Linear Classifiers (Binary vs Multi-Class)

- Prediction rule:

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

Linear Classifiers (Binary vs Multi-Class)

- Prediction rule:

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

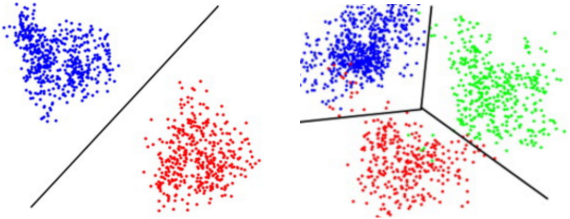
- The decision boundary is defined by the intersection of half spaces

Linear Classifiers (Binary vs Multi-Class)

- Prediction rule:

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

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



Linear Classifier – No Bias Term

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

$$\widehat{y} \;=\; \arg\max_{y \in \mathcal{Y}} \, (w_y)^T \phi(x) + b_y$$

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

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

We assume this for simplicity.

Example: Perceptron

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

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

Perceptron Algorithm: Multi-Class

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

Conclusions

- Linear models involve manipulating weights and features.
- Linear regression is a simple method for regression which has a closed form solution.
- Linear classifiers include several well-known ML methods (both for binary and multi-class classification).
- Today we saw the perceptron and proved a mistake bound.
- Next class: logistic regression (another linear classifier).

References I

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

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

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