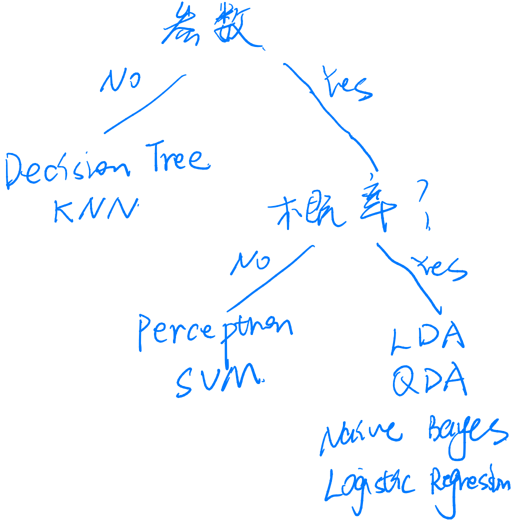
Linear Models for Classification

Table of contents



Section 1

Introduction

Classification: Definition

- Given a collection of records (training set)
  - Each record $(\mathbf{x}, y)$ contains a set of attributes/features/feature variables denoted as $\mathbf{x} \in \mathbb{R}^d$, and one target variable called class $y \in \{0, 1, \ldots, K-1\}$.
  - The entire training set can be denoted as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$.

Classification: Definition

- Given a collection of records (training set)
  - Each record $(\mathbf{x}, y)$ contains a set of attributes/features/feature variables denoted as $\mathbf{x} \in \mathbb{R}^d$, and one target variable called class $y \in \{0, 1, \dots, K-1\}$.
  - The entire training set can be denoted as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$.
- Find a model for class as a function of the values of other attributes.

Classification: Definition

- Given a collection of records (training set)
  - Each record $(\mathbf{x}, y)$ contains a set of attributes/features/feature variables denoted as $\mathbf{x} \in \mathbb{R}^d$, and one target variable called class $y \in \{0, 1, \ldots, K-1\}$.
  - The entire training set can be denoted as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$.
- Find a model for class as a function of the values of other attributes.
- Goal: previously unseen records should be assigned a class as accurately as possible.

Classification: Definition

- Given a collection of records (training set)
  - Each record $(\mathbf{x}, y)$ contains a set of attributes/features/feature variables denoted as $\mathbf{x} \in \mathbb{R}^d$, and one target variable called class $y \in \{0, 1, \ldots, K-1\}$.
  - The entire training set can be denoted as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$.
- Find a model for class as a function of the values of other attributes.
- Goal: previously unseen records should be assigned a class as accurately as possible.
  - A test set is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

Different Approaches to Classification

- Construct a **discriminant function** which assigns each vector $\mathbf{x}$ to a specific class

Different Approaches to Classification

- Construct a **discriminant function** which assigns each vector $\mathbf{x}$ to a specific class
- Model the conditional probability distribution $p(y = k|\mathbf{x})$ in an inference stage, and use this distribution to make optimal decisions

Different Approaches to Classification

- Construct a **discriminant function** which assigns each vector $\mathbf{x}$ to a specific class
- Model the conditional probability distribution $p(y = k|\mathbf{x})$ in an inference stage, and use this distribution to make optimal decisions
  - Two methods to model $p(y = k|\mathbf{x})$
    - Discriminant method
      Example: Representing $p(y = k|\mathbf{x})$ as parametric models and then optimizing the parameters using a training set

Different Approaches to Classification

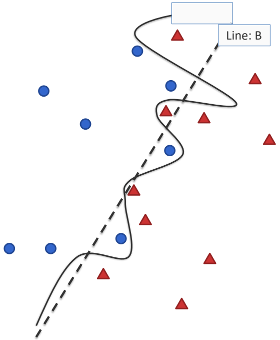
- Construct a **discriminant function** which assigns each vector $\mathbf{x}$ to a specific class
- Model the conditional probability distribution $p(y = k|\mathbf{x})$ in an inference stage, and use this distribution to make optimal decisions
  - Two methods to model $p(y = k|\mathbf{x})$
    - Discriminant method
      Example: Representing $p(y = k|\mathbf{x})$ as parametric models and then optimizing the parameters using a training set
    - Generative method
      Model the class-conditional densities $p(\mathbf{x}|y = k)$ and prior probabilities $p(y = k)$, and compute $p(y = k|\mathbf{x})$ using Bayes theorem

$$p(y = k|\mathbf{x}) = \frac{p(\mathbf{x}|y = k)p(y = k)}{p(\mathbf{x})}$$

Decision Boundary

In the most common scenario, the classes are taken to be disjoint, so that each input $(\mathbf{x})$ is assigned to **one and only one class** (y).

- The input space is thereby divided into *decision regions* whose boundaries are called **decision boundaries** or **decision surfaces**.



- **Curve: A** and **Line: B** are decision boundaries for two different classifiers.
- The decision boundary **Line : B** is linear.

In the next few lectures, we consider classification models with linear decision boundaries.

Section 2

Perceptron

Case Study: Credit Approval

age	32 years
gender	male
salary	$40,000$
debt	$26,000$
years in job	1 year
years at home	3 years
...	...

Approve for credit?

- Using salary, debt, years in residence, etc. approve for credit or not
- Banks have lots of data
  - customer information: salary, debt, etc.
  - whether or not they defaulted on their credit.

- Binary Classification: Two classes, i.e, $K = 2$

Case Study: Credit Approval

age	32 years
gender	male
salary	$40,000$
debt	$26,000$
years in job	1 year
years at home	3 years
...	...

Approve for credit?

- Using salary, debt, years in residence, etc. approve for credit or not
- Banks have lots of data
  - customer information: salary, debt, etc.
  - whether or not they defaulted on their credit.

- Binary Classification: Two classes, i.e, $K = 2$
- For convenience, instead of $y \in \{0, 1\}$, sometimes, we use $y \in \{-1, +1\}$ to denote the two classes.

A Simple Model: Linear Threshold Units

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^\top$, Give importance weights to the different features and compute a "Credit Score"

A Simple Model: Linear Threshold Units

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^\top$, Give importance weights to the different features and compute a "Credit Score"

$$\text{“Credit Score”} = \sum_{i=1}^{d} w_i x_i.$$

A Simple Model: Linear Threshold Units

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^\top$, Give importance weights to the different features and compute a "Credit Score"

$$\text{“Credit Score”} = \sum_{i=1}^{d} w_i x_i.$$

- Approve credit if the "Credit Score" is acceptable.
    - Approve credit if $\sum_{i=1}^{d} w_i x_i >$ threshold, ("Credit score" is good)
    - Deny credit if $\sum_{i=1}^{d} w_i x_i <$ threshold, ("Credit score" is bad)

A Simple Model: Linear Threshold Units

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^\top$, Give importance weights to the different features and compute a "Credit Score"

$$\text{“Credit Score”} = \sum_{i=1}^{d} w_i x_i.$$

- Approve credit if the "Credit Score" is acceptable.
  - Approve credit if $\sum_{i=1}^{d} w_i x_i >$ threshold, ("Credit score" is good)
  - Deny credit if $\sum_{i=1}^{d} w_i x_i <$ threshold, ("Credit score" is bad)
- How to choose the importance weights w_i?
  - x_i is important $\Rightarrow$ large weight

A Simple Model: Linear Threshold Units

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^\top$, Give importance weights to the different features and compute a "Credit Score"

$$\text{“Credit Score”} = \sum_{i=1}^{d} w_i x_i.$$

- Approve credit if the "Credit Score" is acceptable.
  - Approve credit if $\sum_{i=1}^{d} w_i x_i >$ threshold, ("Credit score" is good)
  - Deny credit if $\sum_{i=1}^{d} w_i x_i <$ threshold, ("Credit score" is bad)
- How to choose the importance weights w_i?
  - x_i is important $\Rightarrow$ large weight
  - x_i is beneficial for credit $\Rightarrow$ positive weight
  - x_i is detrimental for credit $\Rightarrow$ negative weight

A Simple Model: Linear Threshold Units

- Approve credit if $\sum_{i=1}^{d} w_i x_i >$ threshold, ("Credit score" is good)
- Deny credit if $\sum_{i=1}^{d} w_i x_i <$ threshold, ("Credit score" is bad)

A Simple Model: Linear Threshold Units

- Approve credit if $\sum_{i=1}^{d} w_i x_i >$ threshold, ("Credit score" is good)
- Deny credit if $\sum_{i=1}^{d} w_i x_i <$ threshold, ("Credit score" is bad)

These can be formally formulated as follows

$$y(\mathbf{x}) = \mathrm{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right)$$

where w_0 is the "bias weight". The "bias weight" w_0 correspond to the threshold.

A Simple Model: Linear Threshold Units

- Approve credit if $\sum_{i=1}^{d} w_i x_i >$ threshold, ("Credit score" is good)
- Deny credit if $\sum_{i=1}^{d} w_i x_i <$ threshold, ("Credit score" is bad)

These can be formally formulated as follows

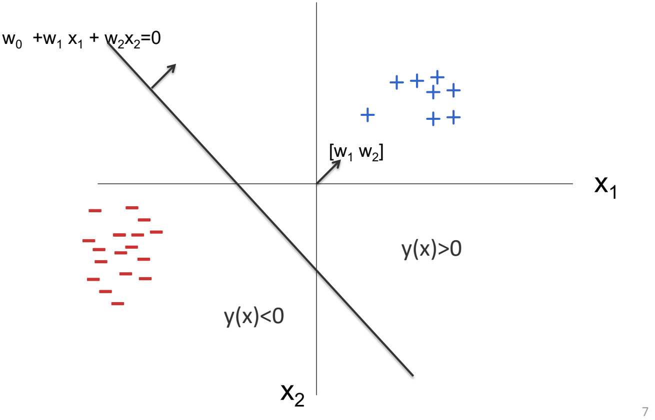
$$y(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right)$$

where w_0 is the "bias weight". The "bias weight" w_0 correspond to the threshold.

Now, it remains to "learn" the weights/parameters.

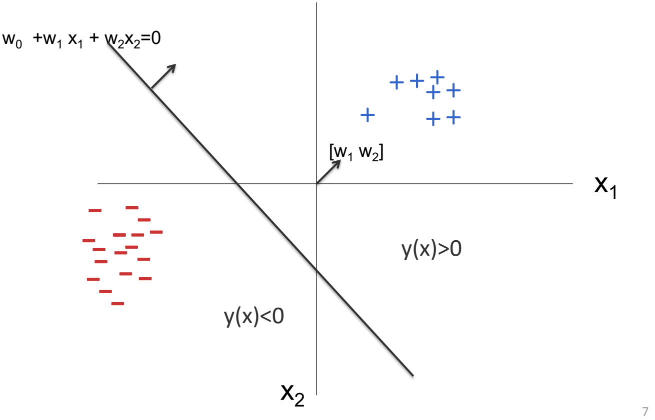
The Geometry of Linear Threshold Units

- In 2-d space, $y(\mathbf{x}) = \operatorname{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right)$ defines a line that separates the space.



The Geometry of Linear Threshold Units

- In 2-d space, $y(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right)$ defines a line that separates the space.



- In higher dimensional space, this corresponds to a hyperplane.

The Perceptron

REPORT NO. 85-460-1

THE PERCEPTRON

A PERCEIVING AND RECOGNIZING AUTOMATON

(PROJECT PARA)

January, 1957

Prepared by: Frank Rosenblatt

Frank Rosenblatt,
Project Engineer

Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN [1]

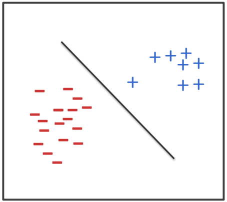
F. ROSENBLATT

Cornell Aeronautical Laboratory

The Perceptron

Assumptions

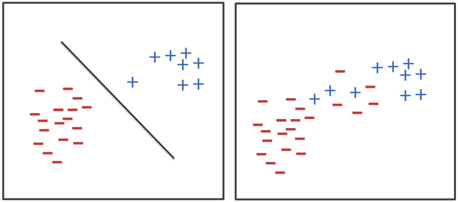
- Binary Classification, where $y \in \{-1, +1\}$.
- Data is **linearly separable**.



The Perceptron

Assumptions

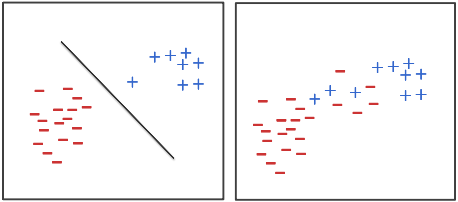
- Binary Classification, where $y \in \{-1, +1\}$.
- Data is **linearly separable**.



The Perceptron

Assumptions

- Binary Classification, where $y \in \{-1, +1\}$.
- Data is **linearly separable**.



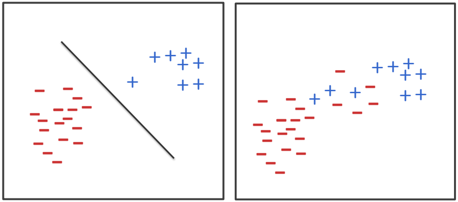
- Dummy variable:

$$\mathbf{w} = [w_0, w_1, \ldots, w_d]^\top; \quad \mathbf{x} = [1, x_1, x_2, \ldots, x_d]^\top$$

The Perceptron

Assumptions

- Binary Classification, where $y \in \{-1, +1\}$.
- Data is **linearly separable**.



- Dummy variable:

$$\mathbf{w} = [w_0, w_1, \ldots, w_d]^\top; \quad \mathbf{x} = [1, x_1, x_2, \ldots, x_d]^\top$$

$$y(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right) \Leftrightarrow y(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x}).$$

Note that we abuse the notation of $\mathbf{x}$ a little bit here.

The Perceptron Algorithm

1. Initialize $\mathbf{w} = 0$
2. For each training sample $(\mathbf{x}_i, y_i) \in \mathcal{D}$:
   - If $y_i \neq \text{sign}(\mathbf{w}^\top \mathbf{x}_i)$: `# The sample is mis-classified`
     - Update $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$ `# Update` $\mathbf{w}$ `with a single sample`
3. Return the final weight vector $\mathbf{w}$

The Perceptron Algorithm

1. Initialize $\mathbf{w} = 0$
2. For each training sample $(\mathbf{x}_i, y_i) \in \mathcal{D}$:
    - If $y_i \neq \text{sign}(\mathbf{w}^\top \mathbf{x}_i)$: `# The sample is mis-classified`
        - Update $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$ `# Update` $\mathbf{w}$ `with a single sample`
3. Return the final weight vector $\mathbf{w}$

When to stop the loop?

The Perceptron Algorithm

1 Initialize $\mathbf{w} = 0$

2 For each training sample $(\mathbf{x}_i, y_i) \in \mathcal{D}$:

- If $y_i \neq \mathrm{sign}(\mathbf{w}^\top \mathbf{x}_i)$: `# The sample is mis-classified`
    - Update $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$ `# Update` $\mathbf{w}$ `with a single sample`

3 Return the final weight vector $\mathbf{w}$

When to stop the loop? When all the samples are correctly classified.

Intuition Behind the Update

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a positive sample $(\mathbf{x}^*, y^*)$ with $y^* = +1$.

Intuition Behind the Update

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a positive sample $(\mathbf{x}^*, y^*)$ with $y^* = +1$.
  - In this case, we have

$$\text{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq +1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \leq 0$$

Intuition Behind the Update

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a positive sample $(\mathbf{x}^*, y^*)$ with $y^* = +1$.
    - In this case, we have

$$\mathrm{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq +1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \leq 0$$

- The weight after the update with this sample:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + y^* \mathbf{w}^* = \mathbf{w}_{old} + \mathbf{w}^*$$

Intuition Behind the Update

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a positive sample $(\mathbf{x}^*, y^*)$ with $y^* = +1$.
  - In this case, we have

$$\mathrm{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq +1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \leq 0$$

- The weight after the update with this sample:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + y^* \mathbf{w}^* = \mathbf{w}_{old} + \mathbf{w}^*$$

  - The new inner product is

$$\mathbf{w}_{new}^\top \mathbf{x}^* = (\mathbf{w}_{old} + \mathbf{x}^*)^\top \mathbf{x}^* = \mathbf{w}_{old}^\top \mathbf{x}^* + \mathbf{x}^{*\top} \mathbf{x}^* \geq \mathbf{w}_{old}^\top \mathbf{x}^*$$

Intuition Behind the Update

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a positive sample $(\mathbf{x}^*, y^*)$ with $y^* = +1$.
  - In this case, we have

$$\text{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq +1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \leq 0$$

- The weight after the update with this sample:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + y^* \mathbf{w}^* = \mathbf{w}_{old} + \mathbf{w}^*$$

  - The new inner product is

$$\mathbf{w}_{new}^\top \mathbf{x}^* = (\mathbf{w}_{old} + \mathbf{x}^*)^\top \mathbf{x}^* = \mathbf{w}_{old}^\top \mathbf{x}^* + \mathbf{x}^{*\top} \mathbf{x}^* \geq \mathbf{w}_{old}^\top \mathbf{x}^*$$

  - For a positive example, the Perceptron update will increase the score assigned to the same input

Intuition Behind the Update

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a positive sample $(\mathbf{x}^*, y^*)$ with $y^* = +1$.
  - In this case, we have

$$\text{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq +1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \leq 0$$

- The weight after the update with this sample:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + y^* \mathbf{w}^* = \mathbf{w}_{old} + \mathbf{w}^*$$

  - The new inner product is

$$\mathbf{w}_{new}^\top \mathbf{x}^* = (\mathbf{w}_{old} + \mathbf{x}^*)^\top \mathbf{x}^* = \mathbf{w}_{old}^\top \mathbf{x}^* + \mathbf{x}^{*\top} \mathbf{x}^* \geq \mathbf{w}_{old}^\top \mathbf{x}^*$$

  - For a positive example, the Perceptron update will increase the score assigned to the same input
  - (Assuming, we always update with $\mathbf{x}^*$.) After finite number of updates, the model can classify $\mathbf{x}^*$ correctly

Intuition Behind the Update

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a positive sample $(\mathbf{x}^*, y^*)$ with $y^* = +1$.
  - In this case, we have

$$\text{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq +1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \leq 0$$

- The weight after the update with this sample:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + y^* \mathbf{w}^* = \mathbf{w}_{old} + \mathbf{w}^*$$

  - The new inner product is

$$\mathbf{w}_{new}^\top \mathbf{x}^* = (\mathbf{w}_{old} + \mathbf{x}^*)^\top \mathbf{x}^* = \mathbf{w}_{old}^\top \mathbf{x}^* + \mathbf{x}^{*\top} \mathbf{x}^* \geq \mathbf{w}_{old}^\top \mathbf{x}^*$$

  - For a positive example, the Perceptron update will increase the score assigned to the same input
  - (Assuming, we always update with $\mathbf{x}^*$.) After finite number of updates, the model can classify $\mathbf{x}^*$ correctly
- Similar reasoning for negative samples

Intuition Behind the Update: Negative Sample

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a negative sample $(\mathbf{x}^*, y^*)$ with $y^* = -1$.

Intuition Behind the Update: Negative Sample

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a negative sample $(\mathbf{x}^*, y^*)$ with $y^* = -1$.
  - In this case, we have

$$\text{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq -1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \geq 0$$

Intuition Behind the Update: Negative Sample

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a negative sample $(\mathbf{x}^*, y^*)$ with $y^* = -1$.
  - In this case, we have

$$\mathrm{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq -1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \geq 0$$

- The weight after the update with this sample:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + y^* \mathbf{w}^* = \mathbf{w}_{old} - \mathbf{w}^*$$

Intuition Behind the Update: Negative Sample

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a negative sample $(\mathbf{x}^*, y^*)$ with $y^* = -1$.
  - In this case, we have

$$\text{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq -1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \geq 0$$

- The weight after the update with this sample:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + y^* \mathbf{w}^* = \mathbf{w}_{old} - \mathbf{w}^*$$

  - The new inner product is

$$\mathbf{w}_{new}^\top \mathbf{x}^* = (\mathbf{w}_{old} - \mathbf{x}^*)^\top \mathbf{x}^* = \mathbf{w}_{old}^\top \mathbf{x}^* - \mathbf{x}^{*\top} \mathbf{x}^* \leq \mathbf{w}_{old}^\top \mathbf{x}^*$$

Intuition Behind the Update: Negative Sample

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a negative sample $(\mathbf{x}^*, y^*)$ with $y^* = -1$.
  - In this case, we have

$$\text{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq -1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \geq 0$$

- The weight after the update with this sample:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + y^* \mathbf{w}^* = \mathbf{w}_{old} - \mathbf{w}^*$$

  - The new inner product is

$$\mathbf{w}_{new}^\top \mathbf{x}^* = (\mathbf{w}_{old} - \mathbf{x}^*)^\top \mathbf{x}^* = \mathbf{w}_{old}^\top \mathbf{x}^* - \mathbf{x}^{*\top} \mathbf{x}^* \leq \mathbf{w}_{old}^\top \mathbf{x}^*$$

  - For a negative example, the Perceptron update will decrease the score assigned to the same input

Intuition Behind the Update: Negative Sample

- Consider the model with current parameter $\mathbf{w}_{old}$ mis-classified a negative sample $(\mathbf{x}^*, y^*)$ with $y^* = -1$.
    - In this case, we have

$$\mathrm{sign}(\mathbf{w}_{old}^\top \mathbf{x}^*) \neq -1 \Leftrightarrow \mathbf{w}_{old}^\top \mathbf{x}^* \geq 0$$

- The weight after the update with this sample:

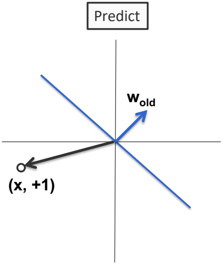
$$\mathbf{w}_{new} = \mathbf{w}_{old} + y^* \mathbf{w}^* = \mathbf{w}_{old} - \mathbf{w}^*$$

- The new inner product is

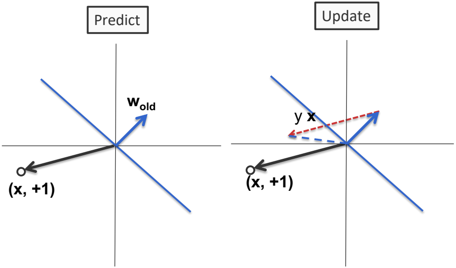
$$\mathbf{w}_{new}^\top \mathbf{x}^* = (\mathbf{w}_{old} - \mathbf{x}^*)^\top \mathbf{x}^* = \mathbf{w}_{old}^\top \mathbf{x}^* - \mathbf{x}^{*\top} \mathbf{x}^* \leq \mathbf{w}_{old}^\top \mathbf{x}^*$$

- For a negative example, the Perceptron update will decrease the score assigned to the same input
- (Assuming, we always update with $\mathbf{x}^*$.) After finite number of updates, the model can classify $\mathbf{x}^*$ correctly

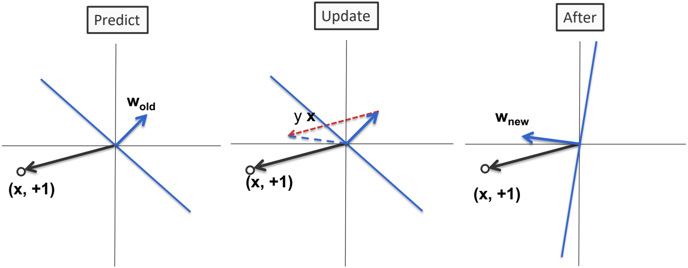
Geometry of the Perceptron Update



Geometry of the Perceptron Update

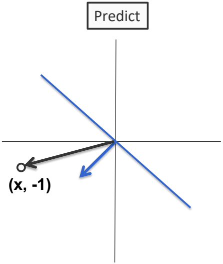


Geometry of the Perceptron Update

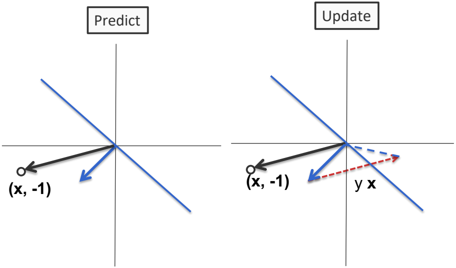


Update for a mis-classified positive sample

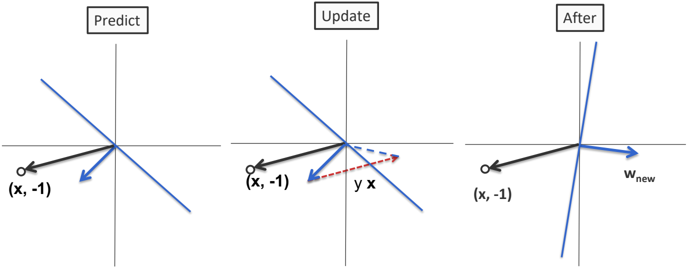
Geometry of the Perceptron Update



Geometry of the Perceptron Update



Geometry of the Perceptron Update



Update for a mis-classified negative sample

Theory

- If the training data is **linear separable**, then after some finite number of steps, the Perceptron algorithm will find a linear separator.
  - Novikoff, Albert B. On convergence proofs for perceptrons. STANFORD RESEARCH INST MENLO PARK CA, 1963.

Theory

- If the training data is **linear separable**, then after some finite number of steps, the Perceptron algorithm will find a linear separator.
  - Novikoff, Albert B. On convergence proofs for perceptrons. STANFORD RESEARCH INST MENLO PARK CA, 1963.
- If the training data is **not linearly separable**, then the learning algorithm will eventually repeat the same set of weights and enter an infinite loop

Section 3

Logistic Regression

Predicting a Probability

Will someone have a heart attack over the next year?

age	62 years
gender	male
blood suger	120 mg/DL40,000
HDL	50
LDL	120
Mass	190 lbs
Height	5' 10"
...	...

Predicting a Probability

Will someone have a heart attack over the next year?

age	62 years
gender	male
blood suger	120 mg/DL40,000
HDL	50
LDL	120
Mass	190 lbs
Height	5' 10"
...	...

- Binary Classification: Two classes, i.e, $K = 2$
- For convenience, instead of $y \in \{0, 1\}$, sometimes, we use $y \in \{-1, +1\}$ to denote the two classes.

Predicting a Probability

Will someone have a heart attack over the next year?

age	62 years
gender	male
blood suger	120 mg/DL40,000
HDL	50
LDL	120
Mass	190 lbs
Height	5' 10"
...	...

- Binary Classification: Two classes, i.e, $K = 2$
- For convenience, instead of $y \in \{0, 1\}$, sometimes, we use $y \in \{-1, +1\}$ to denote the two classes.
- In addition to Yes/No, we also care about the probability of heart attack : $p(y|x)$.

Generative Model Vs. Discriminative Model

- Probabilistic Generative Model
  Model the class-conditional densities $p(\mathbf{x}|y=k)$ and prior probabilities $p(y=k)$, and compute $p(y=k|\mathbf{x})$ using Bayes' theorem

$$p(y=k|\mathbf{x}) = \frac{p(\mathbf{x}|y=k)p(y=k)}{p(\mathbf{x})}$$

Generative Model Vs. Discriminative Model

- Probabilistic Generative Model
  Model the class-conditional densities $p(\mathbf{x}|y=k)$ and prior probabilities $p(y=k)$, and compute $p(y=k|\mathbf{x})$ using Bayes' theorem

$$p(y=k|\mathbf{x}) = \frac{p(\mathbf{x}|y=k)p(y=k)}{p(\mathbf{x})}$$

- Probabilistic Discriminative Model
  Maximize a likelihood function defined through the conditional distribution $p(y=k|\mathbf{x})$

Generative Model Vs. Discriminative Model

- Probabilistic Generative Model
  Model the class-conditional densities $p(\mathbf{x}|y=k)$ and prior probabilities $p(y=k)$, and compute $p(y=k|\mathbf{x})$ using Bayes' theorem

$$p(y=k|\mathbf{x}) = \frac{p(\mathbf{x}|y=k)p(y=k)}{p(\mathbf{x})}$$

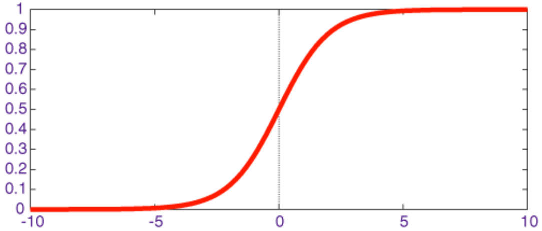
- Probabilistic Discriminative Model
  Maximize a likelihood function defined through the conditional distribution $p(y=k|\mathbf{x})$
  - Logistic Regression:

$$p(y=1|\mathbf{x}) = h(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} w_i x_i\right) = \sigma(\mathbf{w}^T\mathbf{x}).$$

Sigmoid Function σ

Logistic regression, $h(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} w_i x_i\right) = \sigma(\mathbf{w}^T\mathbf{x})$.

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

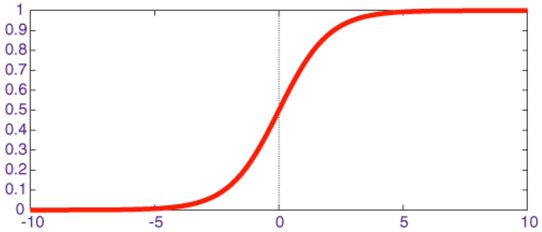


How is $\sigma(-s)$ related to $\sigma(s)$?

Sigmoid Function σ

Logistic regression, $h(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} w_i x_i\right) = \sigma(\mathbf{w}^T\mathbf{x})$.

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



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

What Makes an h Good?

$$p(y=1|\mathbf{x}) = h(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} w_i x_i\right) = \sigma(\mathbf{w}^T\mathbf{x}).$$

$$h \text{ is good if: } \begin{cases} h(\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n) \approx 1 & \text{whenever } y_n = +1; \\ h(\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n) \approx 0 & \text{whenever } y_n = -1. \end{cases}$$

What Makes an h Good?

$$p(y=1|\mathbf{x}) = h(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} w_i x_i\right) = \sigma(\mathbf{w}^T\mathbf{x}).$$

h is good if: $\begin{cases} h(\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n) \approx 1 & \text{whenever } y_n = +1; \\ h(\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n) \approx 0 & \text{whenever } y_n = -1. \end{cases}$

A simple error measure that captures this:

$$E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(h(\mathbf{x}_n) - \frac{1}{2}(1+y_n)\right)^2.$$

What Makes an h Good?

$$p(y=1|\mathbf{x}) = h(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} w_i x_i\right) = \sigma(\mathbf{w}^T\mathbf{x}).$$

h is good if: $\begin{cases} h(\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n) \approx 1 & \text{whenever } y_n = +1; \\ h(\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n) \approx 0 & \text{whenever } y_n = -1. \end{cases}$

A simple error measure that captures this:

$$E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(h(\mathbf{x}_n) - \frac{1}{2}(1+y_n)\right)^2.$$

Hard to minimize! (The loss function is not guaranteed to be convex).

The Logistic Loss Function

$$E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + \exp\left(-y_n \cdot \mathbf{w}^T\mathbf{x}\right))$$

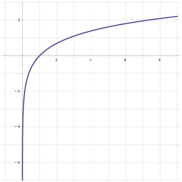
It looks complicated and ugly $(\ln, e^{(\cdot)}, \dots)$,
But,

- it is based on an intuitive probabilistic interpretation of h.
- it is very convenient and mathematically friendly ('easy' to minimize).

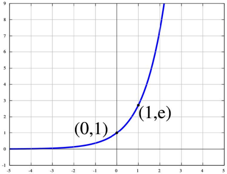
The Logistic Loss Function

$$E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n))$$

ln



exp



- For $y_n = +1$, minimizing $E(\mathbf{w})$ encourages $\mathbf{w}^T\mathbf{x}_n \gg 0$, so $h(\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n) \approx 1$
- For $y_n = -1$, minimizing $E(\mathbf{w})$ encourages $\mathbf{w}^T\mathbf{x}_n \ll 0$, so $h(\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n) \approx 0$

The Probabilistic Interpretation

Recall that we model $p(y=+1|\mathbf{x}) = h(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$,

$$P(y|\mathbf{x}) = \begin{cases} \sigma(\mathbf{w}^T\mathbf{x}) & \text{for } y=+1; \\ 1-\sigma(\mathbf{w}^T\mathbf{x}) & \text{for } y=-1. \end{cases}$$

The Probabilistic Interpretation

Recall that we model $p(y = +1|\mathbf{x}) = h(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$,

$$P(y|\mathbf{x}) = \begin{cases} \sigma(\mathbf{w}^T\mathbf{x}) & \text{for } y = +1; \\ 1 - \sigma(\mathbf{w}^T\mathbf{x}) & \text{for } y = -1. \end{cases}$$

And thus

$$P(y|\mathbf{x}) = \begin{cases} \sigma(\mathbf{w}^T\mathbf{x}) & \text{for } y = +1; \\ \sigma(-\mathbf{w}^T\mathbf{x}) & \text{for } y = -1. \end{cases}$$

The Probabilistic Interpretation

Recall that we model $p(y = +1|\mathbf{x}) = h(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$,

$$P(y|\mathbf{x}) = \begin{cases} \sigma(\mathbf{w}^T\mathbf{x}) & \text{for } y = +1; \\ 1 - \sigma(\mathbf{w}^T\mathbf{x}) & \text{for } y = -1. \end{cases}$$

And thus

$$P(y|\mathbf{x}) = \begin{cases} \sigma(\mathbf{w}^T\mathbf{x}) & \text{for } y = +1; \\ \sigma(-\mathbf{w}^T\mathbf{x}) & \text{for } y = -1. \end{cases}$$

... or, more compactly

$$P(y|\mathbf{x}) = \sigma(y \cdot \mathbf{w}^T\mathbf{x})$$

The Likelihood

$$P(y|\mathbf{x}) = \sigma(y \cdot \mathbf{w}^T \mathbf{x})$$

Assume: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ are independently generated.
Likelihood:

The Likelihood

$$P(y|\mathbf{x}) = \sigma(y \cdot \mathbf{w}^T\mathbf{x})$$

Assume: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ are independently generated.
Likelihood:

The probability of getting the $y_1, \ldots, y_N$ in $\mathcal{D}$ from the corresponding $\mathbf{x}_1, \ldots, \mathbf{x}_N$:

$$P(y_1, \ldots, y_N|\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{n=1}^{N} P(y_n|\mathbf{x}_n).$$

The Likelihood

$$P(y|\mathbf{x}) = \sigma(y \cdot \mathbf{w}^T\mathbf{x})$$

Assume: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ are independently generated.
Likelihood:

The probability of getting the $y_1, \ldots, y_N$ in $\mathcal{D}$ from the corresponding $\mathbf{x}_1, \ldots, \mathbf{x}_N$:

$$P(y_1, \ldots, y_N|\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{n=1}^{N} P(y_n|\mathbf{x}_n).$$

Maximizing the Likelihood

$$
\begin{aligned}
& \max \prod_{n=1}^{N} P(y_n|\mathbf{x}_n) \\
\Leftrightarrow & \max \ln \left(\prod_{n=1}^{N} P(y_n|\mathbf{x}_n) \right) \equiv \max \sum_{n=1}^{N} \ln P(y_n|\mathbf{x}_n) \\
\Leftrightarrow & \min -\frac{1}{N} \sum_{n=1}^{N} \ln P(y_n|\mathbf{x}_n) \equiv \min \frac{1}{N} \sum_{n=1}^{N} \ln \frac{1}{P(y_n|\mathbf{x}_n)} \\
\equiv & \min \frac{1}{N} \sum_{n=1}^{N} \ln \frac{1}{\sigma(y_n \mathbf{w}^T \mathbf{x}_n)} \\
\equiv & \min \frac{1}{N} \sum_{n=1}^{N} \ln(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n))
\end{aligned}
$$

Learning by Gradient Descent

- The gradient descent algorithm can be applied to minimize any *smooth* function, e.g.,
  - logistic regression $E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n))$
  - ridge regression $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} (\mathbf{w}^T \mathbf{x}_n - y_n)^2 + \lambda \|\mathbf{w}\|_2^2$

Learning by Gradient Descent

- The gradient descent algorithm can be applied to minimize any *smooth* function, e.g.,
  - logistic regression $E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n))$
  - ridge regression $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} (\mathbf{w}^T \mathbf{x}_n - y_n)^2 + \lambda\|\mathbf{w}\|_2^2$
- The update rule of gradient descent: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w})$

Learning by Gradient Descent

Newton
Gradient Descent
Stocasthc gradient descent

- The gradient descent algorithm can be applied to minimize any *smooth* function, e.g.,
  - logistic regression $E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n))$
  - ridge regression $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} (\mathbf{w}^T \mathbf{x}_n - y_n)^2 + \lambda \|\mathbf{w}\|_2^2$
- The update rule of gradient descent: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w})$
- When the objective function is convex, we can obtain **one** global optimal solution.

Learning by Gradient Descent

- The gradient descent algorithm can be applied to minimize any *smooth* function, e.g.,
  - logistic regression $E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n))$
  - ridge regression $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} (\mathbf{w}^T \mathbf{x}_n - y_n)^2 + \lambda \|\mathbf{w}\|_2^2$
- The update rule of gradient descent: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w})$
- When the objective function is convex, we can obtain **one** global optimal solution.
- What if the data points cannot be loaded into the memory?

Stochastic Gradient Descent (SGD)

$$E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1+\exp(-y_n \cdot \mathbf{w}^T\mathbf{x}_n)) = \frac{1}{N}\sum_{n=1}^{N} \ell(\mathbf{w}, \mathbf{x}_n, y_n)$$

- A variation of GD that considers only the error on one data point.
  - Pick a random data point $(\mathbf{x}_*, y_*)$
  - Run an iteration of GD on $\ell(\mathbf{w}, \mathbf{x}_*, y_*)$

SGD for Logistic Regression

- Gradient of $\ell(\mathbf{w}, \mathbf{x}_*, y_*)$:

$$\nabla \ell(\mathbf{w}, \mathbf{x}_*, y_*) = -y_* \mathbf{x}_* \frac{\exp\left(-y_* \mathbf{w}^T \mathbf{x}_*\right)}{1 + \exp\left(-y_* \mathbf{w}^T \mathbf{x}_*\right)} = -y_* \mathbf{x}_* \frac{1}{1 + \exp\left(+y_* \mathbf{w}^T \mathbf{x}_*\right)}$$

- The update rule

$$\mathbf{w} \leftarrow \mathbf{w} + y_* \mathbf{x}_* \left(\frac{\eta}{1 + \exp\left(+y_* \mathbf{w}^T \mathbf{x}_*\right)} \right)$$

SGD for Logistic Regression

- Gradient of $\ell(\mathbf{w}, \mathbf{x}_*, y_*)$:

$$\nabla \ell(\mathbf{w}, \mathbf{x}_*, y_*) = -y_* \mathbf{x}_* \frac{\exp\left(-y_* \mathbf{w}^T \mathbf{x}_*\right)}{1 + \exp\left(-y_* \mathbf{w}^T \mathbf{x}_*\right)} = -y_* \mathbf{x}_* \frac{1}{1 + \exp\left(+y_* \mathbf{w}^T \mathbf{x}_*\right)}$$

- The update rule

$$\mathbf{w} \leftarrow \mathbf{w} + y_* \mathbf{x}_* \left(\frac{\eta}{1 + \exp\left(+y_* \mathbf{w}^T \mathbf{x}_*\right)} \right)$$

Recall the update of Perceptron: $\mathbf{w} \leftarrow \mathbf{w} + y_* \mathbf{x}_*$

Decision Boundary of Logistic Regression

Inference: Given an unseen data $\mathbf{x}_u$, how do the learned logistic regression model assign a label to it?

- Calculate $p(y_u = +1|\mathbf{x}_u)$ and $p(y_u = -1|\mathbf{x}_u)$

Decision Boundary of Logistic Regression

Inference: Given an unseen data $\mathbf{x}_u$, how do the learned logistic regression model assign a label to it?

- Calculate $p(y_u = +1|\mathbf{x}_u)$ and $p(y_u = -1|\mathbf{x}_u)$
- Assign $y_u = +1$ if and only if $p(y_u = +1|\mathbf{x}_u) > p(y_u = -1|\mathbf{x}_u)$, other wise assign $y_u = -1$

Decision Boundary of Logistic Regression

Inference: Given an unseen data $\mathbf{x}_u$, how do the learned logistic regression model assign a label to it?

- Calculate $p(y_u = +1|\mathbf{x}_u)$ and $p(y_u = -1|\mathbf{x}_u)$
- Assign $y_u = +1$ if and only if $p(y_u = +1|\mathbf{x}_u) > p(y_u = -1|\mathbf{x}_u)$, other wise assign $y_u = -1$

Decision boundary

- Defined by the equation $p(y_u = +1|\mathbf{x}_u) = p(y_u = -1|\mathbf{x}_u)$

Decision Boundary of Logistic Regression

Inference: Given an unseen data $\mathbf{x}_u$, how do the learned logistic regression model assign a label to it?

- Calculate $p(y_u = +1|\mathbf{x}_u)$ and $p(y_u = -1|\mathbf{x}_u)$
- Assign $y_u = +1$ if and only if $p(y_u = +1|\mathbf{x}_u) > p(y_u = -1|\mathbf{x}_u)$, other wise assign $y_u = -1$

Decision boundary

- Defined by the equation $p(y_u = +1|\mathbf{x}_u) = p(y_u = -1|\mathbf{x}_u)$

$$\sigma(\mathbf{w}^\top \mathbf{x}) = \sigma(-\mathbf{w}^\top \mathbf{x}) \Leftrightarrow \mathbf{w}^\top \mathbf{x} = 0$$

Decision Boundary of Logistic Regression

Inference: Given an unseen data $\mathbf{x}_u$, how do the learned logistic regression model assign a label to it?

- Calculate $p(y_u = +1|\mathbf{x}_u)$ and $p(y_u = -1|\mathbf{x}_u)$
- Assign $y_u = +1$ if and only if $p(y_u = +1|\mathbf{x}_u) > p(y_u = -1|\mathbf{x}_u)$, other wise assign $y_u = -1$

Decision boundary

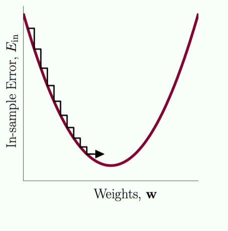
- Defined by the equation $p(y_u = +1|\mathbf{x}_u) = p(y_u = -1|\mathbf{x}_u)$

$$\sigma(\mathbf{w}^\top \mathbf{x}) = \sigma(-\mathbf{w}^\top \mathbf{x}) \Leftrightarrow \mathbf{w}^\top \mathbf{x} = 0$$

- The boundary is linear

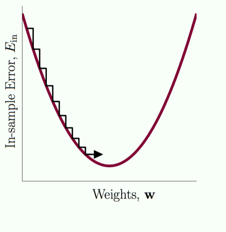
Some Additional Notes on Step Size

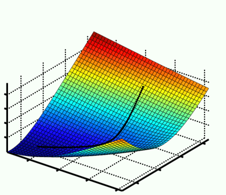
η too small



Some Additional Notes on Step Size

η too small

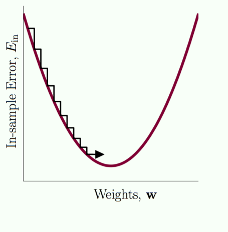




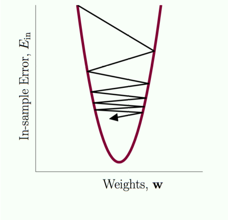
$\eta = 0.1$; 75 steps

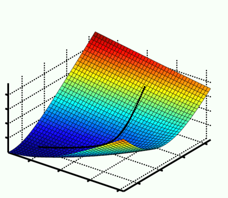
Some Additional Notes on Step Size

η too small



η too large

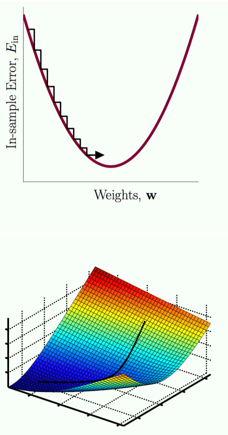




$\eta = 0.1$; 75 steps

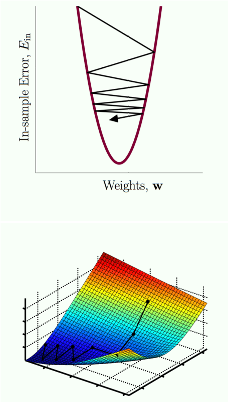
Some Additional Notes on Step Size

η too small



$\eta = 0.1$; 75 steps

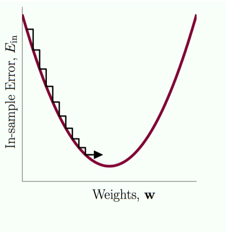
η too large

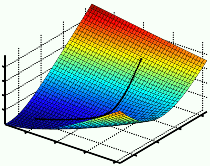


$\eta = 2$; 10 steps

Some Additional Notes on Step Size

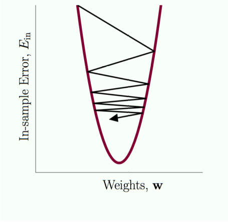
η too small

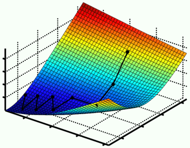




$\eta = 0.1$; 75 steps

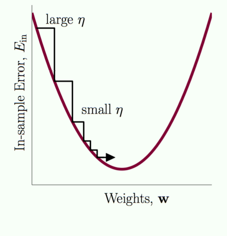
η too large





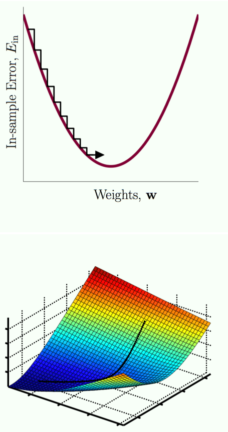
$\eta = 2$; 10 steps

η just right



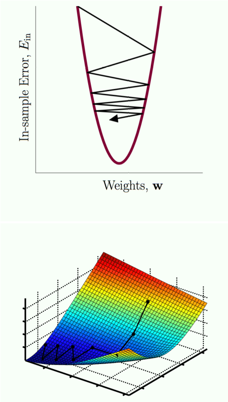
Some Additional Notes on Step Size

η too small



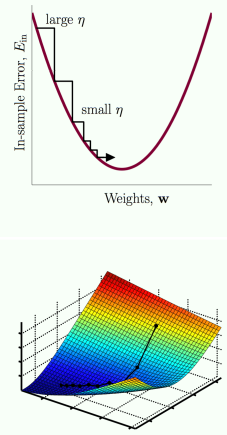
$\eta = 0.1$; 75 steps

η too large



$\eta = 2$; 10 steps

η just right



variable η_t; 10 steps