CSC 2515 (Lecture 3): Linear Classification

Raquel Urtasun & Rich Zemel

University of Toronto

Jan 19, 2015

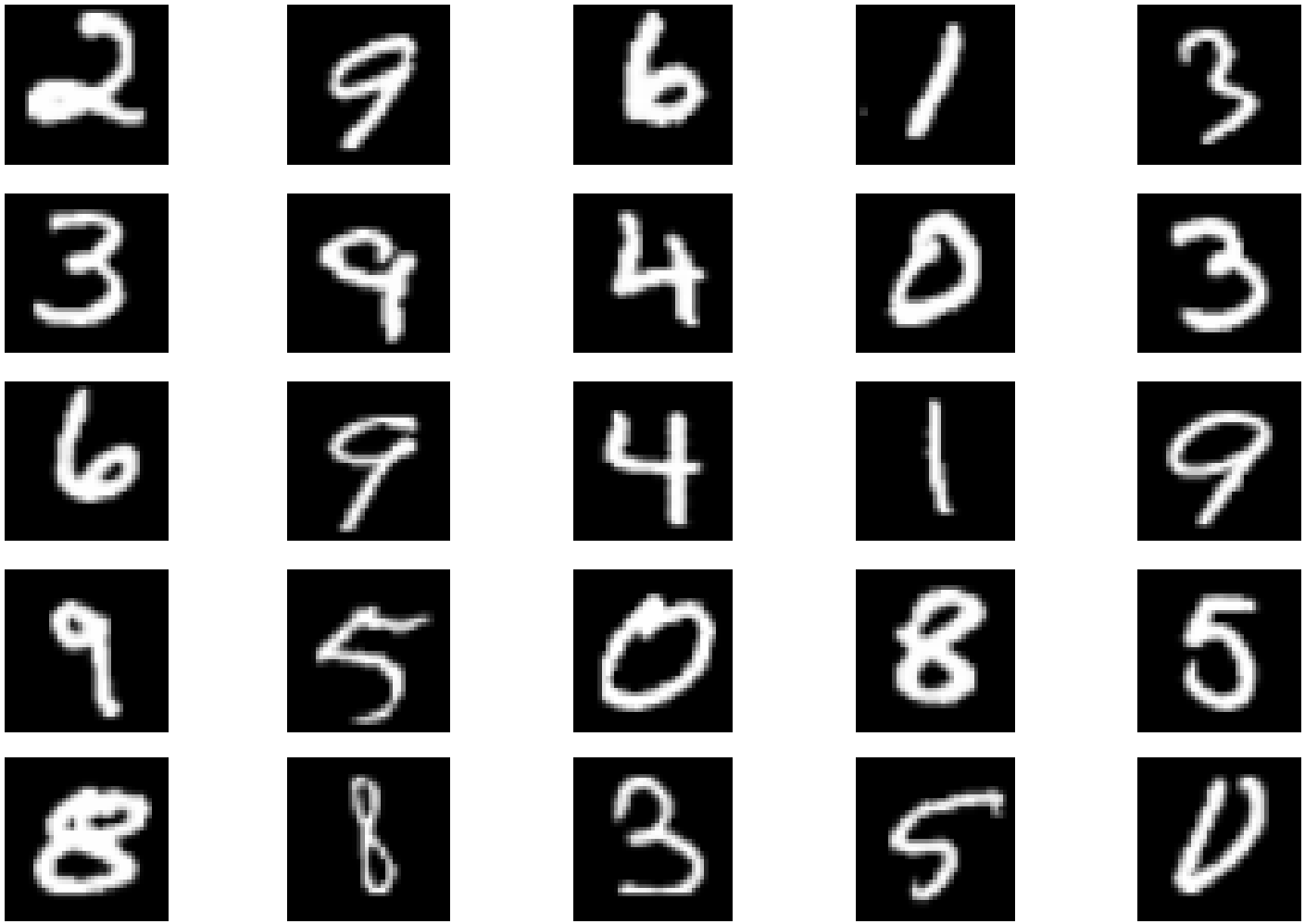
Today

- Classification as regression
- Decision boundary
- Loss functions
- Logistic Regression
- Regularization
- Nearest Neighbors

Classification vs Regression

- We are interested in mapping the input $\mathbf{x} \in \mathcal{X}$ to a *label* $t \in \mathcal{Y}$
- In regression typically $\mathcal{Y} = \Re$
- Now its a category
- Examples?

Examples of Classification



What digit is this?
How can I predict this? What are my input features?

Examples of Classification

Is this a dog?
How can I predict this? What are my input features?

Examples of Classification

what about this one?

Examples of Classification

Am I going to pass the exam?
How can I predict this? What are my input features?

Examples of Classification

Do I have diabetes?
How can I predict this? What are my input features?

Discriminative vs Generative

- **Generative approach**: Model the distribution of inputs characteristic of the class (Bayes classifier)
  - build a model of $p(\mathbf{x}|t = k)$ for every class
  - apply Bayes rule to predict the class
- **Discriminative approach:** estimate parameters of decision boundary/class separator directly from labeled examples
  - learn boundary parameters directly (e.g., logistic regression)
  - learn mappings from inputs to classes $x \rightarrow t$ (e.g., neural nets)

Classification as Regression

- Can we do this task using what we have learned in the previous lecture?

Classification as Regression

- Can we do this task using what we have learned in the previous lecture?
- Simple hack: Ignore that the input is categorical!

Classification as Regression

- Can we do this task using what we have learned in the previous lecture?
- Simple hack: Ignore that the input is categorical!
- Suppose we have a binary problem, $t \in \{-1, 1\}$

Classification as Regression

- Can we do this task using what we have learned in the previous lecture?
- Simple hack: Ignore that the input is categorical!
- Suppose we have a binary problem, $t \in \{-1, 1\}$
- Assuming the standard model used for regression

$$y = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

Classification as Regression

- Can we do this task using what we have learned in the previous lecture?
- Simple hack: Ignore that the input is categorical!
- Suppose we have a binary problem, $t \in \{-1, 1\}$
- Assuming the standard model used for regression

$$y = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

- How can we obtain $\mathbf{w}$?

Classification as Regression

- Can we do this task using what we have learned in the previous lecture?
- Simple hack: Ignore that the input is categorical!
- Suppose we have a binary problem, $t \in \{-1, 1\}$
- Assuming the standard model used for regression

$$y = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

- How can we obtain $\mathbf{w}$?
- Use least squares, $\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$. How is $\mathbf{X}$ computed? and $\mathbf{t}$?

Classification as Regression

- Can we do this task using what we have learned in the previous lecture?
- Simple hack: Ignore that the input is categorical!
- Suppose we have a binary problem, $t \in \{-1, 1\}$
- Assuming the standard model used for regression

$$y = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

- How can we obtain $\mathbf{w}$?
- Use least squares, $\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$. How is $\mathbf{X}$ computed? and $\mathbf{t}$?
- Which loss are we minimizing? Does it make sense?

$$\ell_{square}(\mathbf{w}, t) = \frac{1}{N} \sum_{i=1}^{N} (t_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Classification as Regression

- Can we do this task using what we have learned in the previous lecture?
- Simple hack: Ignore that the input is categorical!
- Suppose we have a binary problem, $t \in \{-1, 1\}$
- Assuming the standard model used for regression

$$y = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

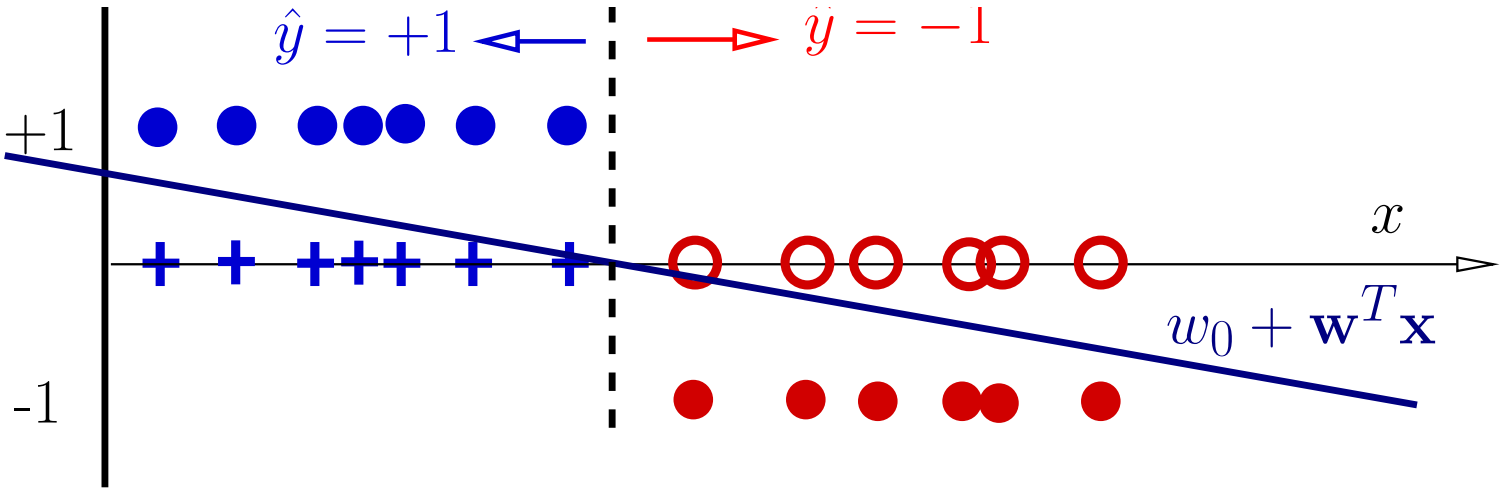
- How can we obtain $\mathbf{w}$?
- Use least squares, $\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$. How is $\mathbf{X}$ computed? and $\mathbf{t}$?
- Which loss are we minimizing? Does it make sense?

$$\ell_{square}(\mathbf{w}, t) = \frac{1}{N} \sum_{i=1}^{N} (t_i - \mathbf{w}^T \mathbf{x}_i)^2$$

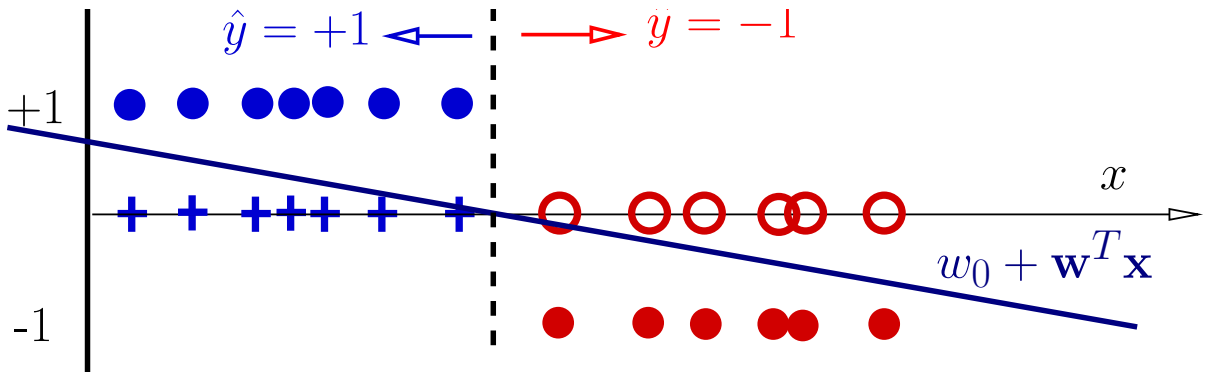
- How do I compute a label for a new example? Let's see an example

Classification as Regression

Classification as Regression



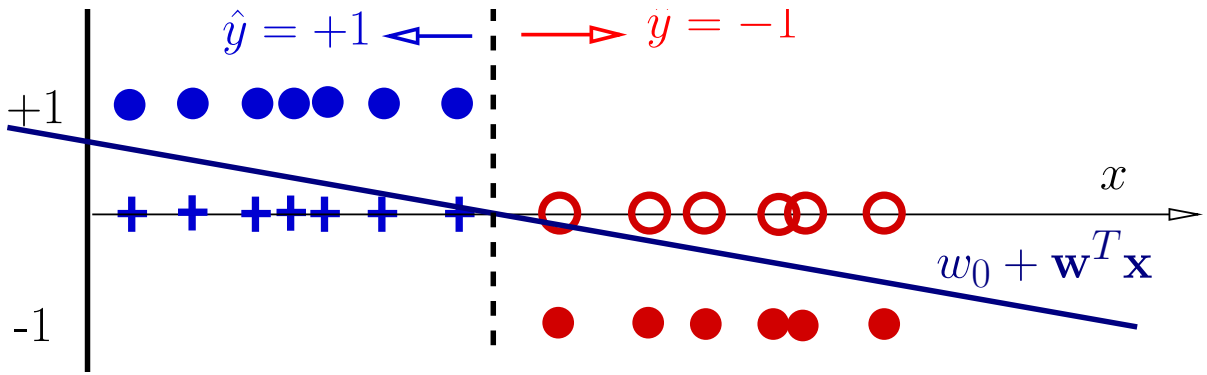
Decision Rules



- Our classifier has the form

$$f(\mathbf{x}, \mathbf{w}) = w_o + \mathbf{w}^T\mathbf{x}$$

Decision Rules



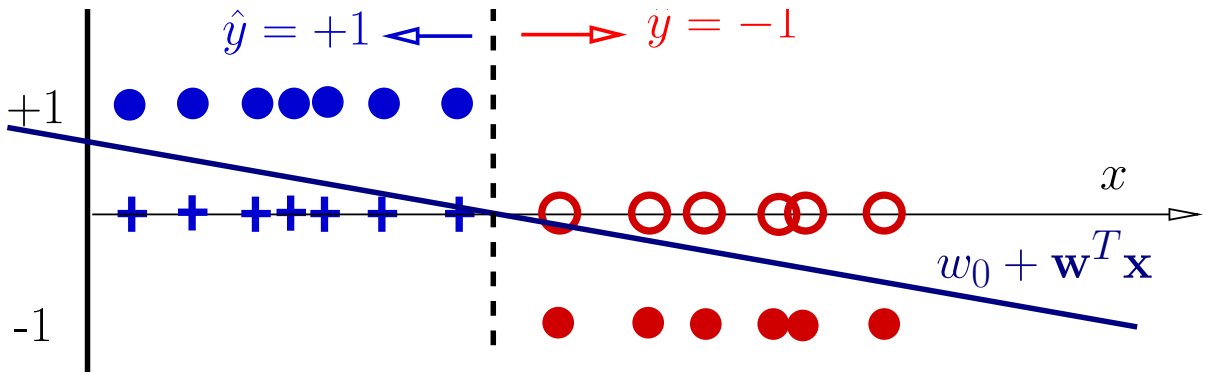
- Our classifier has the form

$$f(\mathbf{x}, \mathbf{w}) = w_o + \mathbf{w}^T\mathbf{x}$$

- A reasonable **decision rule** is

$$y = \begin{cases} 1 & \text{if } f(\mathbf{x}, \mathbf{w}) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Decision Rules



- Our classifier has the form

$$f(\mathbf{x}, \mathbf{w}) = w_o + \mathbf{w}^T\mathbf{x}$$

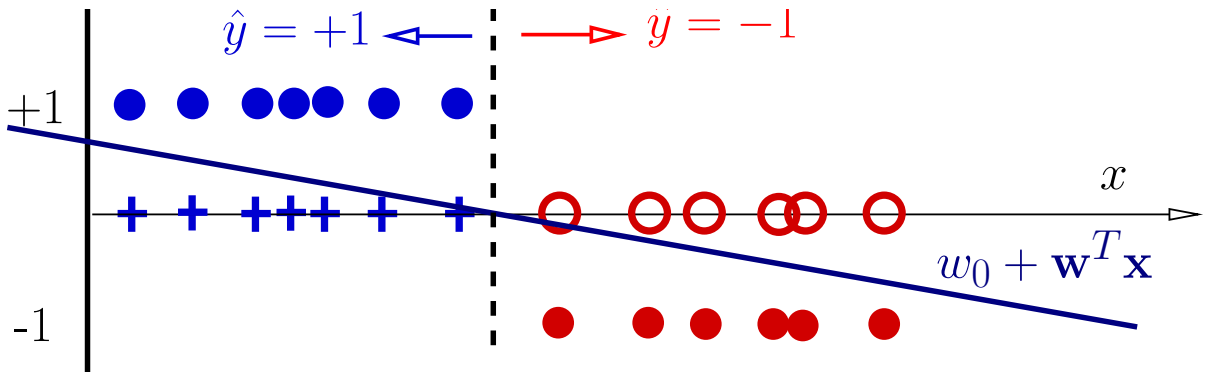
- A reasonable **decision rule** is

$$y = \begin{cases} 1 & \text{if } f(\mathbf{x}, \mathbf{w}) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- How can I mathematically write this rule?

$$y = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

Decision Rules



- Our classifier has the form

$$f(\mathbf{x}, \mathbf{w}) = w_o + \mathbf{w}^T\mathbf{x}$$

- A reasonable **decision rule** is

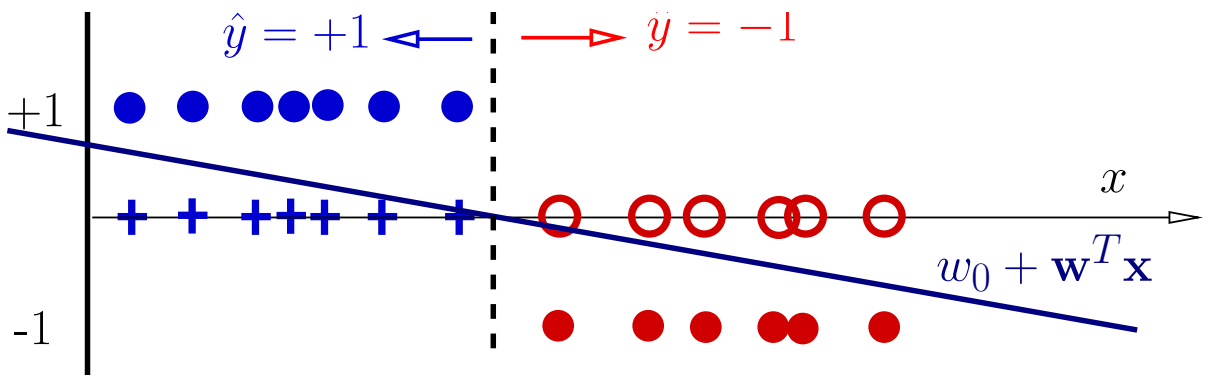
$$y = \begin{cases} 1 & \text{if } f(\mathbf{x}, \mathbf{w}) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- How can I mathematically write this rule?

$$y = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

- How does this function look like?

Decision Rules



- How can I mathematically write this rule?

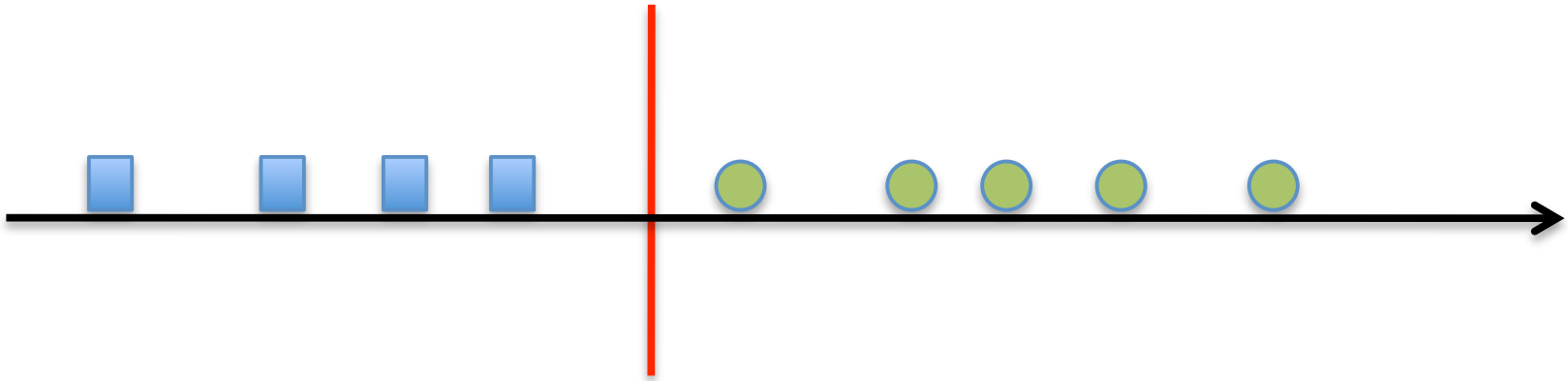
$$y = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

- This specifies a **linear classifier**: it has a **linear boundary (hyperplane)**

$$w_0 + \mathbf{w}^T\mathbf{x} = 0$$

which separates the space into two "half-spaces"

Example in 1D

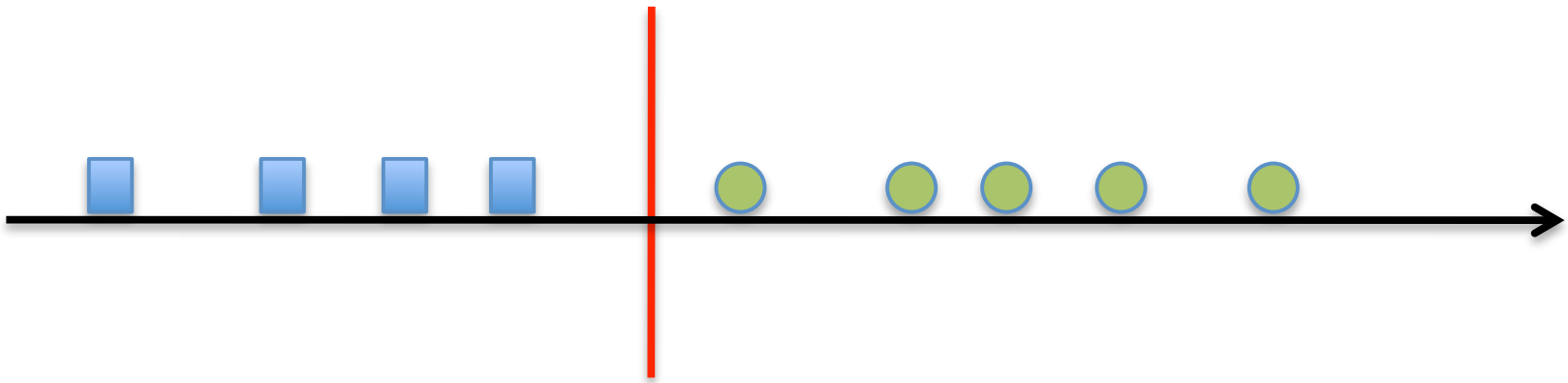


- The **linear classifier** has a **linear boundary (hyperplane)**

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

which separates the space into two "half-spaces"

Example in 1D

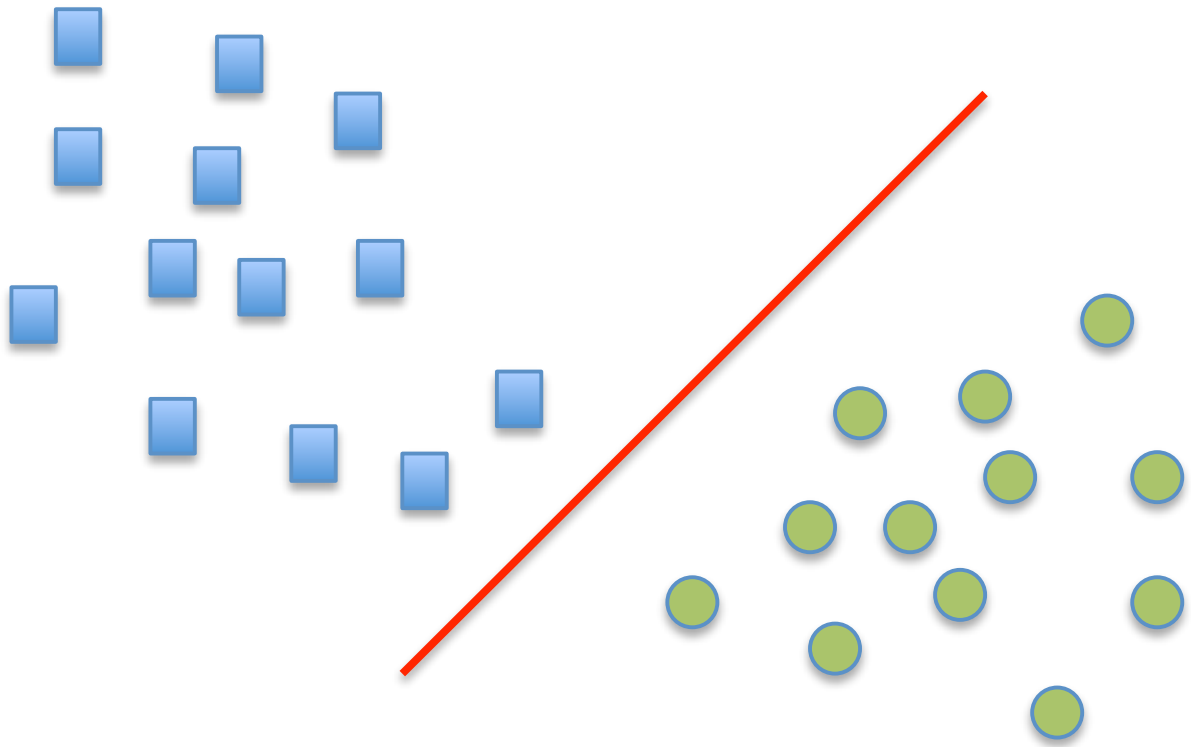


- The **linear classifier** has a **linear boundary (hyperplane)**

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

  which separates the space into two "half-spaces"
- In 1D this is simply a threshold

Example in 2D

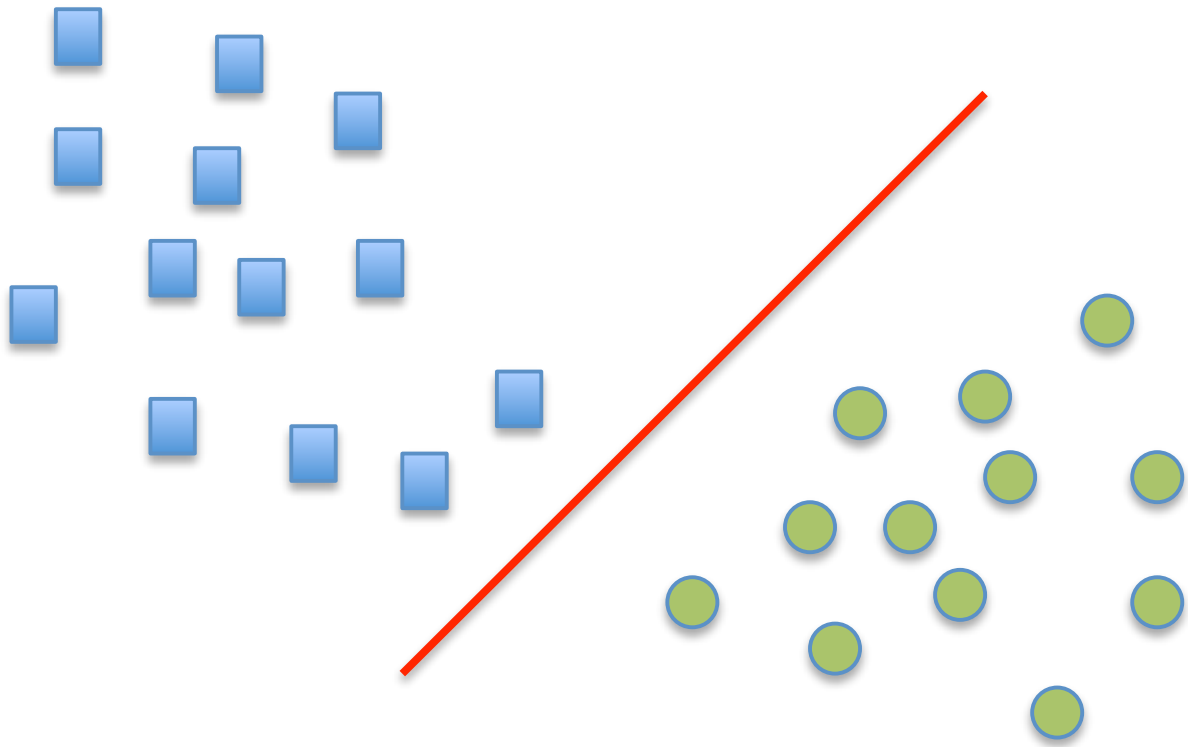


- The **linear classifier** has a **linear boundary (hyperplane)**

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

which separates the space into two "half-spaces"

Example in 2D

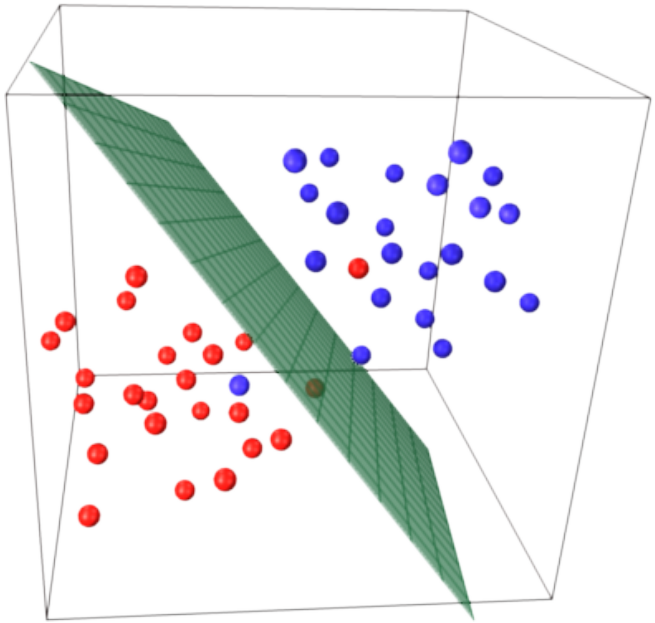


- The **linear classifier** has a **linear boundary (hyperplane)**

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

  which separates the space into two "half-spaces"
- In 2D this is a line

Example in 3D

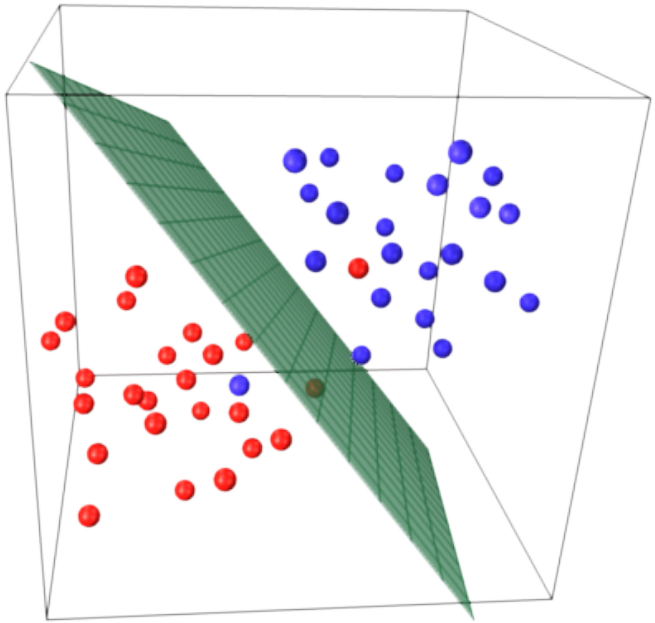


- The **linear classifier** has a **linear boundary (hyperplane)**

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

which separates the space into two "half-spaces"

Example in 3D

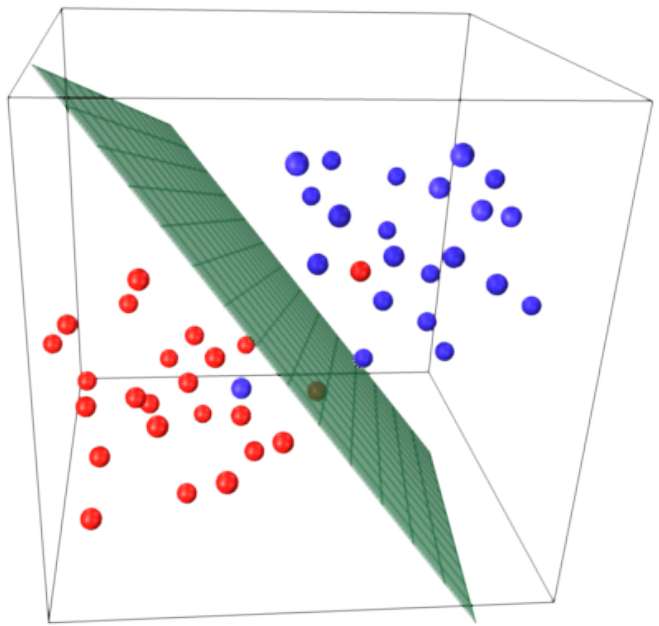


- The **linear classifier** has a **linear boundary (hyperplane)**

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

  which separates the space into two "half-spaces"
- In 3D this is a plane

Example in 3D



- The **linear classifier** has a **linear boundary (hyperplane)**

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

  which separates the space into two "half-spaces"
- In 3D this is a plane
- What about higher-dimensional spaces?

Geometry

$\mathbf{w}^T\mathbf{x} = 0$ a line passing though the origin and orthogonal to $\mathbf{w}$
$\mathbf{w}^T\mathbf{x} + w_0 = 0$ shifts it by w_0

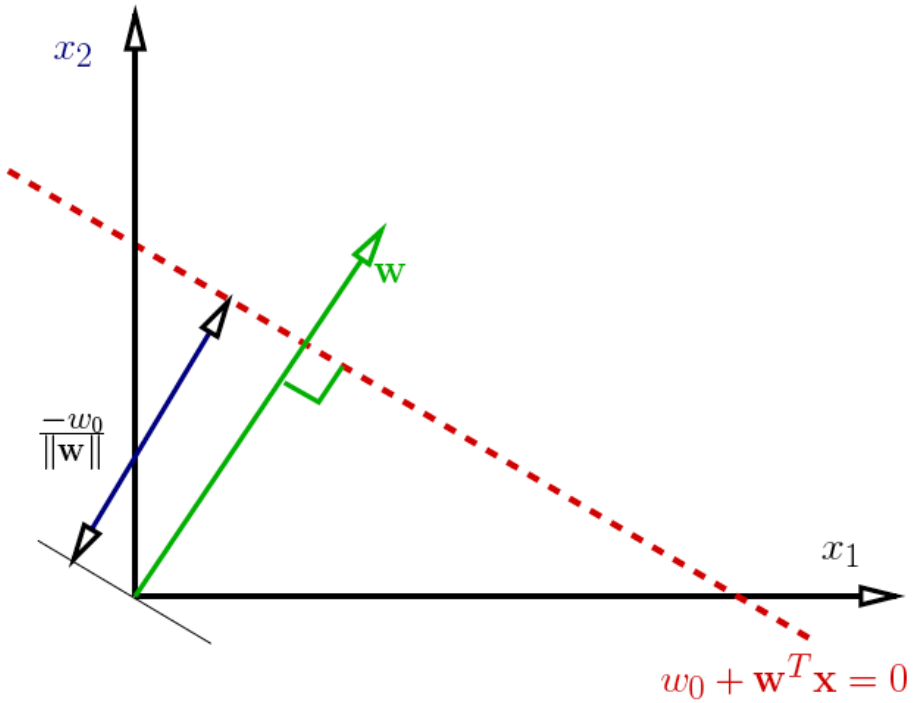


Figure from G. Shakhnarovich

Learning Linear Classifiers

- Learning consists in estimating a "good" decision boundary

Learning Linear Classifiers

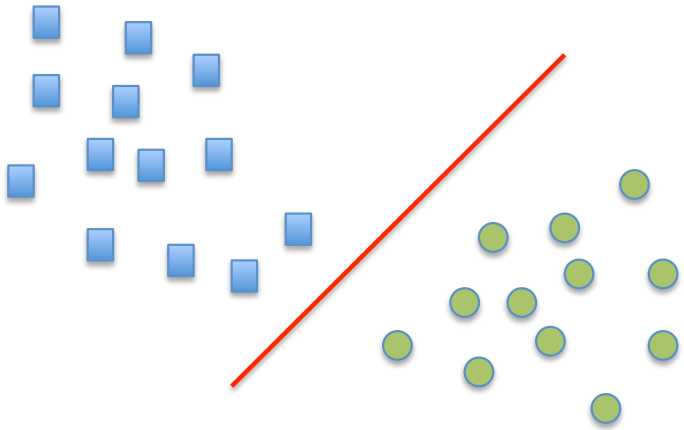
- Learning consists in estimating a "good" decision boundary
- We need to find $\mathbf{w}$ (direction) and w_0 (location) of the boundary

Learning Linear Classifiers

- Learning consists in estimating a "good" decision boundary
- We need to find $\mathbf{w}$ (direction) and w_0 (location) of the boundary
- What does "good" mean?

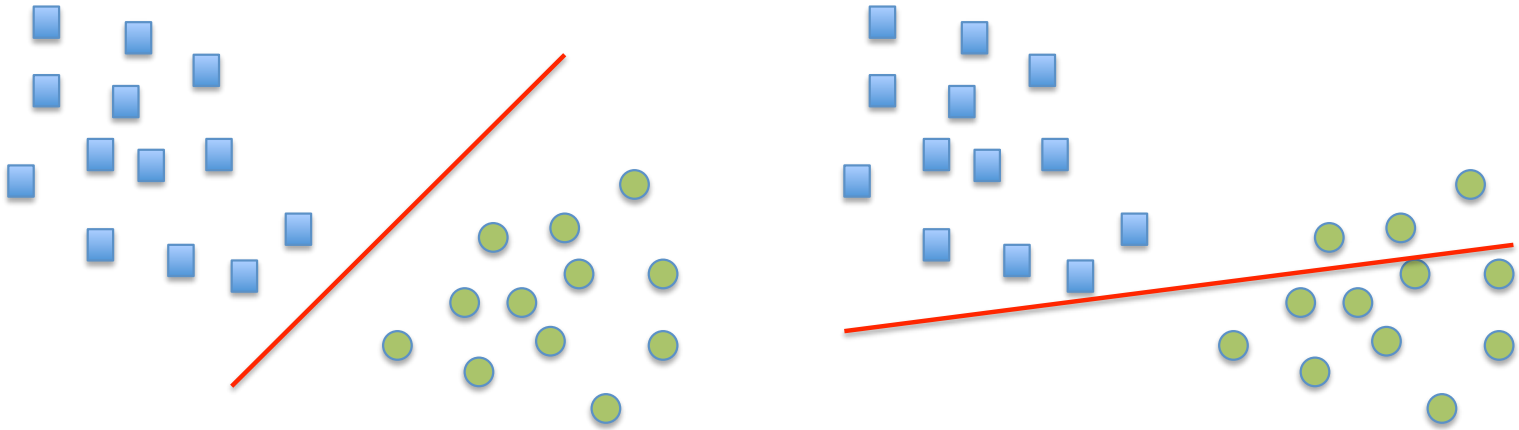
Learning Linear Classifiers

- Learning consists in estimating a "good" decision boundary
- We need to find $\mathbf{w}$ (direction) and w_0 (location) of the boundary
- What does "good" mean?
- Is this boundary good?



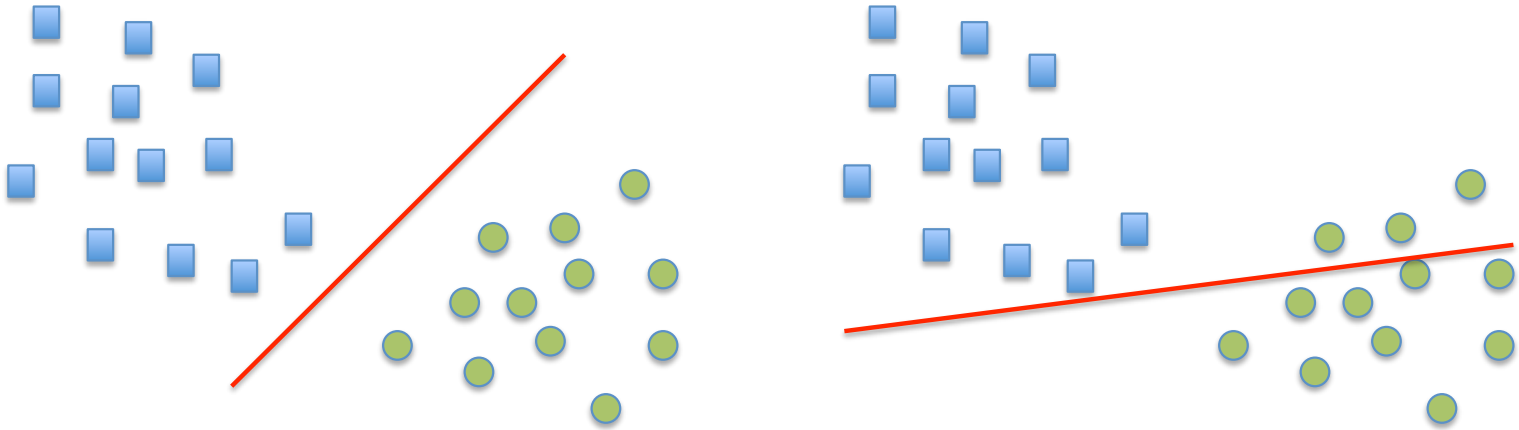
Learning Linear Classifiers

- Learning consists in estimating a "good" decision boundary
- We need to find $\mathbf{w}$ (direction) and w_0 (location) of the boundary
- What does "good" mean?
- Is this boundary good?



Learning Linear Classifiers

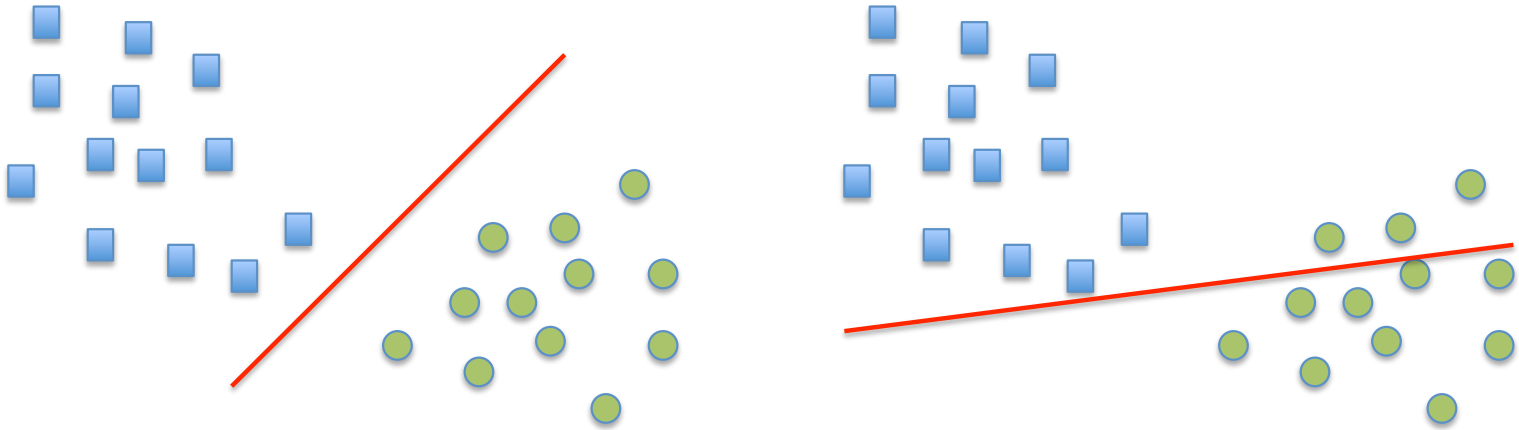
- Learning consists in estimating a "good" decision boundary
- We need to find $\mathbf{w}$ (direction) and w_0 (location) of the boundary
- What does "good" mean?
- Is this boundary good?



- We need a criteria that tell us how to select the parameters

Learning Linear Classifiers

- Learning consists in estimating a "good" decision boundary
- We need to find $\mathbf{w}$ (direction) and w_0 (location) of the boundary
- What does "good" mean?
- Is this boundary good?



- We need a criteria that tell us how to select the parameters
- Do you know any?

Loss functions

- Classifying using a linear decision boundary reduces the data dimension to 1

$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

Loss functions

- Classifying using a linear decision boundary reduces the data dimension to 1

$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

- What is the cost of being wrong?

Loss functions

- Classifying using a linear decision boundary reduces the data dimension to 1

$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

- What is the cost of being wrong?
- **Loss function:** $L(y, t)$ is the loss incurred for predicting y when correct answer is t

Loss functions

- Classifying using a linear decision boundary reduces the data dimension to 1

$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

- What is the cost of being wrong?
- **Loss function:** $L(y, t)$ is the loss incurred for predicting y when correct answer is t
- For medical diagnosis: For a diabetes screening test is it better to have false positives or false negatives?

Loss functions

- Classifying using a linear decision boundary reduces the data dimension to 1

$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

- What is the cost of being wrong?
- **Loss function:** $L(y, t)$ is the loss incurred for predicting y when correct answer is t
- For medical diagnosis: For a diabetes screening test is it better to have false positives or false negatives?
- For movie ratings: The "truth" is that Alice thinks E.T. is worthy of a 4. How bad is it to predict a 5? How about a 2?

Loss functions

- A possible loss to minimize is the zero/one loss

$$L(y(\mathbf{x}), t) = \begin{cases} 0 & \text{if } y(\mathbf{x}) = t \\ 1 & \text{if } y(\mathbf{x}) \neq t \end{cases}$$

Loss functions

- A possible loss to minimize is the zero/one loss

$$L(y(\mathbf{x}), t) = \begin{cases} 0 & \text{if } y(\mathbf{x}) = t \\ 1 & \text{if } y(\mathbf{x}) \neq t \end{cases}$$

- Is this minimization easy to do? why?

Other Loss functions

- Zero/one loss for a classifier

$$L_{0-1}(y(\mathbf{x}), t) = \begin{cases} 0 & \text{if } y(\mathbf{x}) = t \\ 1 & \text{if } y(\mathbf{x}) \neq t \end{cases}$$

Other Loss functions

- Zero/one loss for a classifier

$$L_{0-1}(y(\mathbf{x}), t) = \begin{cases} 0 & \text{if } y(\mathbf{x}) = t \\ 1 & \text{if } y(\mathbf{x}) \neq t \end{cases}$$

- Asymmetric Binary Loss

$$L_{ABL}(y(\mathbf{x}), t) = \begin{cases} \alpha & \text{if } y(\mathbf{x}) = 1 \wedge t = 0 \\ \beta & \text{if } y(\mathbf{x}) = 0 \wedge t = 1 \\ 0 & \text{if } y(\mathbf{x}) = t \end{cases}$$

Other Loss functions

- Zero/one loss for a classifier

$$L_{0-1}(y(\mathbf{x}), t) = \begin{cases} 0 & \text{if } y(\mathbf{x}) = t \\ 1 & \text{if } y(\mathbf{x}) \neq t \end{cases}$$

- Asymmetric Binary Loss

$$L_{ABL}(y(\mathbf{x}), t) = \begin{cases} \alpha & \text{if } y(\mathbf{x}) = 1 \wedge t = 0 \\ \beta & \text{if } y(\mathbf{x}) = 0 \wedge t = 1 \\ 0 & \text{if } y(\mathbf{x}) = t \end{cases}$$

- Squared (quadratic) loss

$$L_{squared}(y(\mathbf{x}), t) = (t - y(\mathbf{x}))^2$$

Other Loss functions

- Zero/one loss for a classifier

$$L_{0-1}(y(\mathbf{x}), t) = \begin{cases} 0 & \text{if } y(\mathbf{x}) = t \\ 1 & \text{if } y(\mathbf{x}) \neq t \end{cases}$$

- Asymmetric Binary Loss

$$L_{ABL}(y(\mathbf{x}), t) = \begin{cases} \alpha & \text{if } y(\mathbf{x}) = 1 \wedge t = 0 \\ \beta & \text{if } y(\mathbf{x}) = 0 \wedge t = 1 \\ 0 & \text{if } y(\mathbf{x}) = t \end{cases}$$

- Squared (quadratic) loss

$$L_{squared}(y(\mathbf{x}), t) = (t - y(\mathbf{x}))^2$$

- Absolute Error

$$L_{quadratic}(y(\mathbf{x}), t) = |t - y(\mathbf{x})|$$

More complex Loss Functions

- What if the movie predictions are used for rankings? Now the predicted ratings don't matter, just the order that they imply.

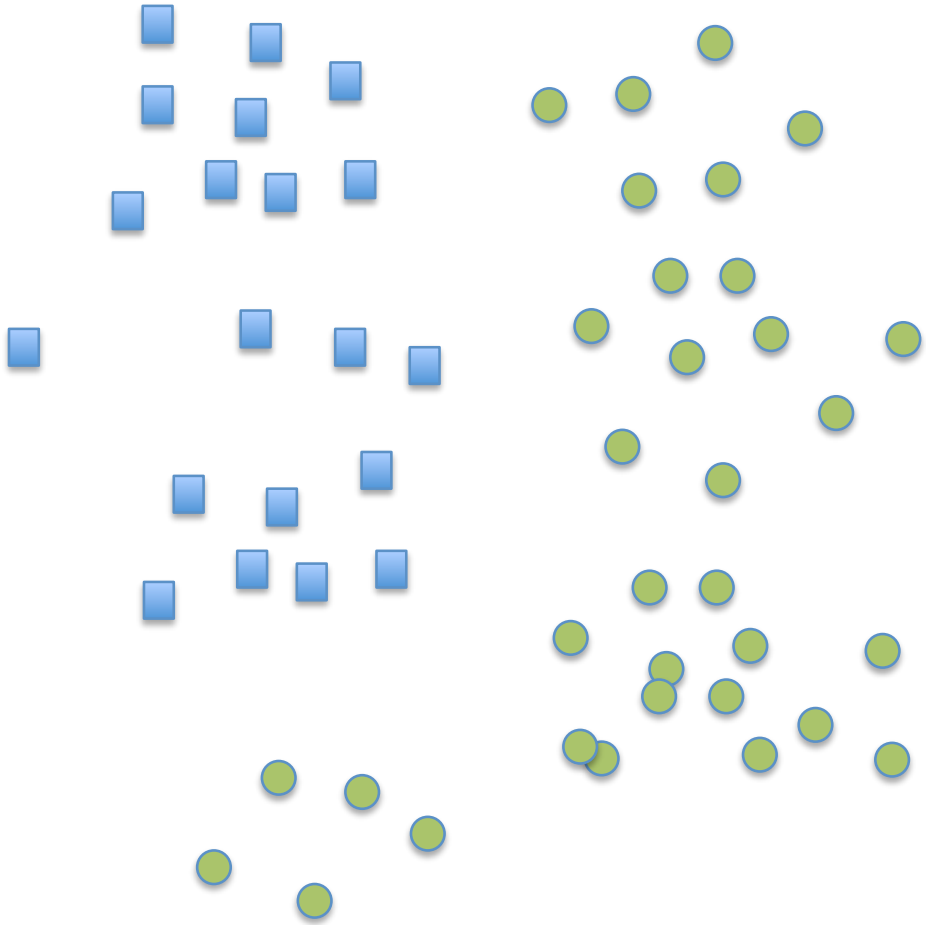
More complex Loss Functions

- What if the movie predictions are used for rankings? Now the predicted ratings don't matter, just the order that they imply.
- In what order does Alice prefer E.T., Amelie and Titanic?

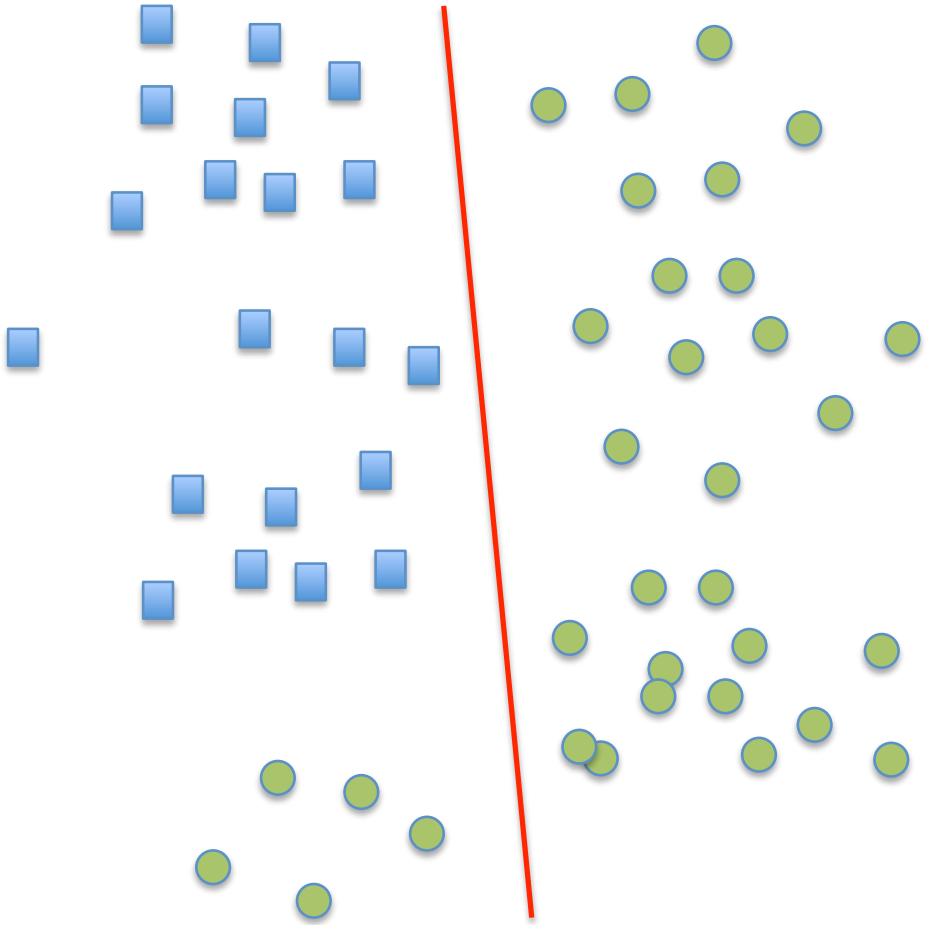
More complex Loss Functions

- What if the movie predictions are used for rankings? Now the predicted ratings don't matter, just the order that they imply.
- In what order does Alice prefer E.T., Amelie and Titanic?
- Possibilities:
  - 0-1 loss on the winner
  - Permutation distance
  - Accuracy of top K movies.

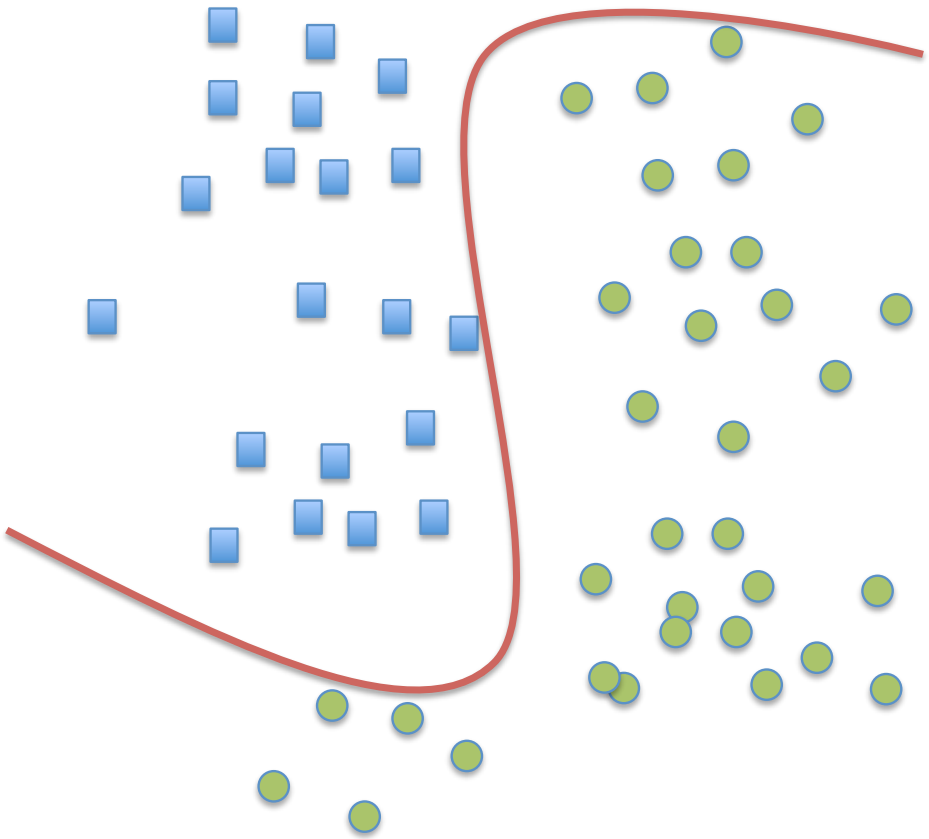
Can we always separate the classes?



Can we always separate the classes?



Can we always separate the classes?



How can we obtain a non-linear decision boundary?

Can we always separate the classes?

Causes of non perfect separation

- Model is too simple
- Noise in the inputs (i.e., data attributes)
- Simple features that do not account for all variations
- Errors in data targets (miss labelings)

Can we always separate the classes?

Causes of non perfect separation

- Model is too simple
- Noise in the inputs (i.e., data attributes)
- Simple features that do not account for all variations
- Errors in data targets (miss labelings)

Should we make the model complex enough to have perfect separation in the training data?

Linear Classifier

- The classifier we have looked at is

$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T \mathbf{x})$$

Linear Classifier

- The classifier we have looked at is

$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

- It was difficult to optimize any loss on $\ell(y, t)$ due to the form of $y(\mathbf{x})$

Linear Classifier

- The classifier we have looked at is

$$y(\mathbf{x}) = \text{sign}(w_0 + \mathbf{w}^T\mathbf{x})$$

- It was difficult to optimize any loss on $\ell(y, t)$ due to the form of $y(\mathbf{x})$
- Can we have a smoother function such that things become easier to optimize?

Logistic Regression

- An alternative: replace the $sign(\cdot)$ with the **sigmoid** or **logistic function**

Logistic Regression

- An alternative: replace the $sign(\cdot)$ with the **sigmoid** or **logistic function**
- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$$

where the sigmoid is defined as

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

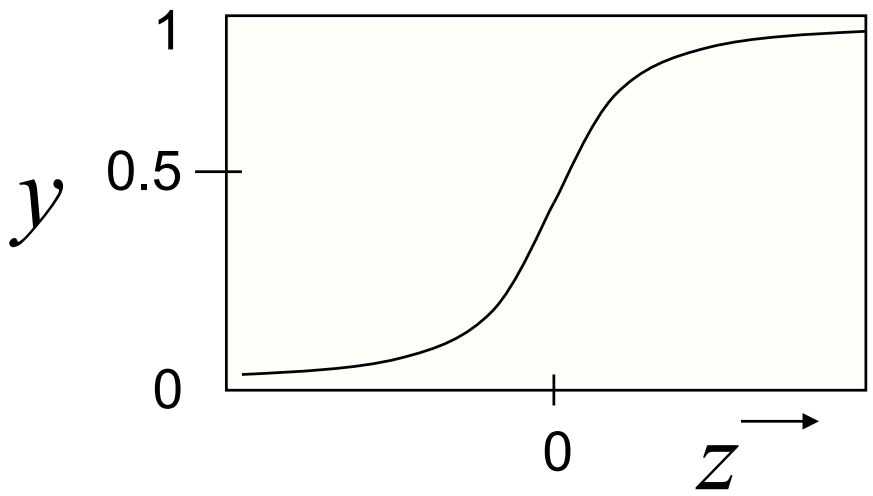
Logistic Regression

- An alternative: replace the $sign(\cdot)$ with the **sigmoid** or **logistic function**
- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$$

where the sigmoid is defined as

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



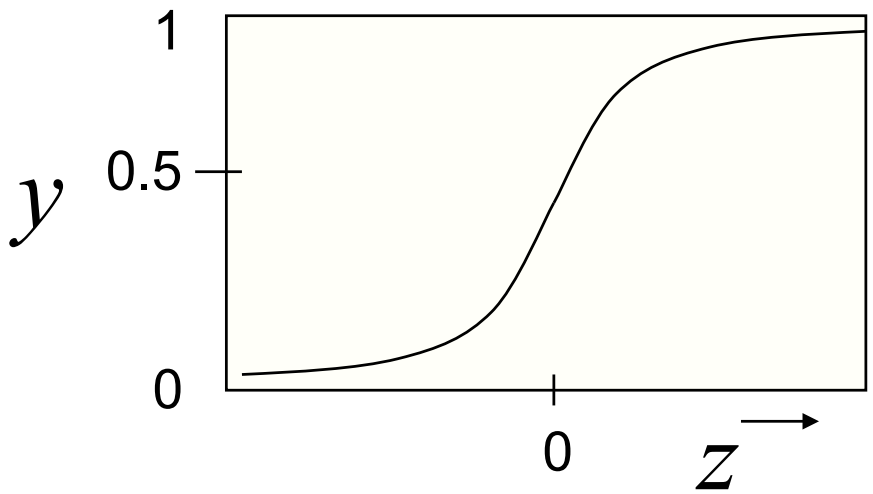
Logistic Regression

- An alternative: replace the $sign(\cdot)$ with the **sigmoid** or **logistic function**
- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$$

where the sigmoid is defined as

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



- The output is a smooth function of the inputs and the weights

Logistic Regression

- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$$

where the sigmoid is defined as

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

Logistic Regression

- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$$

where the sigmoid is defined as

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

  - One parameter per data dimension (feature)

Logistic Regression

- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$$

  where the sigmoid is defined as

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

  - One parameter per data dimension (feature)
  - Features can be discrete or continuous

Logistic Regression

- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$$

  where the sigmoid is defined as

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

  - One parameter per data dimension (feature)
  - Features can be discrete or continuous
  - Output of the model: value $y \in [0, 1]$

Logistic Regression

- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$$

  where the sigmoid is defined as

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

  - One parameter per data dimension (feature)
  - Features can be discrete or continuous
  - Output of the model: value $y \in [0, 1]$
  - This allows for gradient-based learning of the parameters: smoothed version of the $sign(\cdot)$

Shape of the Logistic Function

- Let's look at how modifying $\mathbf{w}$ changes the function shape

Shape of the Logistic Function

- Let's look at how modifying **w** changes the function shape
- 1D example:

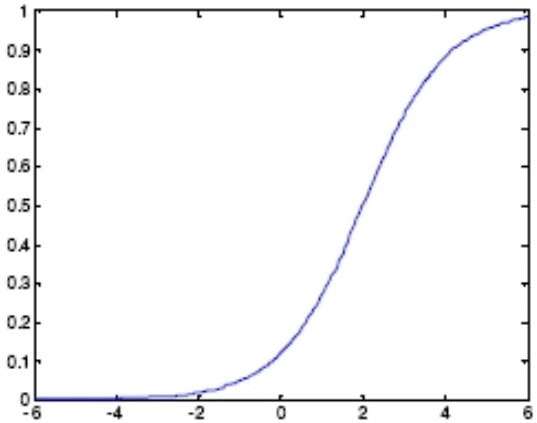
$$y = \sigma\left(w_1 x + w_0\right)$$

Shape of the Logistic Function

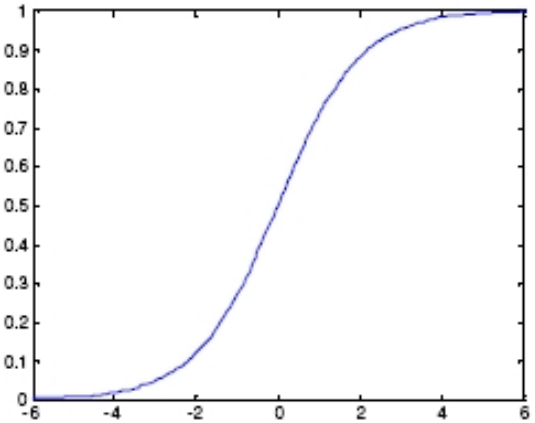
- Let's look at how modifying **w** changes the function shape
- 1D example:

$$y = \sigma\left(w_1 x + w_0\right)$$

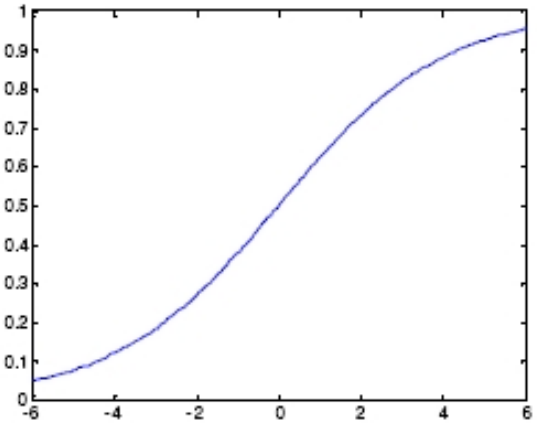
$w_0 = -2, w_1 = 1$



$w_0 = 0, w_1 = 1$



$w_0 = 0, w_1 = 0.5$



Show Matlab demo

Probabilistic Interpretation

- If we have a value between 0 and 1, let's use it to model the posterior

$$p(C = 0|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0) \quad \text{with} \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

Probabilistic Interpretation

- If we have a value between 0 and 1, let's use it to model the posterior

$$p(C = 0|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0) \quad \text{with} \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Substituting we have

$$p(C = 0|\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

Probabilistic Interpretation

- If we have a value between 0 and 1, let's use it to model the posterior

$$p(C = 0|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0) \quad \text{with} \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Substituting we have

$$p(C = 0|\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

- Supposed we have two classes, how can I compute $p(C = 1|\mathbf{x})$?

Probabilistic Interpretation

- If we have a value between 0 and 1, let's use it to model the posterior

$$p(C = 0|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0) \quad \text{with} \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Substituting we have

$$p(C = 0|\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

- Supposed we have two classes, how can I compute $p(C = 1|\mathbf{x})$?
- Use the marginalization property of probability

$$p(C = 1|\mathbf{x}) + p(C = 0|\mathbf{x}) = 1$$

Probabilistic Interpretation

- If we have a value between 0 and 1, let's use it to model the posterior

$$p(C=0|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0) \quad \text{with} \quad \sigma(z) = \frac{1}{1+\exp(-z)}$$

- Substituting we have

$$p(C=0|\mathbf{x}) = \frac{1}{1+\exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

- Supposed we have two classes, how can I compute $p(C=1|\mathbf{x})$?
- Use the marginalization property of probability

$$p(C=1|\mathbf{x}) + p(C=0|\mathbf{x}) = 1$$

- Thus (show matlab)

$$p(C=1|\mathbf{x}) = 1 - \frac{1}{1+\exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)} = \frac{\exp(-\mathbf{w}^T\mathbf{x} - w_0)}{1+\exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

Conditional likelihood

- Assume $t \in \{0, 1\}$, we can write the probability distribution of each of our training points $p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)})$

Conditional likelihood

- Assume $t \in \{0, 1\}$, we can write the probability distribution of each of our training points $p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)})$
- Assuming that the training examples are sampled IID: independent and identically distributed

$$p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) = \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)})$$

Conditional likelihood

- Assume $t \in \{0, 1\}$, we can write the probability distribution of each of our training points $p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)})$
- Assuming that the training examples are sampled IID: independent and identically distributed

$$p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) = \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)})$$

- We can write each probability as

$$p(t^{(i)} | \mathbf{x}^{(i)}) \quad = \quad p(C = 1 | \mathbf{x}^{(i)})^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}}$$

Conditional likelihood

- Assume $t \in \{0, 1\}$, we can write the probability distribution of each of our training points $p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)})$
- Assuming that the training examples are sampled IID: independent and identically distributed

$$p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) = \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)})$$

- We can write each probability as

$$\begin{aligned} p(t^{(i)} | \mathbf{x}^{(i)}) &= p(C = 1 | \mathbf{x}^{(i)})^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}} \\ &= \Big(1 - p(C = 0 | \mathbf{x}^{(i)})\Big)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}} \end{aligned}$$

Conditional likelihood

- Assume $t \in \{0, 1\}$, we can write the probability distribution of each of our training points $p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)})$
- Assuming that the training examples are sampled IID: independent and identically distributed

$$p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) = \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)})$$

- We can write each probability as

$$\begin{aligned} p(t^{(i)} | \mathbf{x}^{(i)}) &= p(C = 1 | \mathbf{x}^{(i)})^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}} \\ &= \Big(1 - p(C = 0 | \mathbf{x}^{(i)})\Big)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}} \end{aligned}$$

- We might want to learn the model, by maximizing the conditional likelihood

$$\max_{\mathbf{w}} \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)})$$

Conditional likelihood

- Assume $t \in \{0, 1\}$, we can write the probability distribution of each of our training points $p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)})$
- Assuming that the training examples are sampled IID: independent and identically distributed

$$p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) = \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)})$$

- We can write each probability as

$$\begin{aligned} p(t^{(i)} | \mathbf{x}^{(i)}) &= p(C = 1 | \mathbf{x}^{(i)})^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}} \\ &= \Big(1 - p(C = 0 | \mathbf{x}^{(i)})\Big)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}} \end{aligned}$$

- We might want to learn the model, by maximizing the conditional likelihood

$$\max_{\mathbf{w}} \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)})$$

- Convert this into a minimization so that we can write the **loss function**

Loss Function

$$p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) = \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)})$$

Loss Function

$$
\begin{aligned}
p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) &= \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)}) \\
&= \prod_{i=1}^{N} \Big(1 - p(C = 0 | \mathbf{x}^{(i)})\Big)^{t^{(i)}} \; p(C = 0 | \mathbf{x}^{(i)})^{1 - t^{(i)}}
\end{aligned}
$$

Loss Function

$$
\begin{aligned}
p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) &= \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)}) \\
&= \prod_{i=1}^{N} \left(1 - p(C = 0 | \mathbf{x}^{(i)})\right)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1-t^{(i)}}
\end{aligned}
$$

- It's convenient to take the logarithm and convert the maximization into minimization by changing the sign

$$
\ell_{log}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})
$$

Loss Function

$$
\begin{aligned}
p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) &= \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)}) \\
&= \prod_{i=1}^{N} \left(1 - p(C = 0 | \mathbf{x}^{(i)})\right)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1-t^{(i)}}
\end{aligned}
$$

- It's convenient to take the logarithm and convert the maximization into minimization by changing the sign

$$
\ell_{log}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})
$$

- Why is this equivalent to maximize the conditional likelihood?

Loss Function

$$
\begin{aligned}
p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) &= \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)}) \\
&= \prod_{i=1}^{N} \left(1 - p(C = 0 | \mathbf{x}^{(i)})\right)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1-t^{(i)}}
\end{aligned}
$$

- It's convenient to take the logarithm and convert the maximization into minimization by changing the sign

$$
\ell_{log}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})
$$

- Why is this equivalent to maximize the conditional likelihood?
- Is there a closed form solution?

Loss Function

$$
\begin{aligned}
p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}) &= \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)}) \\
&= \prod_{i=1}^{N} \left(1 - p(C = 0 | \mathbf{x}^{(i)})\right)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1-t^{(i)}}
\end{aligned}
$$

- It's convenient to take the logarithm and convert the maximization into minimization by changing the sign

$$
\ell_{log}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})
$$

- Why is this equivalent to maximize the conditional likelihood?
- Is there a closed form solution?
- It's a convex function of $\mathbf{w}$. Can we get the global optimum?

Gradient Descent

$$\min_{\mathbf{w}} \ell(\mathbf{w}) = \min_{\mathbf{w}} \left\{ -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0|\mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0|\mathbf{x}^{(i)}, \mathbf{w}) \right\}$$

Gradient Descent

$$\min_{\mathbf{w}} \ell(\mathbf{w}) = \min_{\mathbf{w}} \left\{ -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0|\mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0|\mathbf{x}^{(i)}, \mathbf{w}) \right\}$$

- Gradient descent: iterate and at each iteration compute steepest direction towards optimum, move in that direction, step-size α

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \alpha \frac{\partial \ell(\mathbf{w})}{\partial w_j}$$

Gradient Descent

$$\min_{\mathbf{w}} \ell(\mathbf{w}) = \min_{\mathbf{w}} \left\{ -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0|\mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0|\mathbf{x}^{(i)}, \mathbf{w}) \right\}$$

- Gradient descent: iterate and at each iteration compute steepest direction towards optimum, move in that direction, step-size α

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \alpha \frac{\partial \ell(\mathbf{w})}{\partial w_j}$$

- But where is $\mathbf{w}$?

Gradient Descent

$$\min_{\mathbf{w}} \ell(\mathbf{w}) = \min_{\mathbf{w}} \left\{ -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C = 0|\mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C = 0|\mathbf{x}^{(i)}, \mathbf{w}) \right\}$$

- Gradient descent: iterate and at each iteration compute steepest direction towards optimum, move in that direction, step-size α

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \alpha \frac{\partial \ell(\mathbf{w})}{\partial w_j}$$

- But where is $\mathbf{w}$?

$$p(C = 0|\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T \mathbf{x} - w_0\right)}$$

Gradient Descent

$$\min_{\mathbf{w}} \ell(\mathbf{w}) = \min_{\mathbf{w}} \left\{ -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C=0|\mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C=0|\mathbf{x}^{(i)}, \mathbf{w}) \right\}$$

- Gradient descent: iterate and at each iteration compute steepest direction towards optimum, move in that direction, step-size α

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \alpha \frac{\partial \ell(\mathbf{w})}{\partial w_j}$$

- But where is $\mathbf{w}$?

$$p(C=0|\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)} \qquad p(C=1|\mathbf{x}) = \frac{\exp(-\mathbf{w}^T\mathbf{x} - w_0)}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

Gradient Descent

$$\min_{\mathbf{w}} \ell(\mathbf{w}) = \min_{\mathbf{w}} \left\{ -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C=0|\mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C=0|\mathbf{x}^{(i)}, \mathbf{w}) \right\}$$

- Gradient descent: iterate and at each iteration compute steepest direction towards optimum, move in that direction, step-size α

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \alpha \frac{\partial \ell(\mathbf{w})}{\partial w_j}$$

- But where is $\mathbf{w}$?

$$p(C=0|\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)} \qquad p(C=1|\mathbf{x}) = \frac{\exp(-\mathbf{w}^T\mathbf{x} - w_0)}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

- You can write this in vector form

$$\bigtriangledown \ell(\mathbf{w}) = \left[\frac{\partial \ell(\mathbf{w})}{\partial w_0}, \cdots, \frac{\partial \ell(\mathbf{w})}{\partial w_k} \right]^T,$$

Gradient Descent

$$\min_{\mathbf{w}} \ell(\mathbf{w}) = \min_{\mathbf{w}} \left\{ -\sum_{i=1}^{N} t^{(i)} \log(1 - p(C=0|\mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^{N} (1 - t^{(i)}) \log p(C=0|\mathbf{x}^{(i)}, \mathbf{w}) \right\}$$

- Gradient descent: iterate and at each iteration compute steepest direction towards optimum, move in that direction, step-size α

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \alpha \frac{\partial \ell(\mathbf{w})}{\partial w_j}$$

- But where is $\mathbf{w}$?

$$p(C=0|\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)} \qquad p(C=1|\mathbf{x}) = \frac{\exp(-\mathbf{w}^T\mathbf{x} - w_0)}{1 + \exp\left(-\mathbf{w}^T\mathbf{x} - w_0\right)}$$

- You can write this in vector form

$$\nabla \ell(\mathbf{w}) = \left[\frac{\partial \ell(\mathbf{w})}{\partial w_0}, \cdots, \frac{\partial \ell(\mathbf{w})}{\partial w_k} \right]^T, \qquad \text{and} \qquad \triangle(\mathbf{w}) = -\alpha \nabla \ell(\mathbf{w})$$

Let's look at the updates

- The log likelihood is

$$\ell_{log-loss}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log p(C=1|\mathbf{x}^{(i)}, \mathbf{w}) - \sum_{i=1}^{N} (1-t^{(i)}) \log p(C=0|\mathbf{x}^{(i)}, \mathbf{w})$$

Let's look at the updates

- The log likelihood is

$$\ell_{log-loss}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log p(C=1|\mathbf{x}^{(i)}, \mathbf{w}) - \sum_{i=1}^{N} (1-t^{(i)}) \log p(C=0|\mathbf{x}^{(i)}, \mathbf{w})$$

  where the probabilities are

$$p(C=0|\mathbf{x}, \mathbf{w}) = \frac{1}{1+\exp(-z)} \qquad p(C=1|\mathbf{x}, \mathbf{w}) = \frac{\exp(-z)}{1+\exp(-z)}$$

Let's look at the updates

- The log likelihood is

$$\ell_{log-loss}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log p(C=1|\mathbf{x}^{(i)}, \mathbf{w}) - \sum_{i=1}^{N} (1-t^{(i)}) \log p(C=0|\mathbf{x}^{(i)}, \mathbf{w})$$

where the probabilities are

$$p(C=0|\mathbf{x}, \mathbf{w}) = \frac{1}{1+\exp(-z)} \qquad p(C=1|\mathbf{x}, \mathbf{w}) = \frac{\exp(-z)}{1+\exp(-z)}$$

and $z = \mathbf{w}^T\mathbf{x} + w_0$

Let's look at the updates

- The log likelihood is

$$\ell_{log-loss}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log p(C=1|\mathbf{x}^{(i)},\mathbf{w}) - \sum_{i=1}^{N}(1-t^{(i)}) \log p(C=0|\mathbf{x}^{(i)},\mathbf{w})$$

  where the probabilities are

$$p(C=0|\mathbf{x},\mathbf{w}) = \frac{1}{1+\exp(-z)} \qquad p(C=1|\mathbf{x},\mathbf{w}) = \frac{\exp(-z)}{1+\exp(-z)}$$

  and $z = \mathbf{w}^T\mathbf{x} + w_0$

- We can simplify

$$\ell(\mathbf{w}) = \sum_i t^{(i)} \log(1+\exp(-z^{(i)})) + \sum_i t^{(i)} z^{(i)} + \sum_i (1-t^{(i)}) \log(1+\exp(-z^{(i)}))$$

Let's look at the updates

- The log likelihood is

$$\ell_{log-loss}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log p(C=1|\mathbf{x}^{(i)},\mathbf{w}) - \sum_{i=1}^{N}(1-t^{(i)}) \log p(C=0|\mathbf{x}^{(i)},\mathbf{w})$$

  where the probabilities are

$$p(C=0|\mathbf{x},\mathbf{w}) = \frac{1}{1+\exp(-z)} \qquad p(C=1|\mathbf{x},\mathbf{w}) = \frac{\exp(-z)}{1+\exp(-z)}$$

  and $z = \mathbf{w}^T\mathbf{x} + w_0$

- We can simplify

$$\begin{aligned}\ell(\mathbf{w}) &= \sum_i t^{(i)} \log(1+\exp(-z^{(i)})) + \sum_i t^{(i)} z^{(i)} + \sum_i (1-t^{(i)}) \log(1+\exp(-z^{(i)})) \\ &= \sum_i \log(1+\exp(-z^{(i)})) + \sum_i t^{(i)} z^{(i)}\end{aligned}$$

Let's look at the updates

- The log likelihood is

$$\ell_{log-loss}(\mathbf{w}) = -\sum_{i=1}^{N} t^{(i)} \log p(C=1|\mathbf{x}^{(i)},\mathbf{w}) - \sum_{i=1}^{N}(1-t^{(i)}) \log p(C=0|\mathbf{x}^{(i)},\mathbf{w})$$

  where the probabilities are

$$p(C=0|\mathbf{x},\mathbf{w}) = \frac{1}{1+\exp(-z)} \qquad p(C=1|\mathbf{x},\mathbf{w}) = \frac{\exp(-z)}{1+\exp(-z)}$$

  and $z = \mathbf{w}^T\mathbf{x} + w_0$

- We can simplify

$$\begin{aligned}\ell(\mathbf{w}) &= \sum_i t^{(i)} \log(1+\exp(-z^{(i)})) + \sum_i t^{(i)} z^{(i)} + \sum_i (1-t^{(i)}) \log(1+\exp(-z^{(i)})) \\ &= \sum_i \log(1+\exp(-z^{(i)})) + \sum_i t^{(i)} z^{(i)}\end{aligned}$$

- Now it's easy to take derivatives

Updates

$$\ell(\mathbf{w}) = \sum_i t^{(i)} z^{(i)} + \sum_i \log(1 + \exp(-z^{(i)}))$$

- Now it's easy to take derivatives

Updates

$$\ell(\mathbf{w}) = \sum_i t^{(i)} z^{(i)} + \sum_i \log(1 + \exp(-z^{(i)}))$$

- Now it's easy to take derivatives
- Remember $z = \mathbf{w}^T \mathbf{x} + w_0$

Updates

$$\ell(\mathbf{w}) = \sum_i t^{(i)} z^{(i)} + \sum_i \log(1 + \exp(-z^{(i)}))$$

- Now it's easy to take derivatives
- Remember $z = \mathbf{w}^T \mathbf{x} + w_0$

$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_i t^{(i)} x_j^{(i)} - x_j^{(i)} \cdot \frac{\exp(-z^{(i)})}{1 + \exp(-z^{(i)})}$$

Updates

$$\ell(\mathbf{w}) = \sum_i t^{(i)} z^{(i)} + \sum_i \log(1 + \exp(-z^{(i)}))$$

- Now it's easy to take derivatives
- Remember $z = \mathbf{w}^T \mathbf{x} + w_0$

$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_i t^{(i)} x_j^{(i)} - x_j^{(i)} \cdot \frac{\exp(-z^{(i)})}{1 + \exp(-z^{(i)})}$$

- What's $x_j^{(i)}$?

Updates

$$\ell(\mathbf{w}) = \sum_i t^{(i)} z^{(i)} + \sum_i \log(1 + \exp(-z^{(i)}))$$

- Now it's easy to take derivatives
- Remember $z = \mathbf{w}^T \mathbf{x} + w_0$

$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_i t^{(i)} x_j^{(i)} - x_j^{(i)} \cdot \frac{\exp(-z^{(i)})}{1 + \exp(-z^{(i)})}$$

- What's $x_j^{(i)}$?
- And simplifying

$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_i x_j^{(i)} \left(t^{(i)} - p(C = 1 | \mathbf{x}^{(i)}) \right)$$

Updates

$$\ell(\mathbf{w}) = \sum_i t^{(i)} z^{(i)} + \sum_i \log(1 + \exp(-z^{(i)}))$$

- Now it's easy to take derivatives
- Remember $z = \mathbf{w}^T \mathbf{x} + w_0$

$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_i t^{(i)} x_j^{(i)} - x_j^{(i)} \cdot \frac{\exp(-z^{(i)})}{1 + \exp(-z^{(i)})}$$

- What's $x_j^{(i)}$?
- And simplifying

$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_i x_j^{(i)} \left(t^{(i)} - p(C = 1 | \mathbf{x}^{(i)}) \right)$$

- Don't get confused with indexes: j for the weight that we are updating and i for the training example

Updates

$$\ell(\mathbf{w}) = \sum_i t^{(i)} z^{(i)} + \sum_i \log(1 + \exp(-z^{(i)}))$$

- Now it's easy to take derivatives
- Remember $z = \mathbf{w}^T \mathbf{x} + w_0$

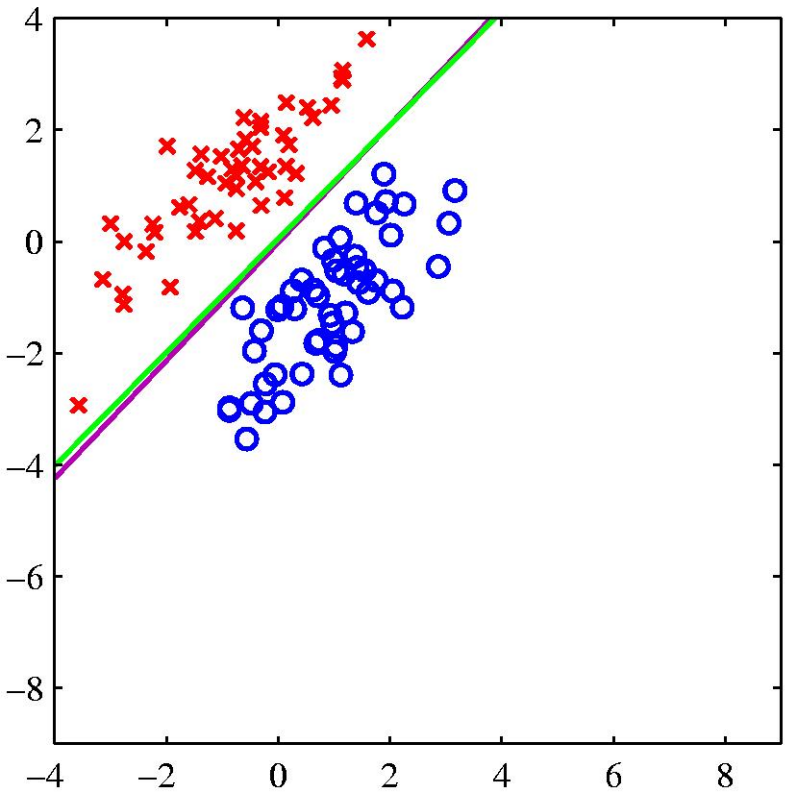
$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_i t^{(i)} x_j^{(i)} - x_j^{(i)} \cdot \frac{\exp(-z^{(i)})}{1 + \exp(-z^{(i)})}$$

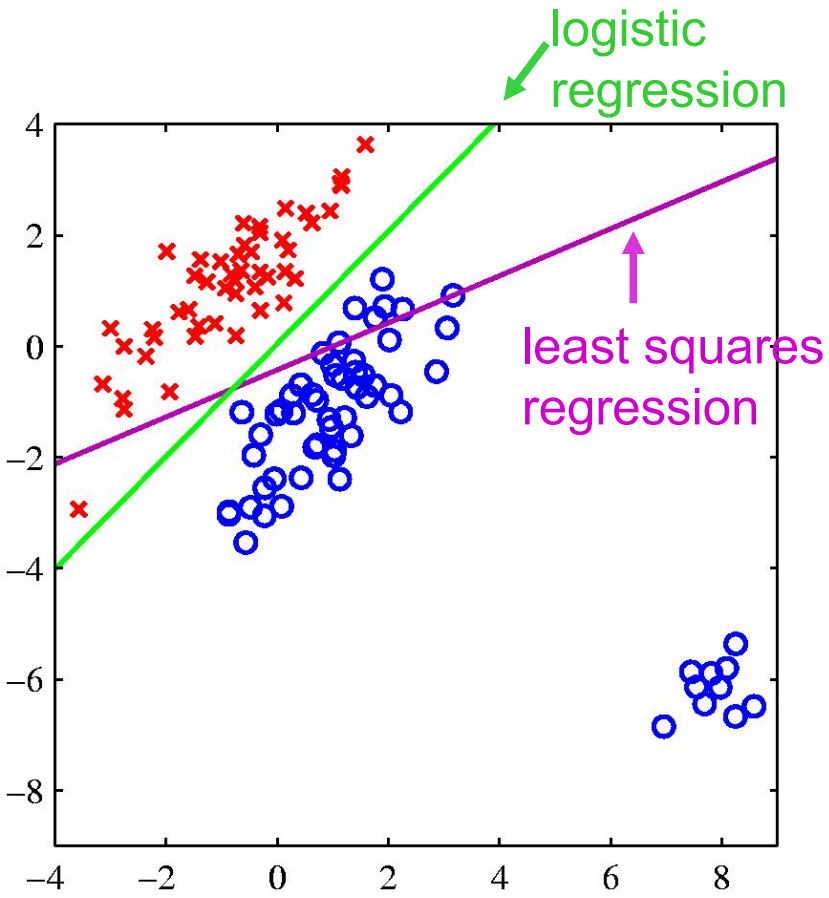
- What's $x_j^{(i)}$?
- And simplifying

$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_i x_j^{(i)} \left(t^{(i)} - p(C = 1 | \mathbf{x}^{(i)}) \right)$$

- Don't get confused with indexes: j for the weight that we are updating and i for the training example
- Logistic regression has linear decision boundary

Logistic regression vs least squares





If the right answer is 1 and the model says 1.5, it loses, so it changes the boundary to avoid being "too correct" (tilts away from outliers)

33

Regularization

- We can also look at

$$p(\mathbf{w}|\{t\},\{\mathbf{x}\}) \propto p(\{t\}|\{\mathbf{x}\},\mathbf{w})\, p(\mathbf{w})$$

with $\{t\} = (t^1, \cdots, t^N)$, and $\{\mathbf{x}\} = (\mathbf{x}^1, \cdots, \mathbf{x}^N)$

Regularization

- We can also look at

$$p(\mathbf{w}|\{t\},\{\mathbf{x}\}) \propto p(\{t\}|\{\mathbf{x}\},\mathbf{w})\, p(\mathbf{w})$$

  with $\{t\} = (t^1, \cdots, t^N)$, and $\{\mathbf{x}\} = (\mathbf{x}^1, \cdots, \mathbf{x}^N)$

- We can define priors on parameters $\mathbf{w}$

Regularization

- We can also look at

$$p(\mathbf{w}|\{t\},\{\mathbf{x}\}) \propto p(\{t\}|\{\mathbf{x}\},\mathbf{w})\, p(\mathbf{w})$$

  with $\{t\} = (t^1, \cdots, t^N)$, and $\{\mathbf{x}\} = (\mathbf{x}^1, \cdots, \mathbf{x}^N)$
- We can define priors on parameters $\mathbf{w}$
- This is a form of regularization

Regularization

- We can also look at

$$p(\mathbf{w}|\{t\},\{\mathbf{x}\}) \propto p(\{t\}|\{\mathbf{x}\},\mathbf{w})\, p(\mathbf{w})$$

  with $\{t\} = (t^1, \cdots, t^N)$, and $\{\mathbf{x}\} = (\mathbf{x}^1, \cdots, \mathbf{x}^N)$
- We can define priors on parameters $\mathbf{w}$
- This is a form of regularization
- Helps avoid large weights and **over fitting**

$$\max_{\mathbf{w}} \log \left[p(\mathbf{w}) \prod_i p(t^{(i)}|\mathbf{x}^{(i)},\mathbf{w}) \right]$$

Regularization

- We can also look at

$$p(\mathbf{w}|\{t\},\{\mathbf{x}\}) \propto p(\{t\}|\{\mathbf{x}\},\mathbf{w})\, p(\mathbf{w})$$

  with $\{t\} = (t^1, \cdots, t^N)$, and $\{\mathbf{x}\} = (\mathbf{x}^1, \cdots, \mathbf{x}^N)$

- We can define priors on parameters $\mathbf{w}$
- This is a form of regularization
- Helps avoid large weights and **over fitting**

$$\max_{\mathbf{w}} \log \left[p(\mathbf{w}) \prod_i p(t^{(i)}|\mathbf{x}^{(i)},\mathbf{w}) \right]$$

- What's $p(\mathbf{w})$?

Regularized Logistic Regression

- For example, define prior: normal distribution, zero mean and identity covariance $p(\mathbf{w}) = \mathcal{N}(0, \lambda \mathbf{I})$

Regularized Logistic Regression

- For example, define prior: normal distribution, zero mean and identity covariance $p(\mathbf{w}) = \mathcal{N}(0, \lambda \mathbf{I})$
- This prior pushes parameters towards zero

Regularized Logistic Regression

- For example, define prior: normal distribution, zero mean and identity covariance $p(\mathbf{w}) = \mathcal{N}(0, \lambda \mathbf{I})$
- This prior pushes parameters towards zero
- Including this prior the new gradient is

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \alpha \frac{\partial \ell(\mathbf{w})}{\partial w_j} - \alpha \lambda w_j^{(t)}$$

Regularized Logistic Regression

- For example, define prior: normal distribution, zero mean and identity covariance $p(\mathbf{w}) = \mathcal{N}(0, \lambda \mathbf{I})$
- This prior pushes parameters towards zero
- Including this prior the new gradient is

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \alpha \frac{\partial \ell(\mathbf{w})}{\partial w_j} - \alpha \lambda w_j^{(t)}$$

- How do we decide the best value of λ?

Use of Validation Set

- We can divide the set of training examples into two disjoint sets: training and validation

Use of Validation Set

- We can divide the set of training examples into two disjoint sets: training and validation
- Use the first set (i.e., training) to estimate the weights $\mathbf{w}$ for different values of λ

Use of Validation Set

- We can divide the set of training examples into two disjoint sets: training and validation
- Use the first set (i.e., training) to estimate the weights $\mathbf{w}$ for different values of λ
- Use the second set (i.e., validation) to estimate the best λ, by evaluating how well the classifier does in this second set

Use of Validation Set

- We can divide the set of training examples into two disjoint sets: training and validation
- Use the first set (i.e., training) to estimate the weights $\mathbf{w}$ for different values of λ
- Use the second set (i.e., validation) to estimate the best λ, by evaluating how well the classifier does in this second set
- This test how well you generalized to unseen data

Use of Validation Set

- We can divide the set of training examples into two disjoint sets: training and validation
- Use the first set (i.e., training) to estimate the weights $\mathbf{w}$ for different values of λ
- Use the second set (i.e., validation) to estimate the best λ, by evaluating how well the classifier does in this second set
- This test how well you generalized to unseen data
- The parameter λ is the importance of the regularization, and it's a **hyper parameter**

Cross-Validation

- **Leave-p-out cross-validation:**

Cross-Validation

- **Leave-p-out cross-validation:**
  - We use p observations as the validation set and the remaining observations as the training set.

Cross-Validation

- **Leave-p-out cross-validation:**
  - We use p observations as the validation set and the remaining observations as the training set.
  - This is repeated on all ways to cut the original training set.

Cross-Validation

- **Leave-p-out cross-validation:**
  - We use p observations as the validation set and the remaining observations as the training set.
  - This is repeated on all ways to cut the original training set.
  - It requires $\mathcal{C}_n^p$ for a set of n examples

Cross-Validation

- **Leave-p-out cross-validation:**
  - We use p observations as the validation set and the remaining observations as the training set.
  - This is repeated on all ways to cut the original training set.
  - It requires $\mathcal{C}_n^p$ for a set of n examples
- **Leave-1-out cross-validation:** When $p = 1$, does not have this problem

Cross-Validation

- **Leave-p-out cross-validation:**
  - We use p observations as the validation set and the remaining observations as the training set.
  - This is repeated on all ways to cut the original training set.
  - It requires $\mathcal{C}_n^p$ for a set of n examples
- **Leave-1-out cross-validation:** When $p = 1$, does not have this problem
- **k-fold cross-validation:**

Cross-Validation

- **Leave-p-out cross-validation:**
  - We use p observations as the validation set and the remaining observations as the training set.
  - This is repeated on all ways to cut the original training set.
  - It requires $\mathcal{C}_n^p$ for a set of n examples
- **Leave-1-out cross-validation:** When $p = 1$, does not have this problem
- **k-fold cross-validation:**
  - The training set is randomly partitioned into k equal size subsamples.

Cross-Validation

- **Leave-p-out cross-validation:**
  - We use p observations as the validation set and the remaining observations as the training set.
  - This is repeated on all ways to cut the original training set.
  - It requires $\mathcal{C}_n^p$ for a set of n examples
- **Leave-1-out cross-validation:** When $p = 1$, does not have this problem
- **k-fold cross-validation:**
  - The training set is randomly partitioned into k equal size subsamples.
  - Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data.

Cross-Validation

- **Leave-p-out cross-validation:**
  - We use p observations as the validation set and the remaining observations as the training set.
  - This is repeated on all ways to cut the original training set.
  - It requires $\mathcal{C}_n^p$ for a set of n examples
- **Leave-1-out cross-validation:** When $p = 1$, does not have this problem
- **k-fold cross-validation:**
  - The training set is randomly partitioned into k equal size subsamples.
  - Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data.
  - The cross-validation process is then repeated k times (the folds).

Cross-Validation

- **Leave-p-out cross-validation:**
  - We use p observations as the validation set and the remaining observations as the training set.
  - This is repeated on all ways to cut the original training set.
  - It requires $\mathcal{C}_n^p$ for a set of n examples
- **Leave-1-out cross-validation:** When $p = 1$, does not have this problem
- **k-fold cross-validation:**
  - The training set is randomly partitioned into k equal size subsamples.
  - Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data.
  - The cross-validation process is then repeated k times (the folds).
  - The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation

Metrics

How to evaluate how good my classifier is?

- **Precision:** is the fraction of retrieved instances that are relevant

$$P = \frac{TP}{TP + FN} = \frac{TP}{P}$$

Metrics

How to evaluate how good my classifier is?

- **Precision:** is the fraction of retrieved instances that are relevant

$$P = \frac{TP}{TP + FN} = \frac{TP}{P}$$

- **Recall:** is the fraction of relevant instances that are retrieved

$$R = \frac{TP}{TP + FP}$$

Metrics

How to evaluate how good my classifier is?

- **Precision:** is the fraction of retrieved instances that are relevant

$$P = \frac{TP}{TP + FN} = \frac{TP}{P}$$

- **Recall:** is the fraction of relevant instances that are retrieved

$$R = \frac{TP}{TP + FP}$$

- **F1 score:** harmonic mean of precision and recall

$$F1 = 2\frac{P \cdot R}{P + R}$$

More on Metrics

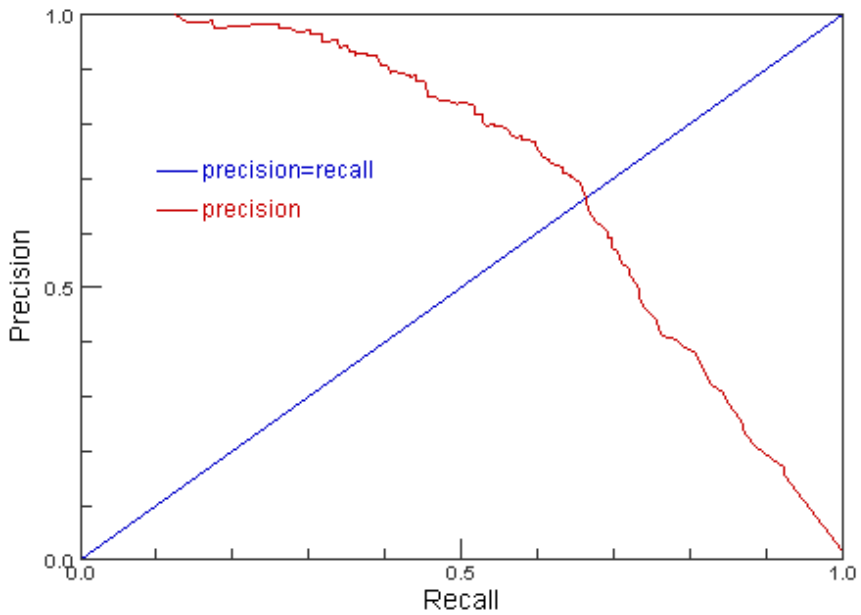
How to evaluate how good my classifier is?

- **Precision:** is the fraction of retrieved instances that are relevant
- **Recall:** is the fraction of relevant instances that are retrieved

More on Metrics

How to evaluate how good my classifier is?

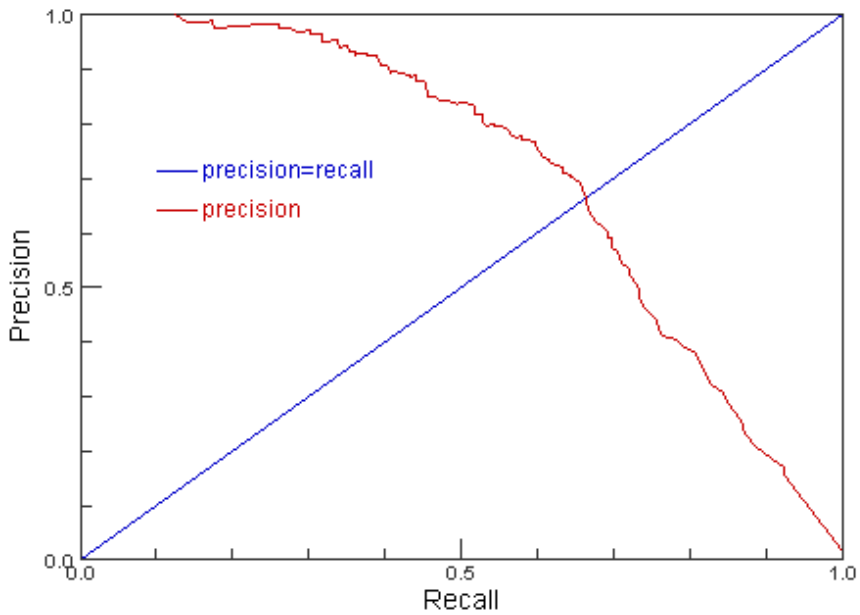
- **Precision:** is the fraction of retrieved instances that are relevant
- **Recall:** is the fraction of relevant instances that are retrieved
- **Precision Recall Curve**



More on Metrics

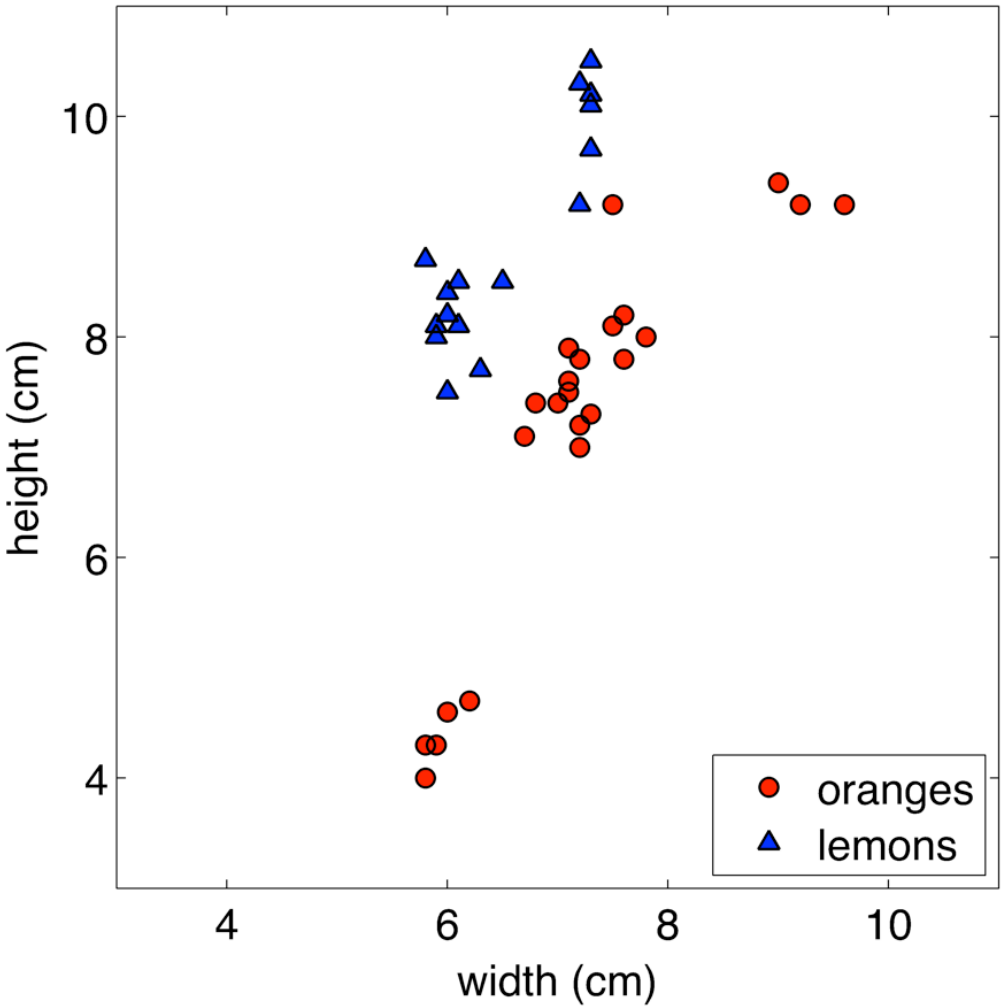
How to evaluate how good my classifier is?

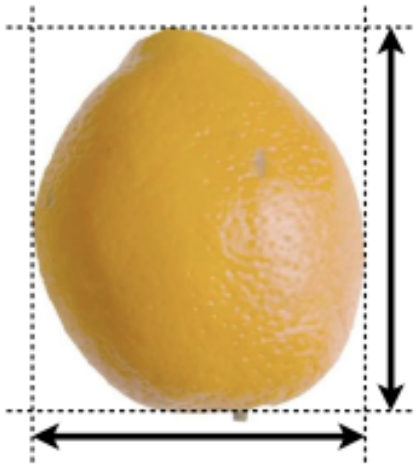
- **Precision:** is the fraction of retrieved instances that are relevant
- **Recall:** is the fraction of relevant instances that are retrieved
- **Precision Recall Curve**



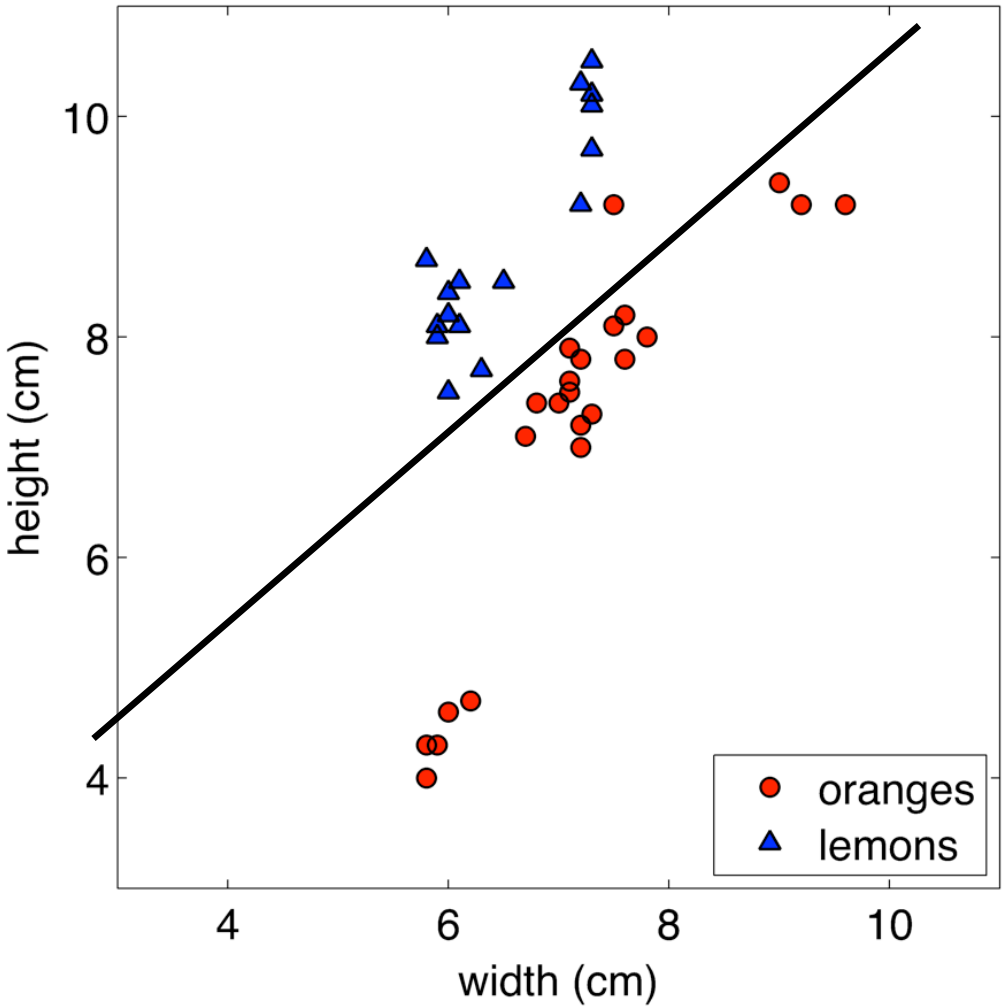
- **Average Precision (AP)**: mean under the curve

Classification: Oranges and Lemons



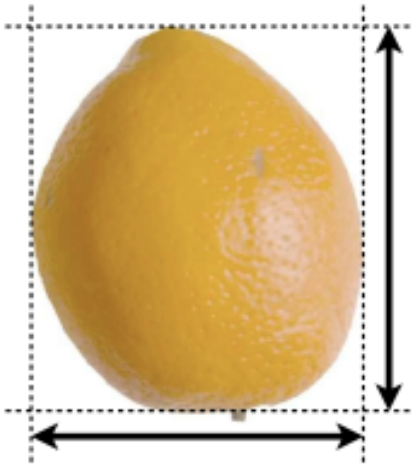


Classification: Oranges and Lemons



Can construct simple linear decision boundary:

$$y = \text{sign}(w_0 + w_1 x_1 + w_2 x_2)$$



What is the meaning of "linear" classification

- Linear classification means that the part that adapts is linear (just like linear regression)

$$z(x) = \mathbf{w}^T \mathbf{x} + w_0$$

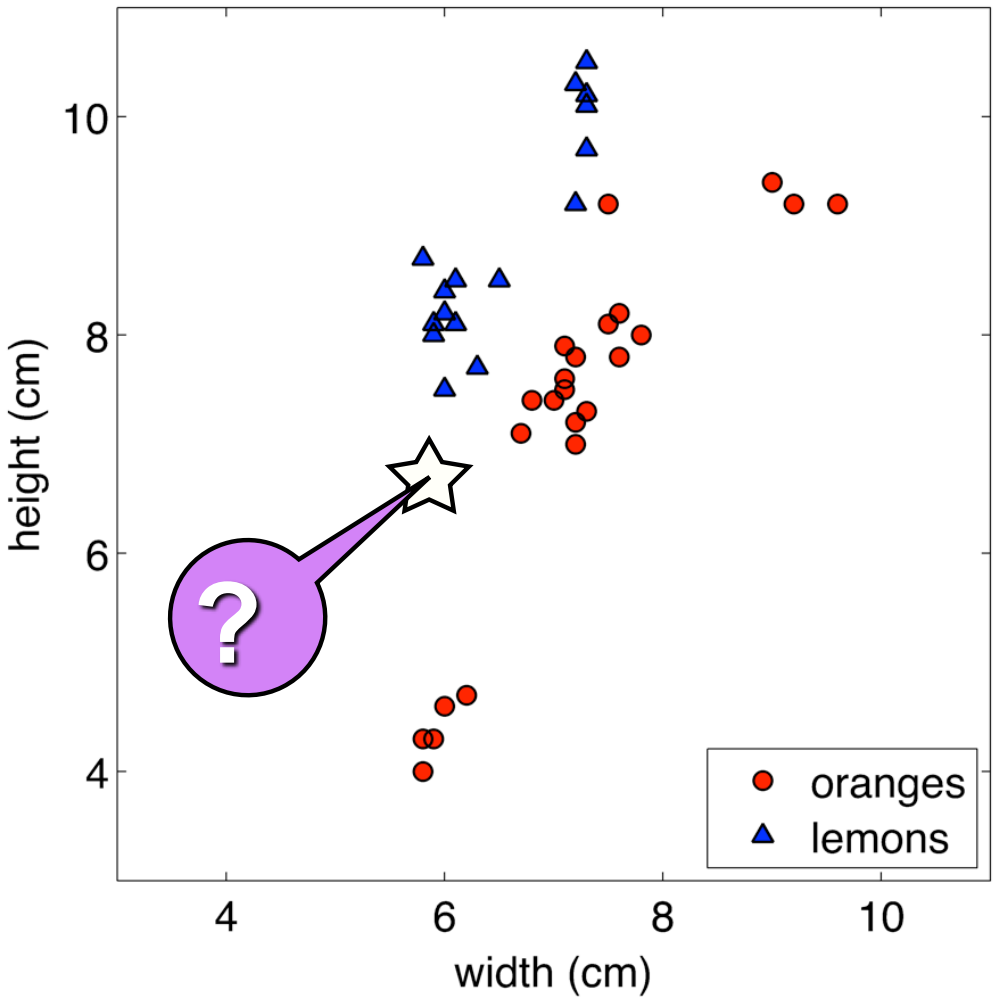
  with adaptive $\mathbf{w}, w_0$

- The adaptive part is follow by a non-linearity

$$y(\mathbf{x}) = f(z(\mathbf{x}))$$

- What f have we seen?

Classification as Induction



Instance-based Learning

- Alternative to parametric model is non-parametric

Instance-based Learning

- Alternative to parametric model is non-parametric
- Simple methods for approximating discrete-valued or real-valued target functions (classification or regression problems)

Instance-based Learning

- Alternative to parametric model is non-parametric
- Simple methods for approximating discrete-valued or real-valued target functions (classification or regression problems)
- Learning amounts to simply storing training data

Instance-based Learning

- Alternative to parametric model is non-parametric
- Simple methods for approximating discrete-valued or real-valued target functions (classification or regression problems)
- Learning amounts to simply storing training data
- Test instances classified using similar training instances

Instance-based Learning

- Alternative to parametric model is non-parametric
- Simple methods for approximating discrete-valued or real-valued target functions (classification or regression problems)
- Learning amounts to simply storing training data
- Test instances classified using similar training instances
- Embodies often sensible underlying assumptions:
  - Output varies smoothly with input
  - Data occupies sub-space of high-dimensional input space

Nearest Neighbors

- Assume training examples correspond to points in d-dimensional Euclidean space

Nearest Neighbors

- Assume training examples correspond to points in d-dimensional Euclidean space
- Target function value for new query estimated from known value of nearest training example(s)

Nearest Neighbors

- Assume training examples correspond to points in d-dimensional Euclidean space
- Target function value for new query estimated from known value of nearest training example(s)
- Distance typically defined to be Euclidean:

$$||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}|| = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - (x_j^{(b)})^2}$$

Nearest Neighbors

- Assume training examples correspond to points in d-dimensional Euclidean space
- Target function value for new query estimated from known value of nearest training example(s)
- Distance typically defined to be Euclidean:

$$||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}|| = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - (x_j^{(b)})^2}$$

- Algorithm
  1. find example $(\mathbf{x}^*, t^*)$ closest to the test instance $\mathbf{x}^{(q)}$
  2. output $y^{(q)} = t^*$

Nearest Neighbors

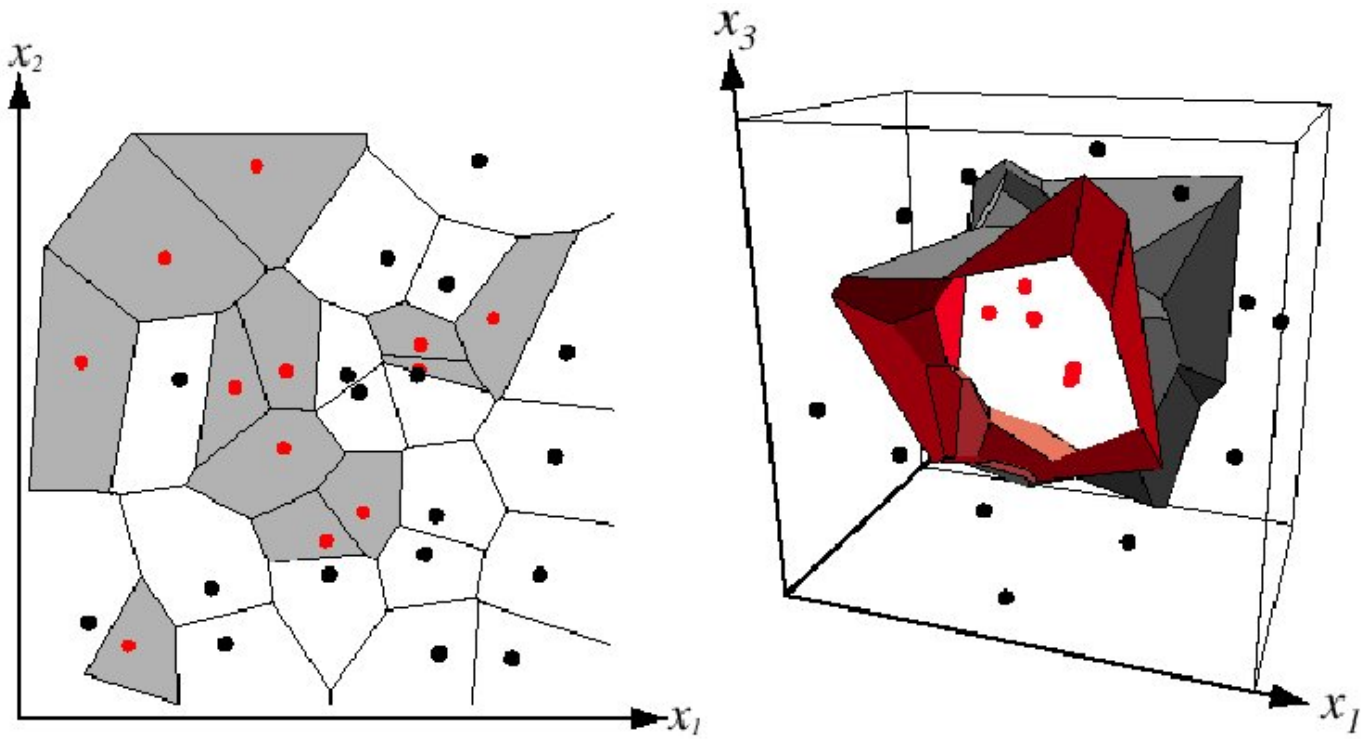
- Assume training examples correspond to points in d-dimensional Euclidean space
- Target function value for new query estimated from known value of nearest training example(s)
- Distance typically defined to be Euclidean:

$$||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}|| = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - (x_j^{(b)})^2}$$

- Algorithm
  1. find example $(\mathbf{x}^*, t^*)$ closest to the test instance $\mathbf{x}^{(q)}$
  2. output $y^{(q)} = t^*$
- Note: we don't need to compute the square root. Why?

Nearest Neighbors Decision Boundaries

- Nearest neighbor algorithm does not explicitly compute decision boundaries, but these can be inferred
- Decision boundaries: Voronoi diagram visualization
  - show how input space divided into classes
  - each line segment is equidistant between two points of opposite classes



k Nearest Neighbors

- Nearest neighbors sensitive to mis-labeled data ("class noise") $\rightarrow$ smooth by having k nearest neighbors vote

k Nearest Neighbors

- Nearest neighbors sensitive to mis-labeled data ("class noise") $\rightarrow$ smooth by having k nearest neighbors vote
- Algorithm:
  1. find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
  2. classification output is majority class

$$y = arg \max_{t^{(z)}} \sum_{r=1}^{k} \delta(t^{(z)}, t^{(r)})$$

k Nearest Neighbors: Issues & Remedies

- Some attributes have larger ranges, so are treated as more important $\rightarrow$ normalize scale

k Nearest Neighbors: Issues & Remedies

- Some attributes have larger ranges, so are treated as more important $\rightarrow$ normalize scale
- Irrelevant, correlated attributes add noise to distance measure $\rightarrow$ eliminate some attributes, or vary and possibly adapt weight of attributes

k Nearest Neighbors: Issues & Remedies

- Some attributes have larger ranges, so are treated as more important $\rightarrow$ normalize scale
- Irrelevant, correlated attributes add noise to distance measure $\rightarrow$ eliminate some attributes, or vary and possibly adapt weight of attributes
- Non-metric attributes (symbols) $\rightarrow$ Hamming distance

k Nearest Neighbors: Issues & Remedies

- Some attributes have larger ranges, so are treated as more important $\rightarrow$ normalize scale
- Irrelevant, correlated attributes add noise to distance measure $\rightarrow$ eliminate some attributes, or vary and possibly adapt weight of attributes
- Non-metric attributes (symbols) $\rightarrow$ Hamming distance
- Brute-force approach: calculate Euclidean distance to test point from each stored point, keep closest: $O(dn^2) \rightarrow$ reduce computational burden:
  1. Use subset of dimensions
  2. Use subset of examples

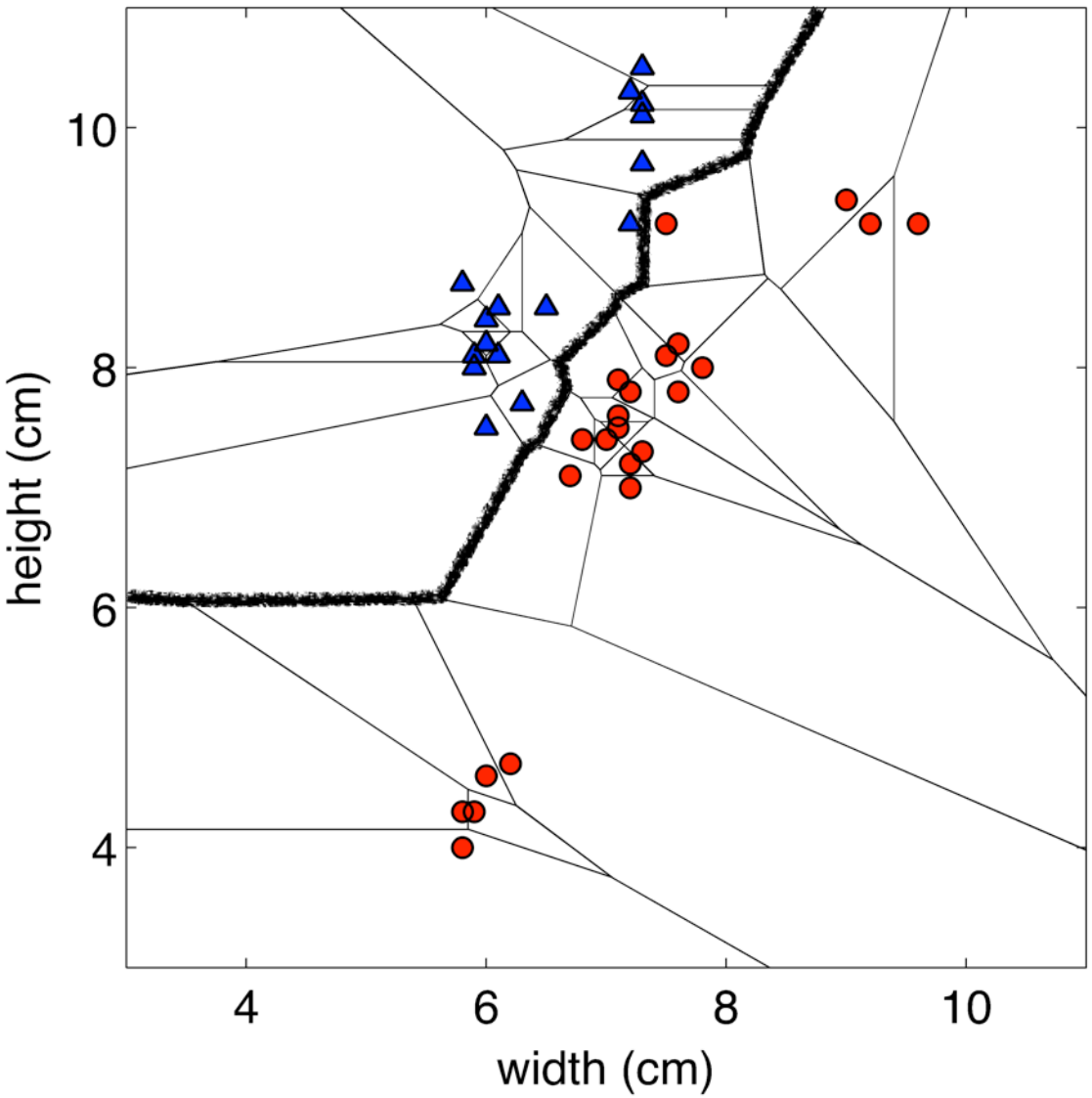
k Nearest Neighbors: Issues & Remedies

- Some attributes have larger ranges, so are treated as more important $\rightarrow$ normalize scale
- Irrelevant, correlated attributes add noise to distance measure $\rightarrow$ eliminate some attributes, or vary and possibly adapt weight of attributes
- Non-metric attributes (symbols) $\rightarrow$ Hamming distance
- Brute-force approach: calculate Euclidean distance to test point from each stored point, keep closest: $O(dn^2) \rightarrow$ reduce computational burden:
  1. Use subset of dimensions
  2. Use subset of examples
     - Remove examples that lie within Voronoi region

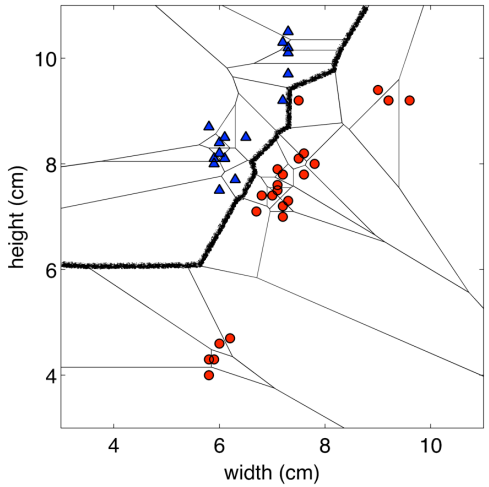
k Nearest Neighbors: Issues & Remedies

- Some attributes have larger ranges, so are treated as more important $\rightarrow$ normalize scale
- Irrelevant, correlated attributes add noise to distance measure $\rightarrow$ eliminate some attributes, or vary and possibly adapt weight of attributes
- Non-metric attributes (symbols) $\rightarrow$ Hamming distance
- Brute-force approach: calculate Euclidean distance to test point from each stored point, keep closest: $O(dn^2) \rightarrow$ reduce computational burden:
  1. Use subset of dimensions
  2. Use subset of examples
     - Remove examples that lie within Voronoi region
     - Form efficient search tree (kd-tree), use Hashing (LSH), etc

Decision Boundary K-NN

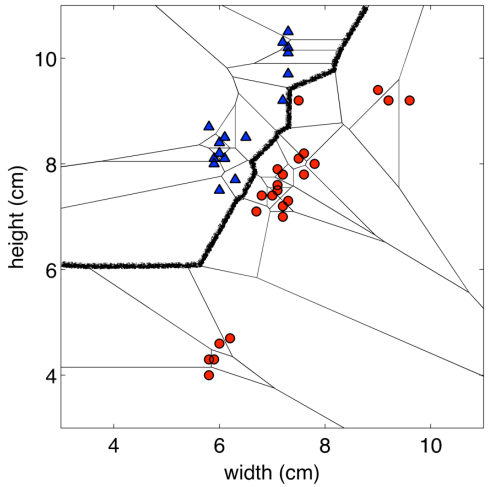


K-NN Summary



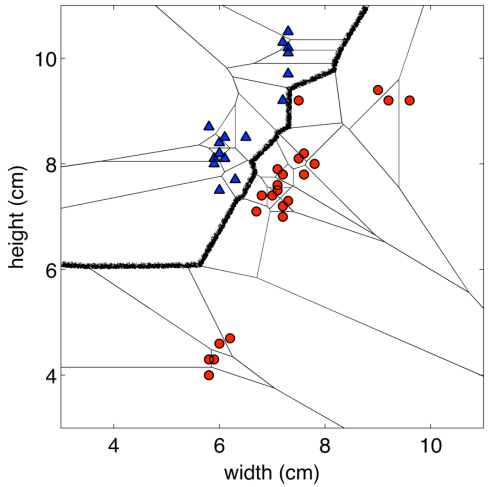
- Single parameter (k) $\rightarrow$ how do we set it?

K-NN Summary



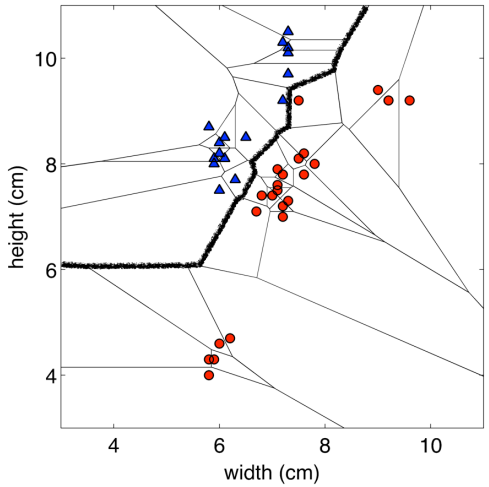
- Single parameter (k) $\rightarrow$ how do we set it?
- Naturally forms complex decision boundaries; adapts to data density

K-NN Summary



- Single parameter (k) $\rightarrow$ how do we set it?
- Naturally forms complex decision boundaries; adapts to data density
- Problems:
  - Sensitive to class noise.
  - Sensitive to dimensional scales.
  - Distances are less meaningful in high dimensions
  - Scales with number of examples

K-NN Summary



- Single parameter (k) $\rightarrow$ how do we set it?
- Naturally forms complex decision boundaries; adapts to data density
- Problems:
  - Sensitive to class noise.
  - Sensitive to dimensional scales.
  - Distances are less meaningful in high dimensions
  - Scales with number of examples
- Inductive Bias: What kind of decision boundaries do we expect to find?