Logistic Regression

Dr Mehrdad Ghaziasgar

Senior Lecturer in Computer Science
Department of Computer Science
University of the Western Cape
mghaziasgar@uwc.ac.za

UNIVERSITY of the
WESTERN CAPE

Content

- Part 1:
    - Classification With Logistic Regression
    - Model Evaluation For Classification
- Part 2:
    - Overfitting and Underfitting
- Part 3:
    - Practical Issues With Classification

Content – Part 1

- Classification With Logistic Regression
    - Model Representation
    - Obtaining the Decision Boundary
    - Cost Function
    - Cost Function Minimization For Classification

- Model Evaluation For Classification
    - Accuracy
    - Precision and Recall
    - F1 score

Classification With Logistic Regression

Model Representation

Classification With Logistic Regression

- Example classification problems:
  - Credit-riskiness
    - Not Risky – Risky $\boldsymbol{y} \in \{0,1\}$
  - Fraud detection
    - Not Fraudulent – Fraudulent $\boldsymbol{y} \in \{0,1\}$
  - Sentiment detection / analysis
    - Negative – Positive $\boldsymbol{y} \in \{0,1\}$
    - Neutral – Positive – Negative $\boldsymbol{y} \in \{0,1,2\}$
  - Facial expression recognition
    - Neutral – Positive – Negative $\boldsymbol{y} \in \{0,1,2\}$
    - Neutral – Happy – Sad – Surprise – Disgust – Angry $\boldsymbol{y} \in \{0,1,2,3,4,5\}$
- $\boldsymbol{y}$ now takes on a set of specific (discrete) values

Classification With Logistic Regression

- With regression problems, we were trying to draw a line to **FIT** the data tightly
- With classification problems, we're now trying to draw a line to **SEPARATE** the data nicely:
  - E.g. For a two-class problem: Separate data points with class $\boldsymbol{y} = \mathbf{1}$ from those with class $\boldsymbol{y} = \mathbf{0}$
  - E.g. For a five-class problem: Separate data points of classes $\boldsymbol{y} = \mathbf{0}$, $\boldsymbol{y} = \mathbf{1}$, $\boldsymbol{y} = \mathbf{2}$, $\boldsymbol{y} = \mathbf{3}$ and $\boldsymbol{y} = \mathbf{4}$ from each other
  - Etc.

Classification With Logistic Regression

- With classification problems, we're now trying to draw a line to **SEPARATE** the data nicely

- E.g. The problem of predicting the credit riskiness ($\boldsymbol{y}$) of a person based on one feature: amount of debt owed by a person (R) ($\boldsymbol{x}$)

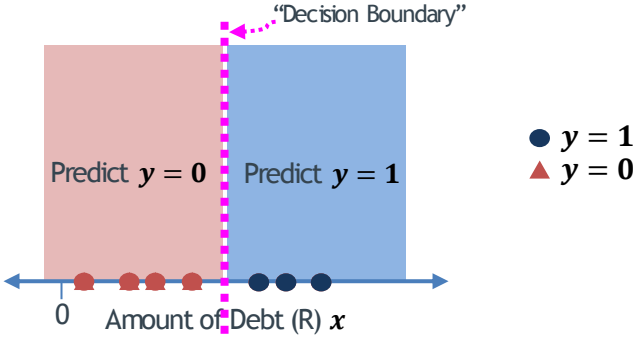
Amount of Debt (R) (x)	Credit Risk (y)
98000	0
549000	1
215000	0
69000	0
950000	1
...	...

Training set of credit riskiness

Classification With Logistic Regression

- With classification problems, we're now trying to draw a line to **SEPARATE** the data nicely

- E.g. 2 The problem of predicting the credit riskiness ($\boldsymbol{y}$) of a person based on two features: amount of debt (R) ($\boldsymbol{x_1}$) and the amount of payment defaults (R) that a person has had ($\boldsymbol{x_2}$)



Classification With Logistic Regression

- With classification problems, we're now trying to draw a line to **SEPARATE** the data nicely

- E.g. 2 The problem of predicting the credit riskiness ($\boldsymbol{y}$) of a person based on two features: amount of debt (R) ($\boldsymbol{x_1}$) and the amount of payment defaults (R) that a person has had ($\boldsymbol{x_2}$)

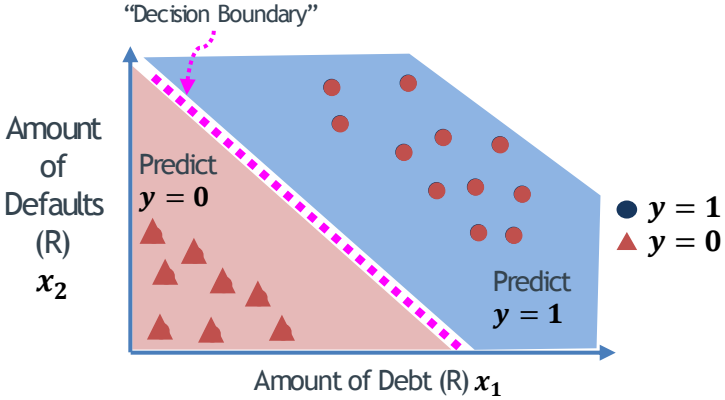
Amount of Debt (R) (x_1)	No. of Defaults (x_2)	Credit Risk (y)
98000	8150	0
549000	29000	1
215000	1750	0
69000	2000	0
950000	13000	1
...		...

Training set of credit riskiness

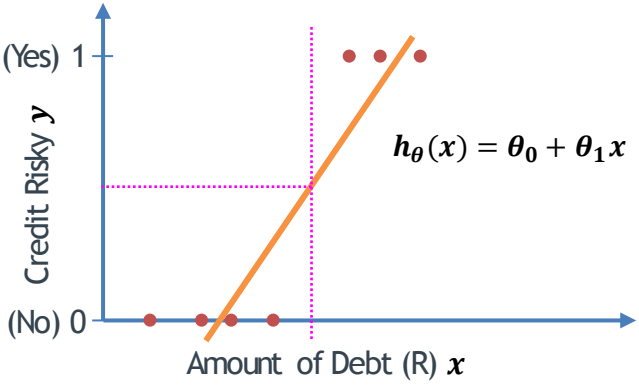
Classification With Logistic Regression

- With classification problems, we're now trying to draw a line to **SEPARATE** the data nicely

- E.g. 2 The problem of predicting the credit riskiness ($\boldsymbol{y}$) of a person based on two features: amount of debt (R) ($\boldsymbol{x_1}$) and the amount of payment defaults (R) that a person has had ($\boldsymbol{x_2}$)



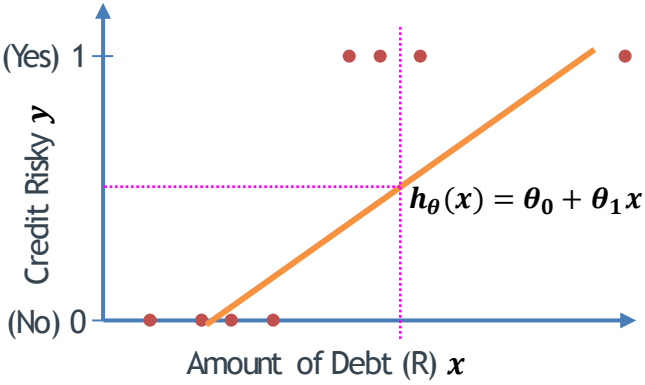
The Problem With Linear Regression



Possible Strategy(?):

- Use linear regression to fit a linear $h_\theta(x)$ to the data
- Then use a thresholding scheme to convert the linear scale into a discrete one e.g.:
  - If $h_\theta(x) < 0.5$ predict $y = 0$
  - If $h_\theta(x) \geq 0.5$ predict $y = 1$

The Problem With Linear Regression



Possible Strategy:

- Use linear regression to fit a linear $h_\theta(x)$ to the data
- Then use a thresholding scheme to convert the linear scale into a discrete one e.g.:
  - If $h_\theta(x) < 0.5$ predict $y = 0$
  - If $h_\theta(x) \geq 0.5$ predict $y = 1$

The Problem With Linear Regression

- Another problem with linear regression:
  - $\boldsymbol{h_\theta}$ ranges from $-\infty$ to $+\infty$
  - Whereas our labels are now very specific and discrete i.e. $\boldsymbol{y} \in \{0,1\}$

- What we want is a learning approach that:
  - Produces $\boldsymbol{h_\theta}$ such that $0 \leq \boldsymbol{h_\theta} \leq 1$
  - Focuses on **separating** (as opposed to **fitting**) data based on the labels
  - Separates the labels effectively regardless of noise in the data

Classification With Logistic Regression

- We will need a few elements to achieve classification:

1. A (new) hypothesis $\boldsymbol{h_\theta(x)}$ that can make predictions in the range $[0,1]$ given a feature vector $\boldsymbol{x}$

2. A (new) cost function $J(\boldsymbol{\theta})$ that gives us a increasingly large costs if the actual label of an example is 0 while the predicted label approaches 1 OR if the actual label of an example is 1 and the predicted label aproaches 0

3. A set of gradient descent update rules for $\boldsymbol{\theta}$ using the new hypothesis and cost function so that we can learn the $\boldsymbol{\theta}$ parameters

Model Representation

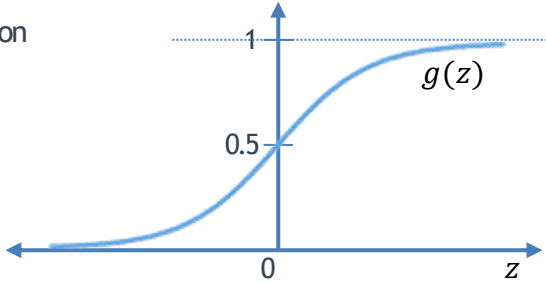
- We want a learning approach that:
  - Produces $\boldsymbol{h_\theta}$ such that $0 \leq \boldsymbol{h_\theta} \leq 1$
  - Focuses on **separating** (as opposed to **fitting**) data based on the labels

- We'll modify $\boldsymbol{h_\theta}$ from linear regression as follows:

$$\boldsymbol{h_\theta}(\boldsymbol{x}) = g(\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x}) \quad \boldsymbol{\theta_1 x_1} + \boldsymbol{\theta_2 x_2} + \cdots + \boldsymbol{\theta_n x_n}$$

- $g(\mathrm{z})$ is the "Sigmoid" or "Logistic" function

$$g(\mathrm{z}) = \frac{1}{1 + e^{-z}}$$



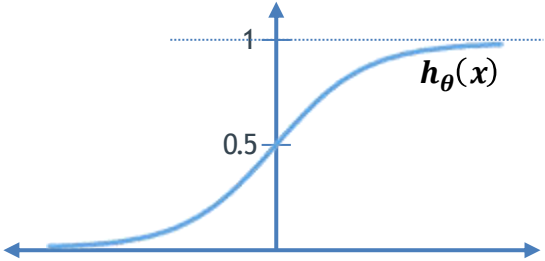
Model Representation

- Note that $g(z)$ is exactly in the range we want i.e. $0 \leq g(z) \leq 1$: $g(z) = \frac{1}{1+e^{-z}}$
  - If z is large e.g. 100 then:
  - If z is very small e.g. -100000 then:
- So now:

$$\boldsymbol{h_\theta(x)} = g(\boldsymbol{\theta}^T \boldsymbol{x})$$

- So:

$$\boldsymbol{h_\theta(x)} = \frac{1}{1+e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$$



Model Representation – Deciphering (the new) h_θ

- The new h_θ can be thought of as a probability function
- It outputs the probability that a given set of features represent $y = 1$
- For a given x:
  - The closer h_θ is to 1, the more likely the class is $y = 1$
  - The closer h_θ is to 0, the more likely the class is $y = 0$

- E.g. In the credit risk example earlier:
  - Assume we have somehow determined the parameters $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$
  - Given an arbitrary $x = \begin{bmatrix} x_0 = & 1 \\ x_1 = & \text{Amount of debt} \end{bmatrix}$
  - $h_\theta(x) = 0.2$ could mean 20% chance that client is a credit risk
  - $h_\theta(x) = 0.85$ could mean 85% chance that client is a credit risk

Model Representation – Deciphering h_θ

- In Mathematical notation:
  - $h_\theta(x) = P(y = 1|x;\theta)$
  - i.e. the probability that the class $y = 1$ given a specific example x and given the learned parameters θ

- As a by the way: Note that since we have only two classes $y = 0$ and $y = 1$, the sum of the probabilities of the two classes should be 1 i.e.

$$P(y = 1|x;\theta) + P(y = 0|x;\theta) = 1$$

$$P(y = 0|x;\theta) = 1 - P(y = 1|x;\theta)$$

$$P(y = 0|x;\theta) = 1 - h_\theta(x)$$

Model Representation – Deciphering h_θ

- As a by the way: Note that since we have only two classes $y = 0$ and $y = 1$, the sum of the probabilities of the two classes should be 1 i.e.

$$P(y = 0|x;\theta) = 1 - h_\theta(x)$$

- E.g. In the credit risk example earlier:

Given an arbitrary $x = \begin{bmatrix} x_0 = & 1 \\ x_1 = & \text{Amount of debt} \end{bmatrix}$

- If $h_\theta(x) = P(y = 1|x;\theta) = 0.8$ then the probability of the client being a credit risk is 80%

- So the probability of the same client NOT being a credit risk is:

$$P(y = 0|x;\theta) = 1 - 0.8 = 0.2 \text{ i.e. } 20\%$$

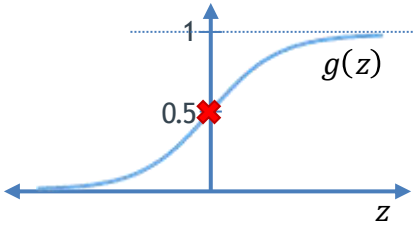
Classification With Logistic Regression

Obtaining the Decision Boundary

Obtaining the Decision Boundary

- Here's what we have so far for logistic regression:

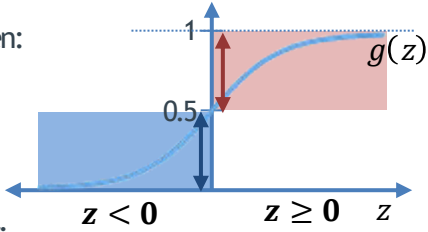
$$\boldsymbol{h_\theta(x)} = g(\boldsymbol{\theta}^T \boldsymbol{x}) \qquad g(z) = \frac{1}{1+e^{-z}}$$



- We know that $\boldsymbol{h_\theta(x)}$ is the probability that $\boldsymbol{y = 1}$
- So we can make predictions by computing $\boldsymbol{h_\theta(x)}$ and then:
  - Predict $\boldsymbol{y = 1}$ whenever $\boldsymbol{h_\theta(x) \geq 0.5}$
  - Predict $\boldsymbol{y = 0}$ whenever $\boldsymbol{h_\theta(x) < 0.5}$

Obtaining the Decision Boundary

- So we can make predictions by computing $\boldsymbol{h_\theta(x)}$ and then:
  - Predict $\boldsymbol{y = 1}$ whenever $\boldsymbol{h_\theta(x) \geq 0.5}$ so:
  - Predict $\boldsymbol{y = 1}$ whenever $g(\boldsymbol{\theta^T x}) \geq \boldsymbol{0.5}$
  - But $\boldsymbol{g(z) \geq 0.5}$ whenever $z \geq 0$
  - So $g(\boldsymbol{\theta^T x}) \geq \boldsymbol{0.5}$ whenever $\boldsymbol{\theta^T x \geq 0}$ so finally:
  - Predict $\boldsymbol{y = 1}$ whenever $\boldsymbol{\theta^T x \geq 0}$



- Similarly:
  - Predict $\boldsymbol{y = 0}$ whenever $\boldsymbol{h_\theta(x) < 0.5}$ so:
  - Predict $\boldsymbol{y = 0}$ whenever $g(\boldsymbol{\theta^T x}) < \boldsymbol{0.5}$
  - But $\boldsymbol{g(z) < 0.5}$ whenever $z < 0$
  - So $g(\boldsymbol{\theta^T x}) < \boldsymbol{0.5}$ whenever $\boldsymbol{\theta^T x < 0}$ so finally:
  - Predict $\boldsymbol{y = 0}$ whenever $\boldsymbol{\theta^T x < 0}$

Obtaining the Decision Boundary

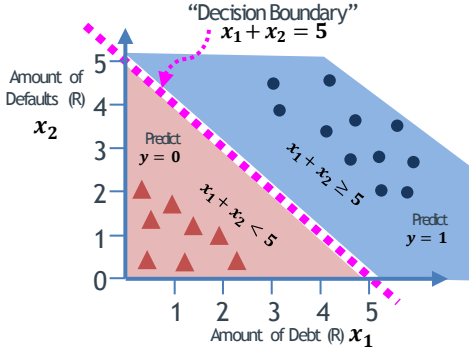
- So:
  - Predict $\boldsymbol{y} = \mathbf{1}$ whenever $\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x} \geq \mathbf{0}$
  - Predict $\boldsymbol{y} = \mathbf{0}$ whenever $\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x} < \mathbf{0}$

- Now let's try to decipher:
  - what $\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x}$ represents
  - what $\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x} \geq \mathbf{0}$ and $\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x} < \mathbf{0}$ mean

Obtaining the Decision Boundary

- Assume we've obtained $\boldsymbol{\theta} = \begin{bmatrix} \theta_0 = -5 \\ \theta_1 = 1 \\ \theta_2 = 1 \end{bmatrix}$

- So $\boldsymbol{\theta}^T \boldsymbol{x} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 = -5 + x_1 + x_2$



- And we now have:
- Predict $y = 1$ whenever $\boldsymbol{\theta}^T \boldsymbol{x} \geq 0$

$$-5 + x_1 + x_2 \geq 0$$
$$x_1 + x_2 \geq 5$$

- Predict $y = 0$ whenever $\boldsymbol{\theta}^T \boldsymbol{x} < 0$

$$-5 + x_1 + x_2 < 0$$
$$x_1 + x_2 \geq 5$$

- The "decision boundary" is given by:
- $\boldsymbol{\theta}^T \boldsymbol{x} = 0$ which is $-5 + x_1 + x_2 = 0$

$$x_1 + x_2 = 5$$

- Note that the decision boundary is defined by parameters $\boldsymbol{\theta}$ only

Obtaining the Decision Boundary

- So to recap:
    - We obtain $\boldsymbol{\theta}$ somehow (later – minimizing a cost function)
    - This $\boldsymbol{\theta}$ defines (in logistic regression) a line that separates the two classes $\boldsymbol{y} = \mathbf{0}$ and $\boldsymbol{y} = \mathbf{1}$ called the "decision boundary"
    - Given any sample $\boldsymbol{x}$, computing $\boldsymbol{h_\theta}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x}\right)$ tells us on which side of the decision boundary (or possibly on it) this sample falls
    - We use this ($\boldsymbol{h_\theta}(\boldsymbol{x})$) to then determine whether $\boldsymbol{x}$ belongs to $\boldsymbol{y} = \mathbf{0}$ or $\boldsymbol{y} = \mathbf{1}$

Non-Linear Decision Boundaries

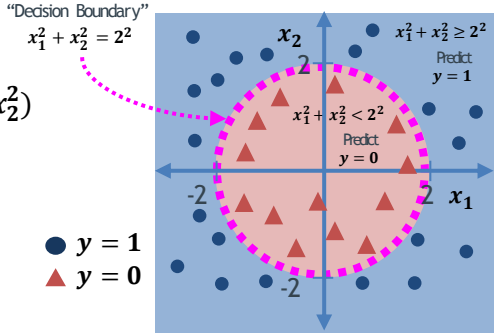
- In Linear Regression we were able to add non-linear features in order to **FIT** non-linear data more tightly:
    - Combinations of features
    - Higher-order features

- In Logistic Regression, we can add non-linear features in order to **SEPARATE** non-linear data (of different classes) more effectively:
    - Technique is exactly the same
    - The goal is different: to **separate** data classes
        - The hypothesis is different to that of Linear Regression

Non-Linear Decision Boundaries

- $h_\theta(x) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$

- Assume we've obtained $\theta = \begin{bmatrix} \theta_0 = -4 \\ \theta_1 = 0 \\ \theta_2 = 0 \\ \theta_3 = 1 \\ \theta_4 = 1 \end{bmatrix}$



- Then:

$$\theta^T x = -4 \cdot 1 + 0 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_1^2 + 1 \cdot x_2^2$$
$$\theta^T x = -4 + x_1^2 + x_2^2$$

- So the decision boundary is $\theta^T x = 0$

$$-4 + x_1^2 + x_2^2 = 0$$
$$x_1^2 + x_2^2 = 2^2$$

- Predict $y = 1$ whenever $\theta^T x \geq 0$

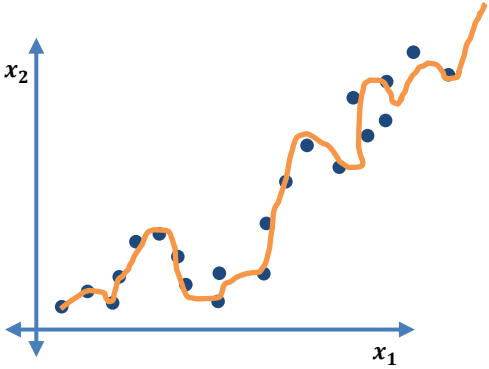
$$x_1^2 + x_2^2 \geq 2^2$$

- Predict $y = 0$ whenever $\theta^T x < 0$

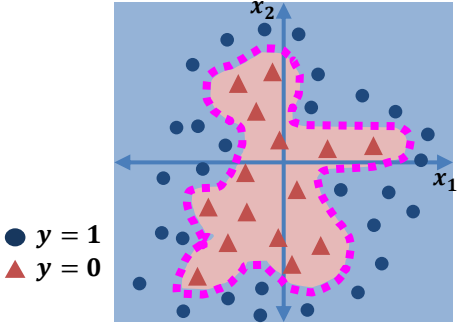
$$x_1^2 + x_2^2 < 2^2$$

Non-Linear Decision Boundaries

- In Linear Regression, we were able to get virtually any complex non-linear fit-line to tightly **FIT** even the most complex data



- In Logistic Regression, using the same technique we can get virtually any kind of complex non-linear decision boundary by adding non-linear features to **SEPARATE** the data



Classification With Logistic Regression

Cost Function

Cost Function – Can We Use $J(\boldsymbol{\theta})$ Of Linear Regression?

- We now have a (proven) valid hypothesis that can represent the separation of data based on two labels

$$\boldsymbol{h_\theta}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$$

- Now given a data set $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} \boldsymbol{x}_0^{(i)} \\ \boldsymbol{x}_1^{(i)} \\ \ldots \\ \boldsymbol{x}_n^{(i)} \end{bmatrix} \qquad \boldsymbol{y}^{(i)} \epsilon \{0,1\}$$

$= 1$

Amount of Debt (R) (x_1)	No. of Defaults (x_2)	Credit Risk (y)
98000	8150	0
549000	29000	1
215000	1750	0
69000	2000	0
950000	13000	1
…		…

Training set of credit riskiness

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}_0 \\ \boldsymbol{\theta}_1 \\ \cdot \\ \cdot \\ \boldsymbol{\theta}_n \end{bmatrix}$$

- We need a cost function $J(\boldsymbol{\theta})$ that we can minimize most effectively separates

Cost Function – Can We Use $J(\boldsymbol{\theta})$ Of Linear Regression?

- The Linear Regression cost function was:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{y}^{(i)}\right)^2$$

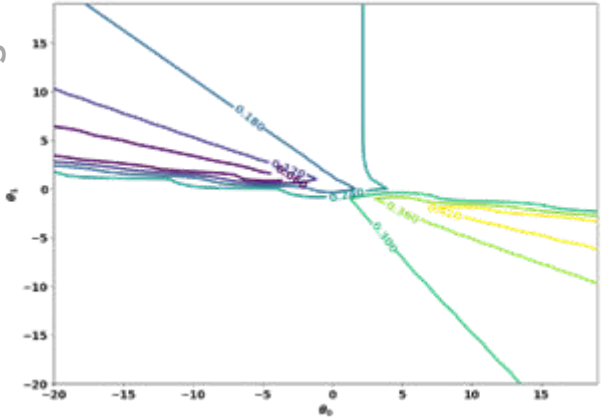
- If we use the Logistic Regression hypothesis $\boldsymbol{h_\theta}(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}$ with this cost function, we get a complex non-convex cost function (feel free to confirm for yourself):

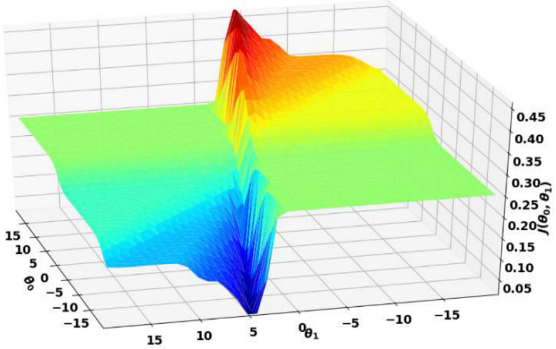
- Has many local minima:
  - Doesn't continuously decrease
  - Many regions where the function is completely flat

- We can't use gradient descent to minimize this
  - We'll end up in a local minimum

Cost Function – Can We Use $J(\boldsymbol{\theta})$ Of Linear Regression?

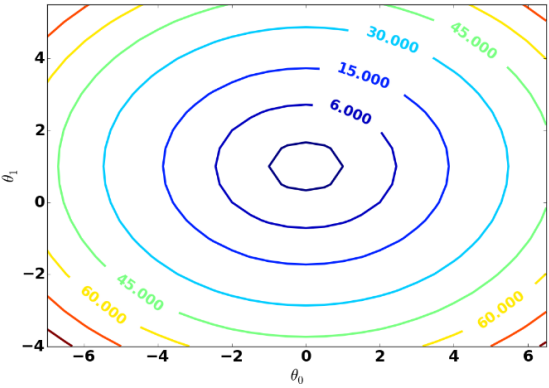
- E.g. For the Credit-Riskiness data set, using the least-squares cost function gives:

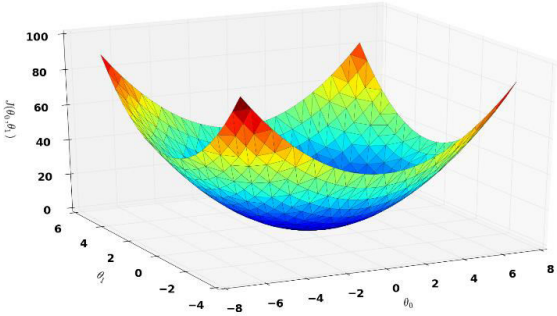




Cost Function

- We need a new/different cost function for Logistic Regression that will be convex (bowl-shaped) so we can minimize it





Cost Function

- We need a new/different cost function for Logistic Regression that will be convex (bowl-shaped) so we can minimize it

- Remember that $\mathbf{0} \leq \boldsymbol{h_\theta(x)} \leq \mathbf{1}$

- The new cost function that we need should:
    - Have increasingly larger cost if the actual label is $\mathbf{if}\ \boldsymbol{y} = \mathbf{1}$ but the prediction $\boldsymbol{h_\theta(x)}$ approaches $\mathbf{0}$ OR the actual label is $\mathbf{if}\ \boldsymbol{y} = \mathbf{0}$ but the prediction $\boldsymbol{h_\theta(x)}$ approaches $\mathbf{1}$

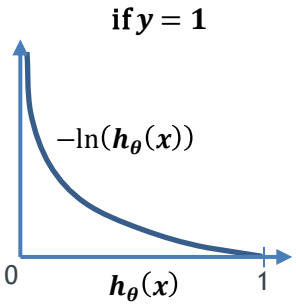
    - Have decreasing (minimum zero) cost if the prediction $\boldsymbol{h_\theta(x)}$ approaches the actual label, whether it is $\boldsymbol{y} = \mathbf{0}$ or $\boldsymbol{y} = \mathbf{1}$.

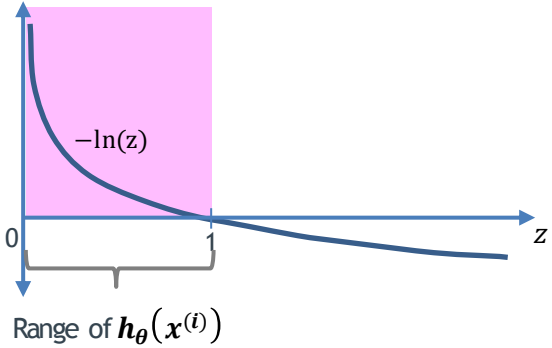
Cost Function

- The following cost function has a convex shape with classification data:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right), \boldsymbol{y}^{(i)}\right)$$

$$\mathrm{Cost}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right), \boldsymbol{y}^{(i)}\right) = \begin{cases} -\ln\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) & \textbf{if } \boldsymbol{y} = \mathbf{1} \\ -\ln\left(1 - \boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) & \textbf{if } \boldsymbol{y} = \mathbf{0} \end{cases}$$



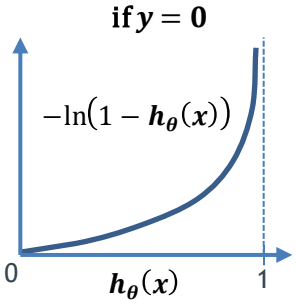


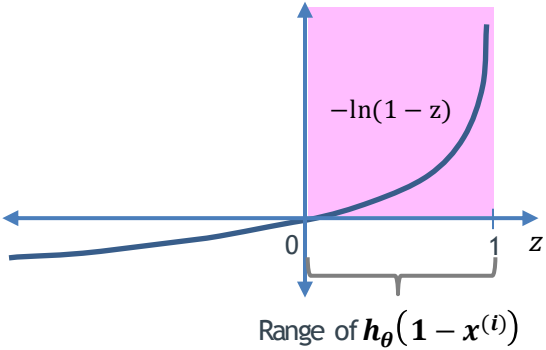
Cost Function

- The following cost function has a convex shape with classification data:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right), \boldsymbol{y}^{(i)}\right)$$

$$\mathrm{Cost}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right), \boldsymbol{y}^{(i)}\right) = \begin{cases} -\ln\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) & \textbf{if } \boldsymbol{y} = \mathbf{1} \\ -\ln\left(1 - \boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) & \textbf{if } \boldsymbol{y} = \mathbf{0} \end{cases}$$



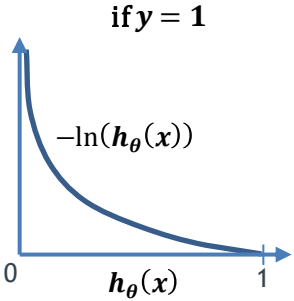


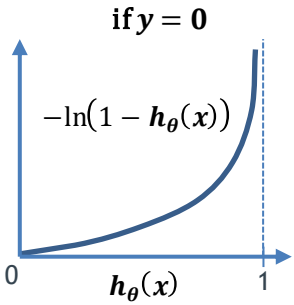
Cost Function

- The following cost function has a convex shape with classification data:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right), \boldsymbol{y}^{(i)}\right)$$

$$\mathrm{Cost}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right), \boldsymbol{y}^{(i)}\right) = \begin{cases} -\ln\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) & \textbf{if } \boldsymbol{y} = \mathbf{1} \\ -\ln\left(1 - \boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) & \textbf{if } \boldsymbol{y} = \mathbf{0} \end{cases}$$





Cost Function

- We need a combined (not piece-wise defined) cost function to use with gradient descent
  - We need to combine the two parts of the function $\mathrm{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y})$

$$\mathrm{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y}) = \begin{cases} -\ln(\boldsymbol{h_\theta}(\boldsymbol{x})) & \textbf{if } \boldsymbol{y} = \mathbf{1} \\ -\ln\left(1 - \boldsymbol{h_\theta}(\boldsymbol{x})\right) & \textbf{if } \boldsymbol{y} = \mathbf{0} \end{cases}$$

- Remember that the labels have to be either $\boldsymbol{y} = \mathbf{0}$ or $\boldsymbol{y} = \mathbf{1}$
- If that's the case, one clever way of combining the two parts of $\mathrm{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y})$ is:

$$\mathrm{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y}) = -\mathbf{y} \ln(\boldsymbol{h_\theta}(\boldsymbol{x})) - (1 - \boldsymbol{y}) \ln\left(1 - \boldsymbol{h_\theta}(\boldsymbol{x})\right)$$

$\textbf{if } \boldsymbol{y} = \mathbf{1}$

$$\begin{aligned} \mathrm{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y}) &= -\mathbf{1} \cdot \ln(\boldsymbol{h_\theta}(\boldsymbol{x})) - (1 - \mathbf{1}) \ln\left(1 - \boldsymbol{h_\theta}(\boldsymbol{x})\right) \\ &= -\mathbf{1} \cdot \ln(\boldsymbol{h_\theta}(\boldsymbol{x})) - (\mathbf{0}) \cdot \ln\left(1 - \boldsymbol{h_\theta}(\boldsymbol{x})\right) \\ &= -\ln(\boldsymbol{h_\theta}(\boldsymbol{x})) \end{aligned}$$

Cost Function

- We need a combined (not piece-wise defined) cost function to use with gradient descent
  - We need to combine the two parts of the function $\mathrm{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y})$

$$\mathrm{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y}) = \begin{cases} -\ln(\boldsymbol{h_\theta}(\boldsymbol{x})) & \textbf{if } \boldsymbol{y} = \mathbf{1} \\ -\ln\left(1 - \boldsymbol{h_\theta}(\boldsymbol{x})\right) & \textbf{if } \boldsymbol{y} = \mathbf{0} \end{cases}$$

- Remember that the labels have to be either $\boldsymbol{y} = \mathbf{0}$ or $\boldsymbol{y} = \mathbf{1}$
- If that's the case, one clever way of combining the two parts of $\mathrm{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y})$ is:

$$\mathrm{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y}) = -\mathbf{y} \ln(\boldsymbol{h_\theta}(\boldsymbol{x})) - (1 - \boldsymbol{y}) \ln\left(1 - \boldsymbol{h_\theta}(\boldsymbol{x})\right)$$

$\textbf{if } \boldsymbol{y} = \mathbf{0}$

$$\begin{aligned} \mathrm{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y}) &= -\mathbf{0} \cdot \ln(\boldsymbol{h_\theta}(\boldsymbol{x})) - (1 - \mathbf{0}) \ln\left(1 - \boldsymbol{h_\theta}(\boldsymbol{x})\right) \\ &= -\mathbf{0} \cdot \ln(\boldsymbol{h_\theta}(\boldsymbol{x})) - (\mathbf{1}) \cdot \ln\left(1 - \boldsymbol{h_\theta}(\boldsymbol{x})\right) \\ &= -\ln\left(1 - \boldsymbol{h_\theta}(\boldsymbol{x})\right) \end{aligned}$$

Cost Function

- So with new combined version, we have: $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right), \boldsymbol{y}^{(i)}\right)$

$$\begin{aligned}\mathrm{Cost}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right), \boldsymbol{y}^{(i)}\right) &= -\boldsymbol{y}^{(i)}\ln\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) - (1-\boldsymbol{y}^{(i)})\ln\left(1-\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) \\ &= -\left[\boldsymbol{y}^{(i)}\ln\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) + (1-\boldsymbol{y}^{(i)})\ln\left(1-\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right)\right]\end{aligned}$$

- So:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[\boldsymbol{y}^{(i)}\ln\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) + (1-\boldsymbol{y}^{(i)})\ln\left(1-\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right)\right]$$

- We can now get the optimal parameters $\boldsymbol{\theta}$ by minimizing this cost function $J(\boldsymbol{\theta})$ using gradient descent (or any other optimization technique)

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \;\; J(\boldsymbol{\theta})$$

Cost Function

- We can now get the optimal parameters $\boldsymbol{\theta}$ by minimizing this cost function $J(\boldsymbol{\theta})$ using gradient descent (or any other optimization technique)

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \;\; J(\boldsymbol{\theta})$$

- We can then use the optimal $\boldsymbol{\theta}$ to make predictions with the hypothesis $\boldsymbol{h_\theta(x)}$:

$$\boldsymbol{h_\theta(x)} = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$$

Predict $\boldsymbol{y = 1}$ whenever $\boldsymbol{h_\theta(x) \geq 0.5}$

Predict $\boldsymbol{y = 0}$ whenever $\boldsymbol{h_\theta(x) < 0.5}$

Classification With Logistic Regression

Cost Function Minimization For Classification

- Classification With Logistic Regression
  - Model Representation
  - Obtaining the Decision Boundary
  - Cost Function
  - Cost Function Minimization For Classification
  - Multi-Class Classification

- Model Evaluation For Classification
  - Accuracy
  - Precision and Recall
  - F1 score

Cost Function Minimization For Classification

- With a cost function defined, we just need to minimize it to get the optimal $\boldsymbol{\theta}$

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \;\; J(\boldsymbol{\theta})$$

- There are a few ways to minimize the cost function (or any function):
  - Gradient Descent (we've already seen this)
  - Advanced optimization algorithms (we won't look at HOW these work – look it up):
    - Conjugate gradient descent
    - BFGS algorithm
    - L-BFGS algorithm
    - Nelder-Mead algorithm
    - Powell algorithm
    - Etc. (there are many)

Gradient Descent For Classification

- Generic gradient descent update equation (as seen before):

Repeat until convergence:

Update all $\boldsymbol{\theta_j}$ simultaneously:

$$\boldsymbol{\theta_j} \leftarrow \boldsymbol{\theta_j} - \boldsymbol{\alpha} \boxed{\frac{\partial}{\partial \boldsymbol{\theta_j}} J(\boldsymbol{\theta})}$$

?

- The hypothesis and cost function are different to that of Linear Regression:

$$\boldsymbol{h_\theta}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$$

- Cost function:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[\boldsymbol{y}^{(i)}\ln\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right) + (1 - \boldsymbol{y}^{(i)})\ln\left(1 - \boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)\right)\right]$$

Gradient Descent for Classification

- For any parameter $\boldsymbol{\theta_j}$, the gradient is given by:

$$\frac{\partial}{\partial \boldsymbol{\theta_j}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{y}^{(i)}\right) \boldsymbol{x}_j^{(i)}$$

Repeat until convergence:

Update all $\boldsymbol{\theta}$ simultaneously:

$$\boldsymbol{\theta_0} \leftarrow \boldsymbol{\theta_0} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{y}^{(i)}\right) \cdot \boldsymbol{x_0^{(i)}}$$

$$\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{y}^{(i)}\right) \cdot \boldsymbol{x_1^{(i)}}$$

$$\ldots$$

$$\boldsymbol{\theta_n} \leftarrow \boldsymbol{\theta_n} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{y}^{(i)}\right) \cdot \boldsymbol{x_n^{(i)}}$$

- Algorithm appears to be identical to gradient descent with Linear Regression
  - It is NOT! The hypothesis here is that of logistic regression
- The tuning of the learning rate follows a similar technique
- Feature scaling should be applied in the same way as explained before to achieve faster convergence
- Shouldn't use a for loop: Vectorize the approach

Advanced Optimization Algorithms For Classification

- Advanced optimization algorithms (we won't look at HOW these work – look it up):
  - Conjugate gradient descent
  - BFGS algorithm
  - L-BFGS algorithm
  - Nelder-Mead algorithm
  - Powell algorithm
  - Etc. (there are many)

- These algorithms take in the definition of the cost function and possibly the gradient values
  - They minimize the function using advanced techniques
  - No need to worry about learning rate
  - Work much faster than gradient descent
  - Implemented in SciPy

Advanced Optimization Algorithms For Classification

- Implemented in SciPy's optimize library:

```
from scipy.optimize import minimize

theta = np.zeros(X.shape[1]) #Initialize all thetas to zeros
result = minimize(costFunc, theta, args=(X,y), method='TNC', jac=gradientFunc,
options={'maxiter' : 400, 'disp': True})

print(result.x)
```

- costFunc is a function that returns the cost as per:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[\boldsymbol{y}^{(i)}\ln\left(\boldsymbol{h}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right) + (1-\boldsymbol{y}^{(i)})\ln\left(1-\boldsymbol{h}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right)\right]$$

- gradientFunc is a function that returns a list/vector of the gradients of all the $\boldsymbol{\theta}_{\boldsymbol{j}}$ as per:

$$\frac{\partial}{\partial\boldsymbol{\theta}_{\boldsymbol{j}}}J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{h}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{y}^{(i)}\right)\boldsymbol{x}_{j}^{(i)}$$

Advanced Optimization Algorithms For Classification

- Implemented in SciPy's optimize library:

```
from scipy.optimize import minimize

theta = np.zeros(X.shape[1]) #Initialize all thetas to zeros
result = minimize(costFunc, theta, args=(X,y), method='TNC', jac=gradientFunc,
options={'maxiter' : 400, 'disp': True})

print(result.x)
```

- result.x will contain the optimal $\boldsymbol{\theta}$ values
- Note that X must contain an extra column of zeros representing feature $\boldsymbol{x_0}$
- Predictions can then be made by applying $\boldsymbol{h_\theta}(\boldsymbol{x}) = g(\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x})$

Advanced Optimization Algorithms For Classification

- Implemented in sklearn specifically for Logistic Regression:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='auto')

clf.fit(X, y)
```

- Note that X should <u>NOT</u> contain the extra column of 1s for feature x_0.

- Predictions can then be made seamlessly:

```
predictionvalue = clf.predict(x_new)
print(predictionvalue)
```

Classification With Logistic Regression

Multi-Class Classification

Multi-Class Classification

- Two broad types of classification

- **Binary classification;** $y \in \{0,1\}$

- Two classes E.g.
- Credit-riskiness
  - Not Risky – Risky

- Fraud detection
  - Non-fraud – Fraud

- Sentiment detection / analysis
  - Negative – Positive

- Facial expression recognition
  - Emotive – Non-Emotive

- **Multi-class classification;** $y \in \{0,1,2 \dots, k\}$

- Three or more classes E.g.
- Credit-riskiness level
  - 0 Risk – 1 Risk – 2 Risk – 3 Risk $y \in \{0,1,2,3\}$

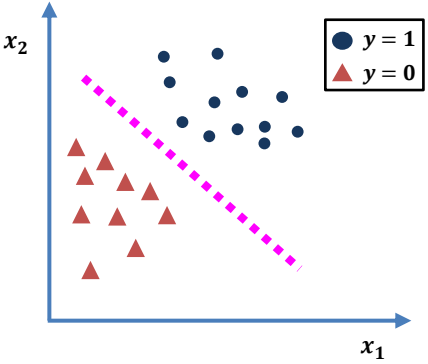
- Sentiment detection / analysis
  - Neutral – Positive – Negative $y \in \{0,1,2\}$

- Facial expression recognition
  - Neutral – Positive – Negative $y \in \{0,1,2\}$
  - Neutral – Happy – Sad – Surprise – Disgust – Angry $y \in \{0,1,2,3,4,5\}$

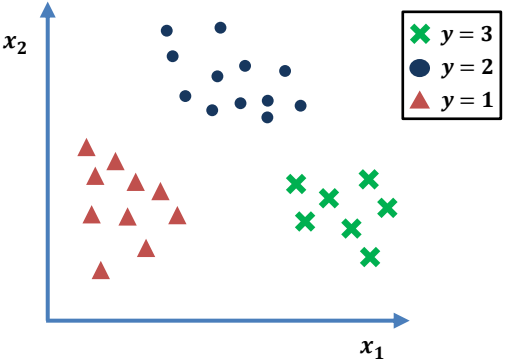
Multi-Class Classification

- Two broad types of classification

- **Binary classification;** $y \in \{0,1\}$



- **Multi-class classification;** $y \in \{1,2,3\}$

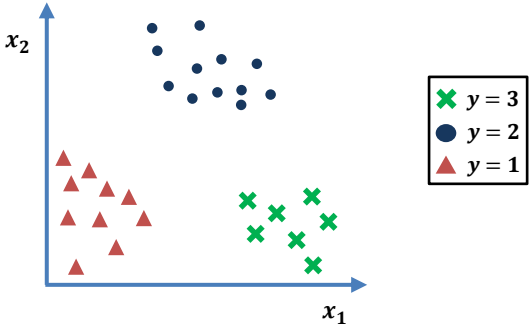


Multi-Class Classification

- Two main methods of achieving multi-class classification:
  - One-vs-Rest
  - One-vs-One

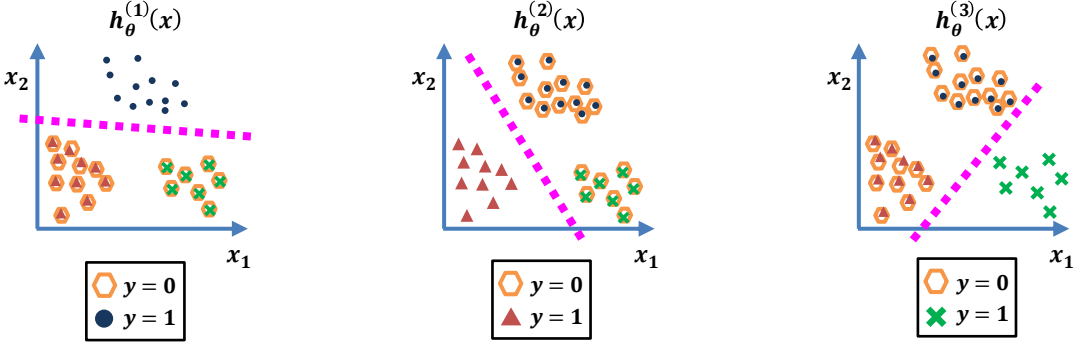
Multi-Class Classification – One-Vs-Rest

- One-vs-Rest:
- Decompose the K classes into K binary classification problems
- Create K separate classifiers $h_\theta^{(i)}(x)$ where $i = 1, 2 \ldots, K$.
- In each classifier, set one of the K classes as $y = 1$ and combine all other classes into a single class $y = 0$
- E.g. for a 3-class problem:



Multi-Class Classification – One-Vs-Rest

- One-vs-Rest:
- Decompose the K classes into K binary classification problems
- Create K separate classifiers $h_\theta^{(i)}(x)$ where $i = 1, 2 \ldots, K$.
- In each classifier, set one of the K classes as $y = 1$ and combine all other classes into a single class $y = 0$
- E.g. for a 3-class problem:



Multi-Class Classification – One-Vs-Rest

- One-vs-Rest:
- Decompose the K classes into K binary classification problems
- Create K separate classifiers $\boldsymbol{h_\theta^{(i)}(x)}$ where $\boldsymbol{i = 1, 2 \ldots, K}$.
- In each classifier, set one of the K classes as $\boldsymbol{y = 1}$ and combine all other classes into a single class $\boldsymbol{y = 0}$

- Remember that the prediction of $\boldsymbol{h_\theta^{(i)}(x)}$ is actually a probability; in this case, the probability that $\boldsymbol{y = i}$:

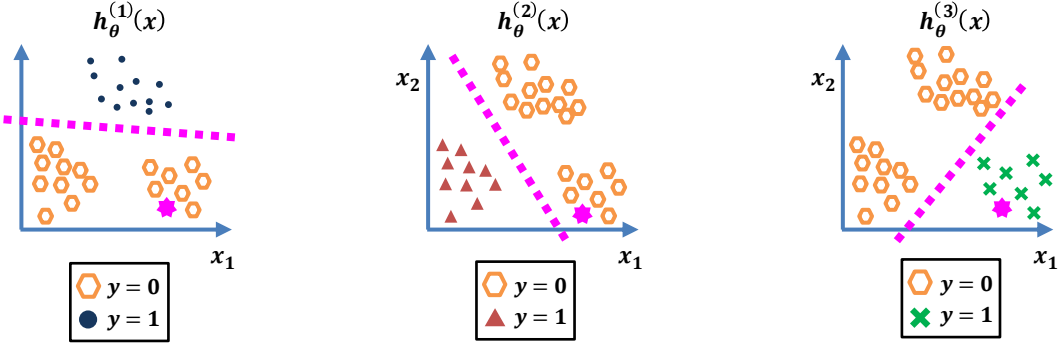
$$\boldsymbol{h_\theta^{(i)}(x) = P(y = 1|x; \theta)}$$

- Then, given an unknown sample $\boldsymbol{x}$, pass it to all $\boldsymbol{h_\theta^{(i)}(x)}$ classifiers: classifier $\boldsymbol{i}$ with the highest probability means that $\boldsymbol{x}$ most likely belongs to class $\boldsymbol{i}$
- Mathematically:

$$\max_{\boldsymbol{i}} \boldsymbol{h_\theta^{(i)}(x)}$$

Multi-Class Classification – One-Vs-Rest

- E.g. With the 3-class problem: assume we have a new $\boldsymbol{x}$ that actually belongs to class ✖ i.e. $\boldsymbol{y} = \mathbf{3}$
- Pass $\boldsymbol{x}$ to $\boldsymbol{h}_{\boldsymbol{\theta}}^{(1)}(\boldsymbol{x})$, $\boldsymbol{h}_{\boldsymbol{\theta}}^{(2)}(\boldsymbol{x})$ and $\boldsymbol{h}_{\boldsymbol{\theta}}^{(3)}(\boldsymbol{x})$
- E.g. $\boldsymbol{h}_{\boldsymbol{\theta}}^{(1)}(\boldsymbol{x})$ will predict 0.01; $\boldsymbol{h}_{\boldsymbol{\theta}}^{(2)}(\boldsymbol{x})$ will predict 0.15; and $\boldsymbol{h}_{\boldsymbol{\theta}}^{(3)}(\boldsymbol{x})$ will predict 0.97. Therefore the predicted class is taken to be $\boldsymbol{y} = \mathbf{3}$



Multi-Class Classification – One-Vs-One

- One-vs-One:
- Create a series of binary classifiers $h_\theta^{(i,j)}(x)$, which deal with only a pair of classes i and j.
- Remember that the prediction of $h_\theta^{(i,j)}(x)$ is actually a probability; in this case, the probability that $y = i$:

$$h_\theta^{(i)}(x) = P(y = 1|x;\theta)$$

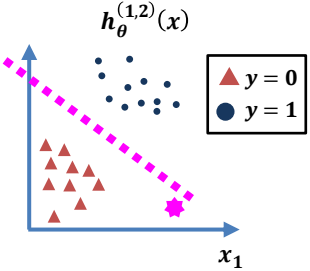
- But we can also infer the probability of class $y = 0$:

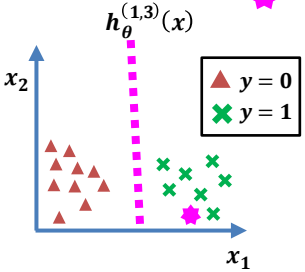
$$P(y = 0|x;\theta) = 1 - h_\theta^{(i)}(x)$$

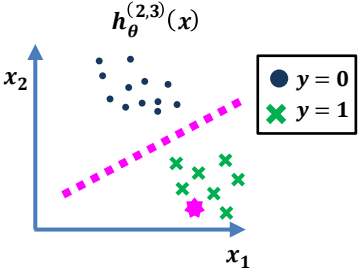
- In this technique, we keep track of the total probability of each class:
  - The class with the max probability wins

Multi-Class Classification – One-Vs-One

- E.g. With the 3-class problem: assume we have a new $\boldsymbol{x}$ that actually belongs to class ✖ i.e. $\boldsymbol{y} = \mathbf{3}$







$h_\theta^{(i)}(x) = 0.48$ so:

- ● $P(y = 1|x;\theta) = 0.48$
- ▲ $P(y = 0|x;\theta) = 0.52$

$h_\theta^{(i)}(x) = 0.97$ so:

- ✖ $P(y = 1|x;\theta) = 0.97$
- ▲ $P(y = 0|x;\theta) = 0.03$

$h_\theta^{(i)}(x) = 0.99$ so:

- ✖ $P(y = 1|x;\theta) = 0.99$
- ● $P(y = 0|x;\theta) = 0.01$

Totals:

● $= 0.48 + 0.01$
$= 0.49$

▲ $= 0.52 + 0.03$
$= 0.55$

✖ $= 0.97 + 0.99$
$= 1.96$

Multi-Class Classification – One-Vs-One

- One-vs-One:
- Number of classifiers trained:
  - 3 classes: 3 classifiers i.e. classes 1-2, classes 1-3, classes 2-3
  - 4 classes: 6 classifiers i.e. classes 1-2, classes 1-3, classes 1-4, classes 2-3, classes 2-4, classes 3-4
  - K classes: $\frac{1}{2}K(K-1)$

Multi-Class Classification

- One-vs-Rest:
  - Computationally cheaper and simpler
    - Fewer classifiers are trained
    - Fewer classifiers are used to make predictions
  - More sensitive to unbalanced classes (find out why)
    - Can be less accurate

- One-vs-One:
  - Computationally more expensive
    - Many more classifiers are trained
    - Many more classifiers are used to make predictions
  - Less sensitive to unbalanced classes (find out why)
    - Can be more accurate

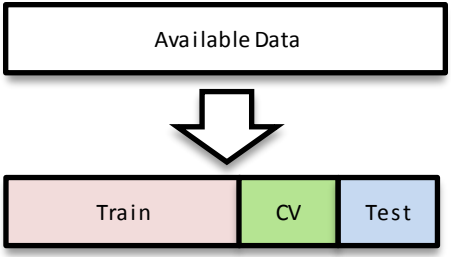
Model Evaluation For Classification

Accuracy

- Classification With Logistic Regression
  - Model Representation
  - Obtaining the Decision Boundary
  - Cost Function
  - Cost Function Minimization For Classification
  - Multi-Class Classification
- Model Evaluation For Classification
  - Accuracy
  - Precision and Recall
  - F1 score

Introduction

- When evaluating/comparing models, all the techniques described under Linear Regression apply i.e.
  - Divide up data into Train-CV-Test OR:
  - *k*-fold Cross-validation



- Question: what metrics do we use for classification?

- The metrics change for classification:
  - Accuracy
  - Precision
  - Recall
  - F1 score

- Running example: credit-riskiness
  - Assume we've trained a model $\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right)$ for credit-riskiness on the Train set $\boldsymbol{x}^{(i)}_{\text{Train}}$
  - We need to evaluate the model on the Test set $\boldsymbol{x}^{(i)}_{\text{Test}}$

Accuracy

- Accuracy: the most basic metric
- Apply the test set samples $\boldsymbol{x}_{\text{Test}}^{(i)}$ to the trained model $\boldsymbol{h}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)$ to get prediction labels for each sample
- The accuracy is the proportion/percentage of test samples for which the actual label $\boldsymbol{y}_{\text{Test}}^{(i)}$ matched the predicted label $\boldsymbol{h}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{\text{Test}}^{(i)}\right)$:

$$\textbf{Accuracy} = \frac{\textbf{Total Correct}}{\textbf{Total Test Samples}} \times \mathbf{100}$$

Model Evaluation For Classification

Precision and Recall

- Classification With Logistic Regression
  - Model Representation
  - Obtaining the Decision Boundary
  - Cost Function
  - Cost Function Minimization For Classification
  - Multi-Class Classification

- Model Evaluation For Classification
  - Accuracy
  - Precision and Recall
  - F1 score

Precision and Recall – The Problem

- Skew classes:
  - A case in which one of the classes to be predicted is scarce in the data set
  - In such data sets, we may not have many examples of the one of the classes
- E.g.
  - Fraud: there may be far fewer examples of fraud cases than legitimate cases
  - Cancer: there may be far fewer examples of cancerous cases than non-cancerous cases
- In such cases, one class (usually the class we're interested in) may only make up a small percentage of the examples e.g. only 0.5% of the examples
- The classifier may become very good at detecting negative cases, but not positive cases
- Using accuracy will not be a good reflection of the classifier performance

Precision and Recall – The Problem

- Skew classes:
  - A case in which one of the classes to be predicted is scarce in the data set
  - In such data sets, we may not have many examples of the one of the classes

- E.g. For the credit-riskiness data set, assume we have a test set of 1500 negative (non-risky $\boldsymbol{y} = \mathbf{0}$) examples, and only 50 positive (risky $\boldsymbol{y} = \mathbf{1}$) examples

- If our classifier correctly predicts 1300 negative samples, but only 10 of the positive examples, accuracy is?

  - $\frac{\mathbf{1300+10}}{\mathbf{1500+50}} \times \mathbf{100} = \mathbf{84.5}\%$

  - But is this classifier really good at predicting credit-riskiness?? (NO!)

Precision and Recall – The Problem

- Skew classes:
  - A case in which one of the classes to be predicted is scarce in the data set
  - In such data sets, we may not have many examples of the one of the classes

- E.g. For the credit-riskiness data set, assume we have a test set of 1500 negative (non-risky $\boldsymbol{y=0}$) examples, and only 50 positive (risky $\boldsymbol{y=1}$) examples

- In fact, considering the skewness of the classes, we can use a dummy function to get an even better accuracy:

```
def predict(X):
    return 0  #Just ignore X. Assume the class is 0
```

- Using this function to "predict" will give us an accuracy of?
  - $\frac{1500}{1500+50} \times 100 = 96.8\%$

- Accuracy is not a true reflection of underlying performance at all (in such cases)

Precision and Recall – The Problem

- We need some other way of evaluating our classifier that is not sensitive to the class distribution of examples – we're going to work towards it

- Four things that are important:
  - Of the cases that are actually **positive** (e.g. risky, fraudulent, cancerous etc.):
    1. how many are predicted **correctly** by the classifier?
    2. how many are predicted **wrongly** by the classifier?

  - Of the cases that are actually **negative** (e.g. reliable, not fraudulent, benign etc.):
    3. how many are predicted **correctly** by the classifier?
    4. how many are predicted **wrongly** by the classifier?

  - We don't want someone who is risky, fraudulent, cancerous etc. to slip through the cracks
  - We don't want to tell someone they are risky, fraudulent, cancerous etc. if they aren't

Precision and Recall – Confusion Matrix

- The four questions on the previous slide have specific "names" in ML literature:
  - Of the cases that are actually **positive**:
    1. how many are predicted **correctly** by the classifier?
    2. how many are predicted **wrongly** by the classifier?
  - Of the cases that are actually **negative**
    3. how many are predicted **correctly** by the classifier?
    4. how many are predicted **wrongly** by the classifier?
- We can represent this information in a table called a "Confusion Matrix"

	predicted	
actually	1.	2.
	4.	3.

Precision and Recall – Confusion Matrix

- The four questions on the previous slide have specific "names" in ML literature:
  - Of the cases that are actually **positive**:
    1. how many are predicted **correctly** by the classifier?
    2. how many are predicted **wrongly** by the classifier?
  - Of the cases that are actually **negative**
    3. how many are predicted **correctly** by the classifier?
    4. how many are predicted **wrongly** by the classifier?
- In the label True/False **Positive/Negative**:
  - **Positive/Negative** refers to the prediction
  - True/False refers to whether the prediction was right or wrong

		predicted	
		1	**0**
actually	**1**		
	0		

Precision and Recall – Confusion Matrix

- E.g. For credit riskiness, fill in the confusion matrix for:
  - 50 positive examples of which 10 were correctly predicted
  - 1500 negative examples of which 1300 were correctly predicted

- TP: FN:

- FP: TN:

		predicted	
		1	0
actually	1		
	0		

Precision and Recall – Confusion Matrix

- E.g. For credit riskiness, fill in the confusion matrix for:
  - 15 positive examples of which 7 were correctly predicted
  - 270 negative examples of which 200 were correctly predicted

- TP FN:

- FP TN:

		predicted	
		1	0
actually	1		
	0		

Precision and Recall – Confusion Matrix

- E.g. For cat detection, fill in the confusion matrix for:
  - 850 positive examples of which 420 were <u>IN</u>correctly predicted
  - 650 negative examples of which 360 were <u>IN</u>correctly predicted

- TP: FN:

- FP: TN:

		predicted	
		1	**0**
actually	**1**		
	0		

Precision and Recall – Confusion Matrix

- E.g. For cancer detection, fill in the confusion matrix for:
  - 160 positive examples of which 10 were <u>IN</u>correctly predicted
  - 1750 negative examples of which 600 were correctly predicted

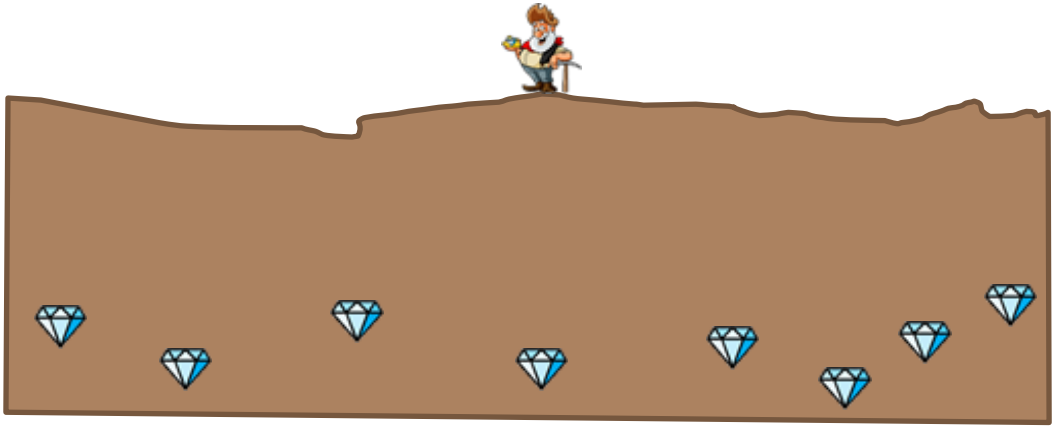
- TP: FN:

- FP: TN:

		predicted	
		1	0
actually	1		
	0		

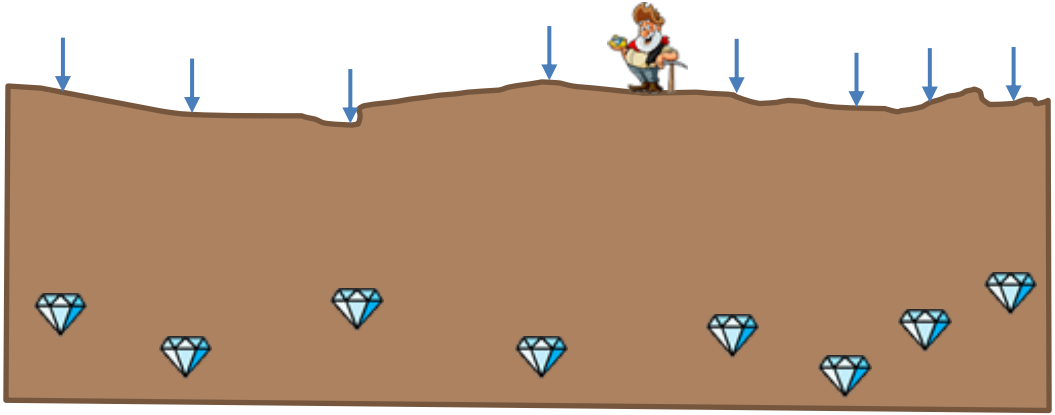
Precision and Recall – The Story

- You've employed a diamond expert to help you dig for diamonds
- The main question you want him to answer: where should you dig?
- When he says "dig here", do you trust him (precision)? How many of the diamonds will he find (recall)?



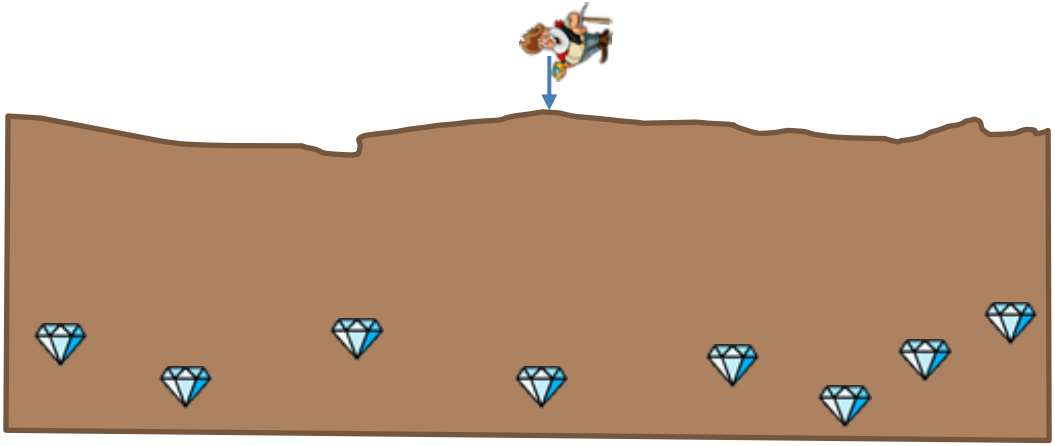
Precision and Recall – Interpretation

- Perfect Case:
  - Whenever he says "Diamond", it definitely is… DIG! (High Precision)
  - He finds ALL the diamonds! (High Recall)



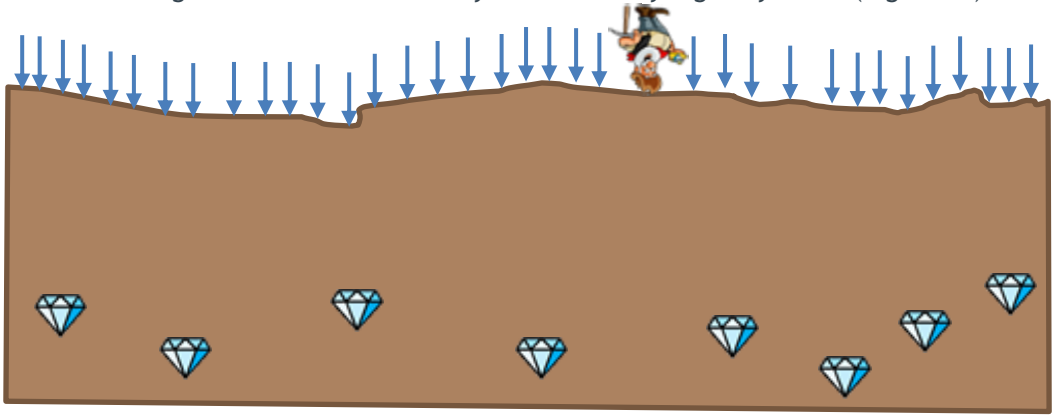
Precision and Recall – Interpretation

- Non-Ideal Case:
  - Whenever he says "Diamond", it definitely is… DIG! (High Precision)
  - He only finds very few diamonds! Is this gonna be sustainable?? (Low Recall)



Precision and Recall – Interpretation

- Non-Ideal Case:
  - When he says "Diamond", it is unlikely that it is… dig?? Rather ask your cat… (Low Precision)
  - He finds many/all the diamonds! In essence, he's just saying "DIG EVERYWHERE. You're bound to get all the diamonds that way…" Can we really dig everywhere? (High Recall)



Precision

- For any classifier, there are two critical questions:
  1. **Precision**: How **reliable / trustworthy / precise** are the classifier's **predictions**?
     - Can we trust the classifier's predictions?
     - When the classifier throws an alarm (i.e. $\boldsymbol{y} = \mathbf{1}$, risky, fraudulent, cancerous) how sure can we be that it actually is? We wouldn't want to react unnecessarily
     - When the classifier says all is fine (i.e. $\boldsymbol{y} = \mathbf{0}$, not risky, not fraudulent, not cancerous) how sure can we be that it actually is?

     - Another way of saying this: of the cases that the classifier **predicts** as **positive**, what proportion are **predicted correctly**?

Precision

- For any classifier, there are two critical questions:
  1. **Precision:** How **reliable / trustworthy / precise** are the classifier's **predictions**?
     - Another way of saying this: of the cases that the classifier **predicts** as **positive**, what proportion are **predicted correctly**?
     - Cases **predicted correctly:**
     - Cases **predicted** as **positive:**

(all of the cases predicted as positive)

		predicted	
		1	**0**
actually	**1**	True Positives	False Negatives
	0	False Positives	True Negatives

$$\text{Precision} = \frac{\text{True Positives}}{\text{Total Positive Predictions}} (\times 100)$$

$$= \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} (\times 100)$$

Recall

- For any classifier, there are two critical questions:
  2. **Recall:** How **comprehensive/thorough** is the classifier in finding cases that are actually positive?
     - How many of the actually positive cases can the classifier actually find?
     - How sure can we be that the classifier will find all of the positive cases?
     - How sure can we be that all of the positive cases (i.e. $\boldsymbol{y} = \mathbf{1}$, risky, fraudulent, cancerous) won't just slip through undetected?
     - When the classifier identifies a bunch of cases as being positive, how sure can we be that these are all of them?
     - We wouldn't want to miss positive cases which need to be reacted on

     - Another way of saying this: of the cases that are **actually positive**, what proportion are **predicted correctly**?

Recall

- For any classifier, there are two critical questions:
  2. **Recall:** How **comprehensive/thorough** is the classifier in finding cases that are actually positive?
     - Another way of saying this: of the cases that are **actually positive**, what proportion are **predicted correctly**?
     - Cases **predicted correctly**:
     - Cases **actually positive**:

(all of the actually positive examples)

		predicted 1	predicted 0
actually	1	True Positives	False Negatives
	0	False Positives	True Negatives

$$\text{Recall} = \frac{\text{True Positives}}{\text{Total Actual Positive Examples}} (\times 100)$$

$$= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} (\times 100)$$

Precision and Recall – The Problem (Revisited)

- E.g. For the credit-riskiness data set, assume we have a test set of 1500 negative (non-risky $\boldsymbol{y} = \mathbf{0}$) examples, and only 50 positive (risky $\boldsymbol{y} = \mathbf{1}$) examples

- If our classifier correctly predicts 1300 negative samples, but only 10 of the positive examples, accuracy was:
  - $\frac{1300+10}{1500+50} \times 100 = 84.5\%$

- What about precision?
  - $\frac{TP}{TP+FP} = \frac{10}{10+200} \times 100 \approx 5\%$

- And recall?
  - $\frac{TP}{TP+FN} = \frac{10}{10+40} \times 100 = 20\%$

		predicted	
		1	0
actually	1		
	0		

- Accuracy was deceiving, but not precision and recall. This classifier is awful

Precision and Recall – The Problem (Revisited)

- E.g. For the credit-riskiness data set, assume we have a test set of 1500 negative (non-risky $\boldsymbol{y=0}$) examples, and only 50 positive (risky $\boldsymbol{y=1}$) examples
- What about the dummy function?

```
def predict(X):
    return 0  #Just ignore X. Assume the class is 0
```

- Using this function to "predict" gave us an accuracy of:
  - $\frac{1500}{1500+50} \times 100 = 96.8\%$
- What's the precision of the dummy function?
  - $\frac{TP}{TP+FP} = \frac{0}{0+0} \times 100 = [NaN] \sim 0\%$
- What's the recall of the dummy function?
  - $\frac{TP}{TP+FN} = \frac{0}{0+50} \times 100 = 0\%$

		predicted	
		1	0
actually	1		
	0		

- Accuracy was deceiving, but not precision and recall

Precision and Recall

- E.g. For cat detection:
  - 275 positive examples of which 200 were <u>IN</u>correctly predicted
  - 260 negative examples of which 20 were <u>IN</u>correctly predicted

- Accuracy:
  -

- Precision:
  -

- Recall:
  -

		predicted	
		1	**0**
actually	**1**	75	200
	0	20	240

- Interpretation: (High Prec): When it says it's a cat, very good (79%) chance it is; (Low Rec) BUT it won't find a lot of these cats… a lot of the cats will simply go undetected (as non-cats)…

Precision and Recall

- E.g. For cancer detection:
  - 160 positive examples of which 10 were **IN**correctly predicted
  - 1750 negative examples of which 600 were correctly predicted

- Accuracy:
  -

- Precision:
  -

- Recall:
  -

actually \ predicted	1	0
1	150	10
0	1150	600

- Interpretation: (High Recall): It'll find almost all cancer cases and "treat" them; (Low Prec) BUT it'll also end up telling a lot of people who aren't cancerous that they are… and possible "treat" them too…

Precision and Recall – Interpretation

- Ideal Case:
  - High Precision – High Recall: Whenever it says it's a positive it most likely is; and it will successfully locate (almost) all positive cases (few/none will slip through)

- Non-Ideal Cases:
  - High Precision – Low Recall: Whenever it says it's a positive it most likely is; BUT it will locate very few (if any) positive cases (many will slip through)

  - High Recall – Low Precision: It will locate (almost) all positive cases (few/none will slip through); BUT when it says it's a positive, it will most likely not be one – we can't trust it

Model Evaluation For Classification

F1 Score

- Classification With Logistic Regression
  - Model Representation
  - Obtaining the Decision Boundary
  - Cost Function
  - Cost Function Minimization For Classification
  - Multi-Class Classification

- Model Evaluation For Classification
  - Accuracy
  - Precision and Recall
  - F1 score

F1 Score – The Problem

- We've now got two numbers (precision and recall)
  - They give us a very good understanding of our classifier's performance
  - BUT they are still TWO numbers
  - We need some way of combining the two numbers into one number

- E.g. For the credit-riskiness example: imagine we have three different hypotheses:
  1. $\boldsymbol{h_\theta(x) = g(\theta_0 x_0 + \theta_1 x_1)}$
  2. $\boldsymbol{h_\theta(x) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2)}$
  3. $\boldsymbol{h_\theta(x) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)}$

- Where $\boldsymbol{x_1}$ is the No of Defaults and $\boldsymbol{x_2}$ is the Total Debt Owed (R)

- How do these hypotheses compare??
  - One way: compute precision and recall

F1 Score – The Problem

- We get these results:

Approach	Precision (%)	Recall (%)
1	60	40
2	12	98
3	95	5

- Which model is best? We need some way of combining the two numbers into one number
- One possible way: taking the average??

F1 Score – The Problem

- We get these results:

Approach	Precision (%)	Recall (%)	Average (%)
1	60	40	50
2	12	98	55
3	95	5	50

- The average doesn't seem to be a good indicator for performance:
  - All these approaches appear to be almost exactly the same considering the average whereas:
    - Algorithm 1 is more balanced between precision and recall
    - Algorithm 2 simply says "Dig everywhere"
    - Algorithm 3 only finds one diamond and calls it a day

F1 Score – Definition

- We get these results:

Approach	Precision (%)	Recall (%)
1	60	40
2	12	98
3	95	5

- The F1 score (a.k.a the "harmonic mean" of precision and recall) provides a much more balanced measure:

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} (\times 100)$$

F1 Score – Characteristics

- We get these results:

Approach	Precision (%)	Recall (%)	F1 Score (%)
1	60	40	48
2	12	98	21
3	95	5	10

- Does the F1 Score reflect what we see in precision and recall?
  - Algorithm 1 is more balanced between precision and recall
  - Algorithm 2 simply says "Dig everywhere"
  - Algorithm 3 only "finds one diamond and calls it a day"

F1 Score – Interpretation

- When precision is 1 or 100% AND recall is 1 or 100% → F1 Score is 1 or 100%
    - In order to get a perfect F1 score, **BOTH** the precision **AND** recall have to be perfect
- If either precision is 0 OR recall is 0 → F1 Score is 0
    - If either the precision or recall suffers (regardless of whether or not the other is high), then the F1 score suffers accordingly too
    - There's no cheating the F1 score: having a very high precision/recall value while the other precision/recall value is very low will clearly reflect in the F1 score
- F1 score favours more balanced precision and recall scores

F1 Score

- E.g. For cat detection:
  - 275 positive examples of which 200 were <u>IN</u>correctly predicted
  - 260 negative examples of which 20 were <u>IN</u>correctly predicted

- Precision:
  - $\frac{\mathbf{TP}}{\mathbf{TP+FP}} = \frac{\mathbf{75}}{\mathbf{75+20}} \times \mathbf{100} = \mathbf{78.9\%}$

- Recall:
  - $\frac{\mathbf{TP}}{\mathbf{TP+FN}} = \frac{\mathbf{75}}{\mathbf{75+200}} \times \mathbf{100} = \mathbf{27.3\%}$

- F1 Score:
  -

		predicted	
		1	**0**
actually	**1**	75	200
	0	20	240

F1 Score

- E.g. For cancer detection:
  - 160 positive examples of which 10 were <u>IN</u>correctly predicted
  - 1750 negative examples of which 600 were correctly predicted

- Precision:
  - $\frac{\mathbf{TP}}{\mathbf{TP+FP}} = \frac{\mathbf{150}}{\mathbf{150+1150}} \times \mathbf{100} = \mathbf{11.5\%}$

- Recall:
  - $\frac{\mathbf{TP}}{\mathbf{TP+FN}} = \frac{\mathbf{150}}{\mathbf{150+10}} \times \mathbf{100} = \mathbf{93.8\%}$

- F1 Score:
  -

actually \ predicted	1	0
1	150	10
0	1150	600

Model Evaluation For Classification

- In general, if we have a number of different approaches e.g. different features that we want to compare, we:
    1. Train each of the approaches on the Train set
    2. Compute the accuracy, precision, recall and F1 score on the CV set: pick the one with the highest F1 score
    3. Compute the accuracy, precision, recall and F1 score on the Test set: quote this as the final performance of the approach

THE END

Of Logistic Regression Part 1

Content

- Classification With Logistic Regression
  - Model Representation
  - Obtaining the Decision Boundary
  - Cost Function
  - Cost Function Minimization For Classification
  - Multi-Class Classification

- Model Evaluation For Classification
  - Accuracy
  - Precision and Recall
  - F1 score

- Overfitting and Underfitting
  - Introduction
  - The Concept of Regularization
  - Regularized Linear Regression
  - Regularized Logistic Regression