MACHINE LEARNING

MEI/1

University of Beira Interior,
Department of Informatics

Hugo Pedro Proença,
hugomcp@di.ubi.pt, 2026/2027

Machine Learning

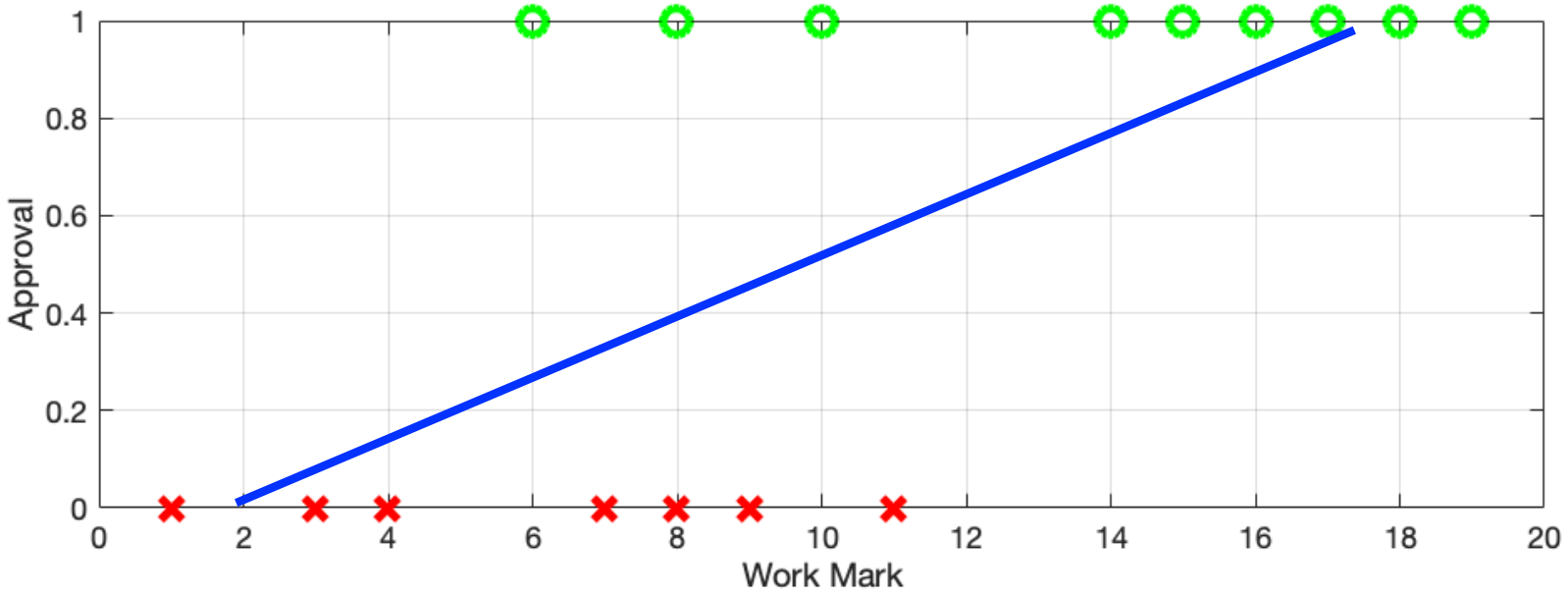
Week 03

Syllabus

- Supervised Learning
  - Classification
    - Logistic Regression

Students Performance

- Suppose that we are interested in predicting the **approval rate** of a class, based on the **students marks in the first practical work**.
  - Typically, students that get good marks in the first work, got approved at the course.
  - Students with very low marks at the first work tend to fail in the final examination.
- Hence, our machine learning model is expected to predict a **binary outcome** (**1: pass** vs. **0: fail**)



Students Performance

- In this kind of problems, the dependent variable assumes a reduced set of labels:
  - Emails: "is this a **spam** or **no spam** email"? $y \in \{0, 1\}$
  - Medical diagnosis: "is the patient **ill** or **healthy**" $y \in \{0, 1\}$
  - How will be the weather tomorrow?: "will it be **sunny**, **cloudy** or **rainy**"? $y \in \{0, 1, 2\}$
- In this case, a best fitting line is not enough
  - Even though this line will be the basis of our **classification model**

$$h_\theta(x) = \theta_1 . x + \theta_2$$

- In this kind of problems, the dependent variable assumes a reduced set of labels:
  - Emails: "is this a **spam** or **no spam** email"? $y \in \{0, 1\}$
  - Medical diagnosis: "is the patient **ill** or **healthy**" $y \in \{0, 1\}$
  - How will be the weather tomorrow?: "will it be **sunny**, **cloudy** or **rainy**"? $y \in \{0, 1, 2\}$
- In this case, a best fitting line is not enough
  - Even though this line will be the basis of our **classification model**

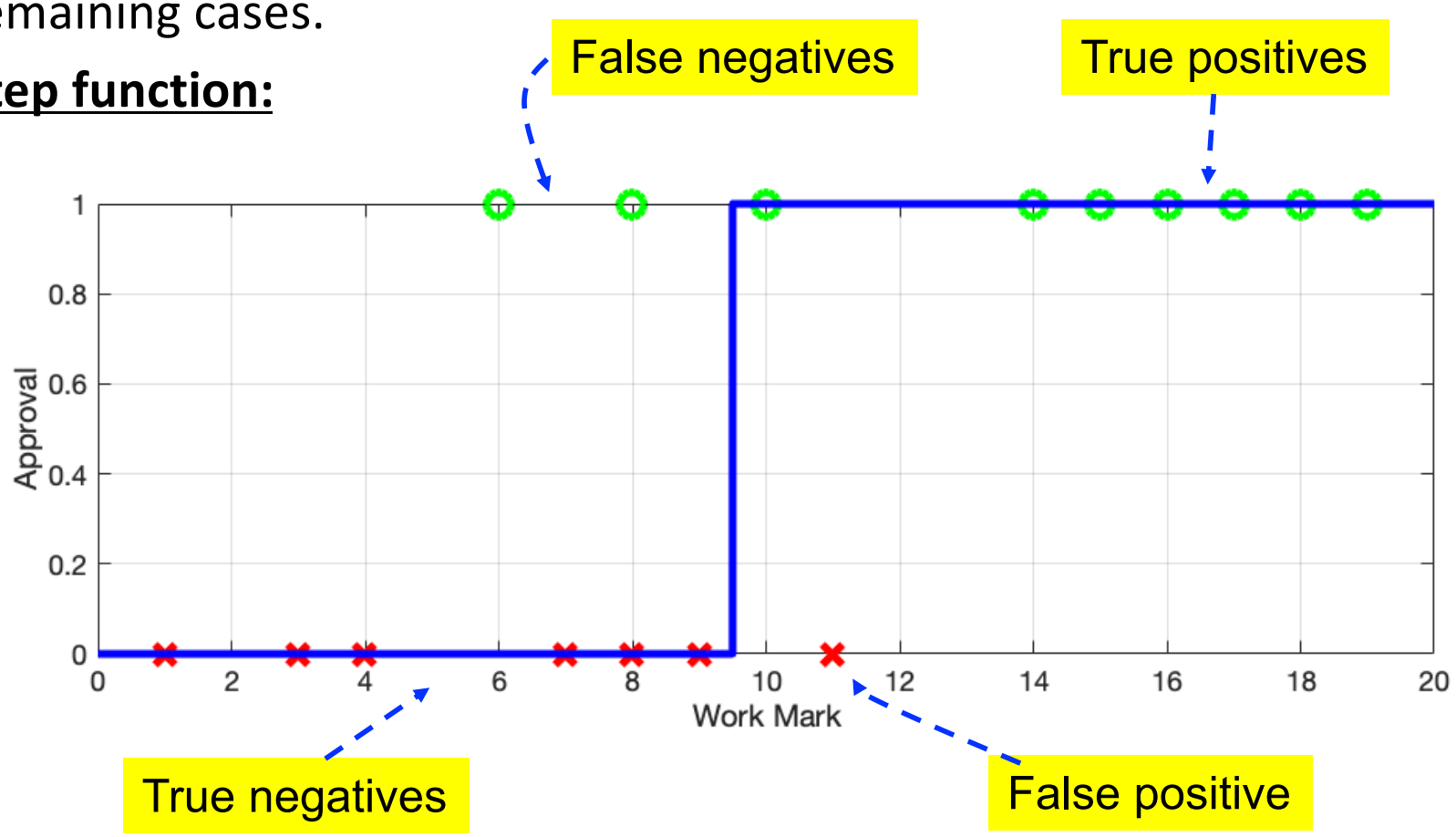
Logistic Regression: Classification

- The obvious idea will be to define a threshold at the classifier output $h_\theta(x)$, that binarizes the system response:
  - Typically, "0.5" would be the choice, for "*equal classification risk*"
    - It might be more dangerous to predict erroneously one class instead of other one.
    - For example, in a machine learning-based systems for medical diagnosis, classes have different risk.
      - Predict a "**malignant cancer**" on a "**healthy**" subject represents a unnecessary concern for the patient and would probably imply to perform additional (an unnecessary) exams.
      - However, provide a "**healthy**" response for a patient suffering of a "**malignant cancer**" might represent the patient dead sentence.
  - $f(x) = \begin{cases} 0, h_\theta(x) < 0 \\ 1, h_\theta(x) \geq 0 \end{cases}$
- Hence, the response of our classification system can be seen as a composition of two functions: $f = g \; o \; h$

  "f" is "g" after "h"
- *We have seen "h" before…*
  - *It is the best fitting line, of the previously seen regression problem*
- *…but what is "g"?*

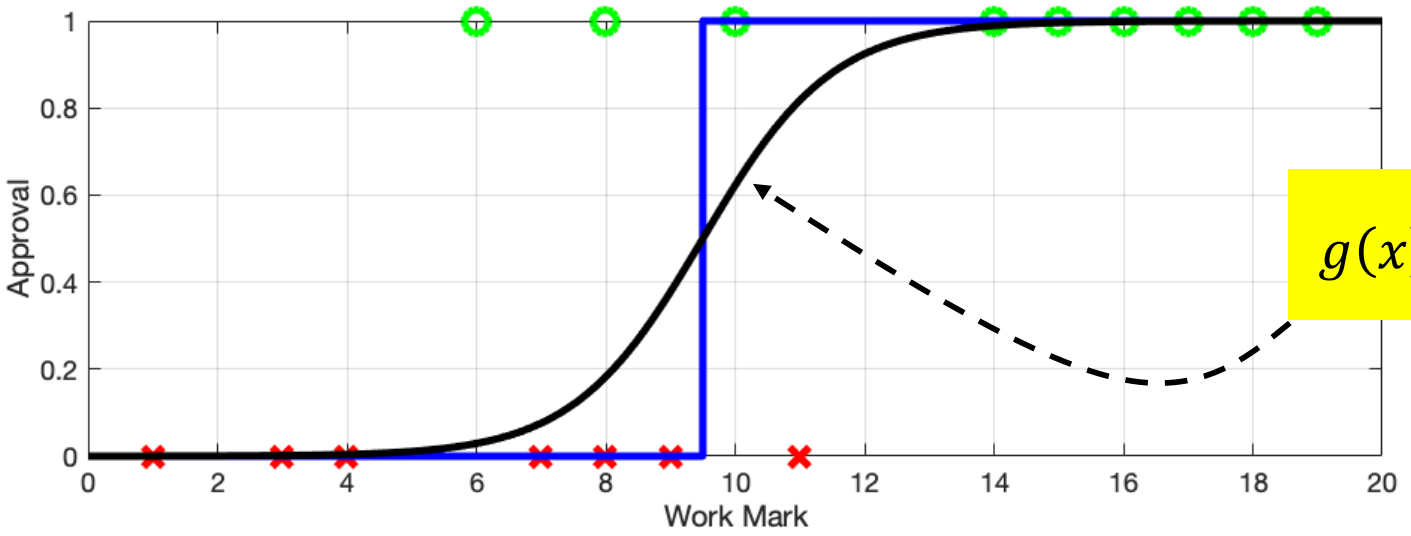
Logistic Regression: Classification

- Essentially, “g” performs a binarization of its input, and produces ”1” responses when the input is higher than some threshold, and “0” in the remaining cases.
- **Step function:**



Logistic Regression: Classification

- Assuming the step function as “g”, and $f = g\ o\ h$, obtaining the automatic optimal parameterization of “f” with respect to our data (i.e., machine learning) yields two problems:
  - **Problem 1**: “g” is **not differentiable**
    - It has not a continuous derivative at a single point
  - **Problem 2:** in every other points **“g” has derivative 0**
- The solution is to use a function is close to the step function, without suffering of the above described problems.
  - **Sigmoid Function**



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

Logistic Regression: Classification

- Using this composition of functions, the classification system is given by:

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

i.e.,

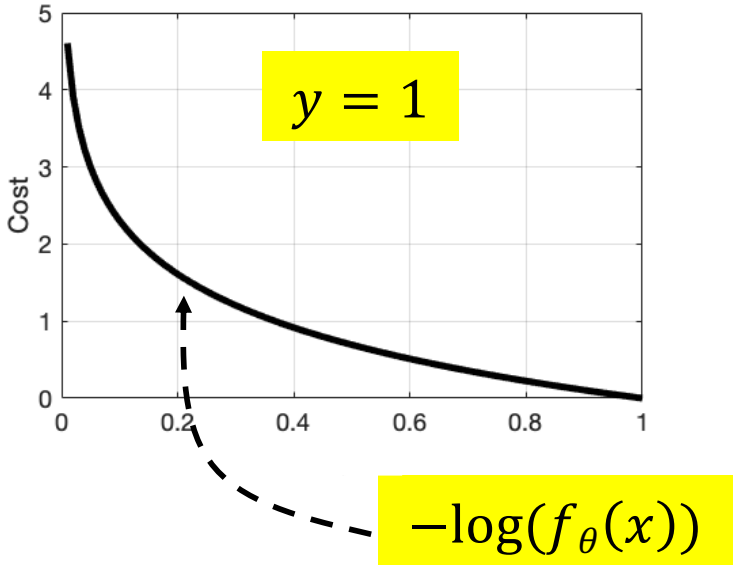
$$f_{\boldsymbol{\theta}}(x) = \frac{1}{1+e^{-(\theta_1 x + \theta_2)}}$$

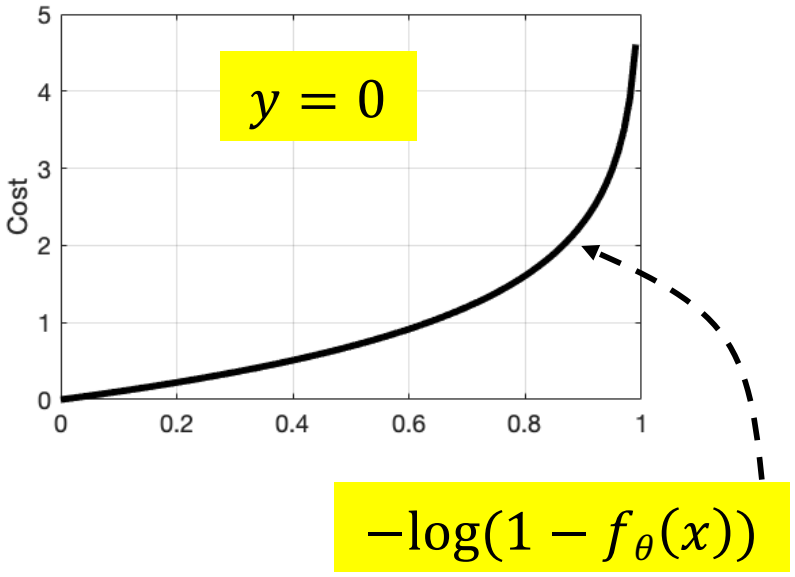
- The remaining problem is the same as in linear regression:
  - How to find the $\boldsymbol{\theta}$ optimal parameterization?

- According to the basic principles of Machine Learning, up to now we've only defined our model.
  - We define a "Cost" (Loss) function that measures "*how good it is an hypothesis*".

Binary Classification: Cost Function

- As for regression, the cost function will measure how well the model responses ($f_{\boldsymbol{\theta}}(\boldsymbol{x})$) resemble the "ground-truth" (y)
  - Intuitively, in cases where the system is supposed to output a "1" and the model predicts a "1", the cost should be 0.
  - The same thing should hold for "0" responses.
  - Also, the cost should grow in cases when the system response is far from the ground-truth.
    - The log() function is a good choice for representing the desired costs (losses)
    - It varies non-linearly with respect to the distance between the desired and actual responses
      - Attempts to avoid "**very wrong responses**".





Binary Classification: Cost Function

- Hence, the cost function for one instance is given by:
  - $Cost(f_\theta(x), y) = \begin{cases} -\log(f_\theta(x)) \,, & y = 1 \\ -\log\big(1 - f_\theta(x)\big) \,, & y = 0 \end{cases}$
- And the cost function for the whole dataset is given by the sum of the individual costs:

$$\mathbf{J}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\Big(Cost(f_\theta(x^{(i)}), y^{(i)})\Big)$$

- Considering that y can only assume 2 values (0 or 1), we have:

$$\mathbf{J}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} y^{(i)} \log\big(\boldsymbol{f_\theta}(x^{(i)})\big) + (1 - y^{(i)}) \log\big(1 - f_{\boldsymbol{\theta}}(x^{(i)})\big)$$

Logistic Regression: Optimization

- The optimization can be done exactly as in the linear regression case.
- Using the gradient descent strategy, it is required to find the derivatives of the cost function J() with respect to the $\boldsymbol{\theta}$ parameters:

$$\frac{\int}{\int \boldsymbol{\theta}} \mathrm{J}(\boldsymbol{\theta})$$

- In matrix form, we have:
  - $\mathrm{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$
  - $\log\left(f_{\boldsymbol{\theta}}(\boldsymbol{x})\right) = \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}\right)$
    $= -\log(1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}})$
  - $\log(1 - f_{\boldsymbol{\theta}}(\boldsymbol{x})) = -\theta\mathbf{x} - \log(1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}})$

$$\boldsymbol{\theta} = [\theta_0, \theta_1]^{\mathsf{T}}$$

$$\boldsymbol{x}^{(\mathrm{i})} = [\boldsymbol{x}^{(\mathrm{i})}, 1]^{\mathsf{T}}$$

Logistic Regression: Optimization

- Plugging the two simplified expressions in the original cost function:

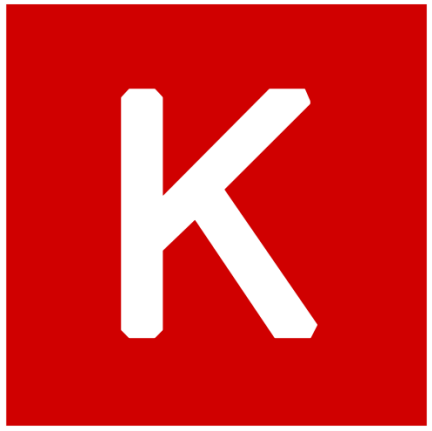
$$J(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} -y^{(i)} \log\left(1 + e^{-\boldsymbol{\theta}\boldsymbol{x}^{(i)}}\right) + (1 - y^{(i)})\left(-\boldsymbol{\theta}\boldsymbol{x}^{(i)} - \log\left(1 + e^{-\boldsymbol{\theta}\boldsymbol{x}^{(i)}}\right)\right)$$

- Which can be simplified to:

$$J(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\theta}\boldsymbol{x}^{(i)}(y^{(i)} - 1) - \log\left(1 + e^{-\boldsymbol{\theta}\boldsymbol{x}^{(i)}}\right)$$

- Obtaining the derivative of $J(\boldsymbol{\theta})$ with respect to each θ_j requires some effort and calculus.

- …That is where “Keras” (e.g.,) come to play!



Logistic Regression: Optimization

- When developing a ML model, the definition of the loss function J($\boldsymbol{\theta}$) is of paramount importance, as it will explicitly defines what is a "good model" and implicitly determines the direction of the learning process
- At this point, considering that the number of parameters of the model might be too large (e.g., for d=1M, we have θ={θ_1 , θ_2, θ_3 ... θ_{1M} }), we need to obtain $\frac{\partial J}{\partial \theta_i}$ i.e., the derivative of J($\boldsymbol{\theta}$) with respect to each parameter.
- At this point, the advantages of using an open-source library that provides a Python interface for this kind of models and learning processes, become evident. Among the different options (e.g., native workflows such as JAX, TensorFlow, or PyTorch), we can use Keras.
- Keras contains numerous implementations of commonly used machine learning building blocks such as layers, objectives, activation functions, optimizers, and a host of tools for working with image and text data to simplify programming for this kind of data structures.

Logistic Regression: Optimization

Minimal working example I

```
import numpy as np
import keras as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Libraries import

```
data = np.loadtxt("data.csv", delimiter=",")
X = data[:, :-1]
y = data[:, -1]
```

Data loading

Logistic Regression: Optimization

Minimal working example II

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Normalize Features

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
```

Split Disjoint Sets

```
model = Sequential()
model.add(Dense(1, input_shape=(X.shape[1],)))
model.add(Activation('sigmoid'))
```

Model Definition

Logistic Regression: Optimization

Minimal working example III

```
def J(y_true, y_pred):
        squared_difference = K.square(y_true - y_pred)
        return K.mean(squared_difference, axis=-1)
```

Define J()

```
model.compile(optimizer='adam', loss=J, metrics=['accuracy'])
```

Compile Model

```
model.fit(X_train, y_train, epochs=100, batch_size=8,
verbose=1)
```

Learn

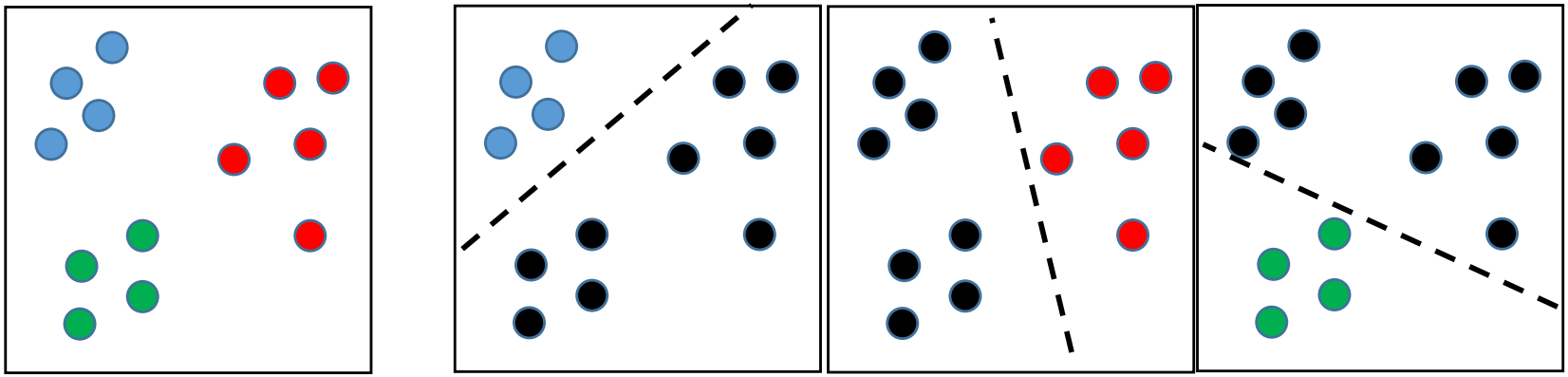
```
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {accuracy:.2f}")
```

Test

Logistic Regression: Multi-class

- Up to now, we've only considering binary classification problems.
- When the number of classes $c \geq 2$, the typical approach is to train "c" classifiers
  - In each classifier $f_\theta^{(i)}(x)$, instances of the ith class are considered positive examples, whereas instances of al the remaining classes are regarded as negative instances.
- During classification, we pick the class that produces the maximum output response, i.e.:

$$\max_i f_\theta^{(i)}(x)$$

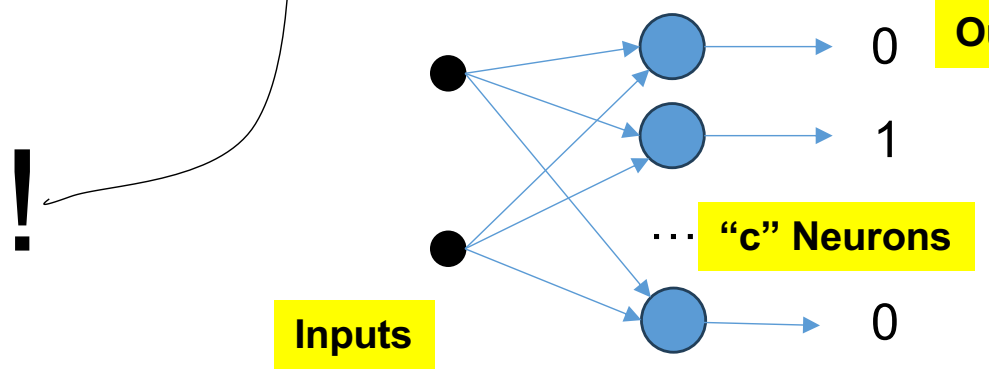


Logistic Regression: Multi-class

- Another possibility is to train "c" logistic regressors (which are in fact "neurons" of a neural network, and assume that they will produce one-hot encoding responses
  - If one instance belongs to the i^{th} class, the i^{th} neuron should produce a 1 response (a.k.a, "be activated"), whereas all the others should produce a 0 (not activated)
- The keras implementation is straightforward:

```
model = Sequential()
model.add(Dense(c, input_shape=(X.shape[1],)))
model.add(Activation('sigmoid'))
model.add(Softmax())
```

Model Definition



E.g.: For a problem with 5 classes, if one instance belongs to the forth class, the ground truth is: [0, 0, 0, 1, 0]