Introduction to Machine Learning

Lecture 3 - 2DV516: Logistic Regression

Amilcar Soares

amilcar.soares@lnu.se

Slides and used datasets are available in Moodle

A big thanks to Dr. Jonas Lundberg for providing most of the slides for this Lecture.

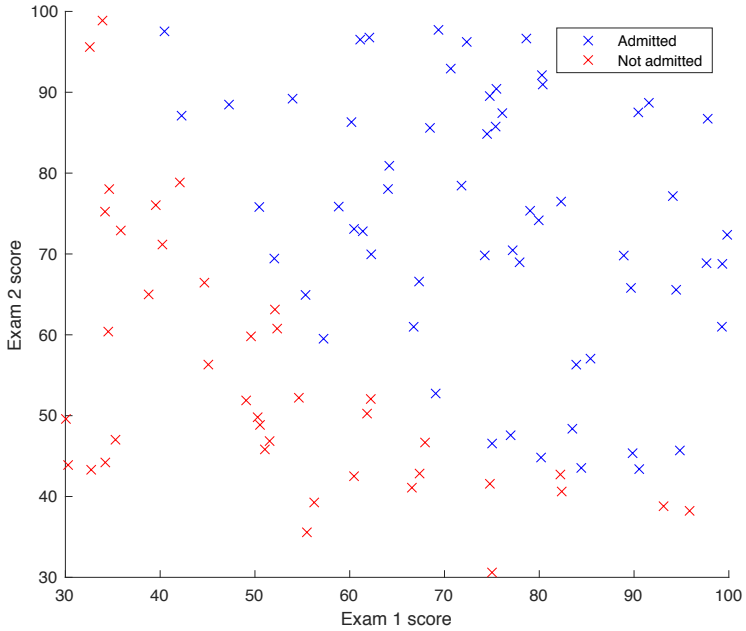
9 april 2024

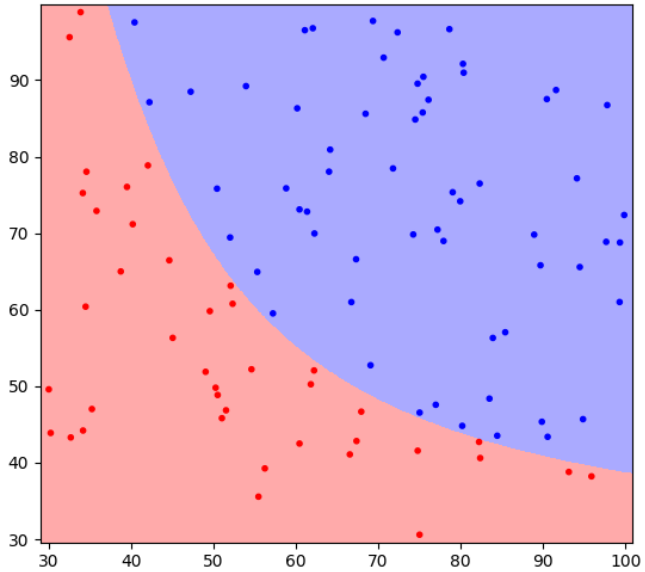
Agenda - Logistic Regression

- Binary Classification
- Logistic Regression
  - Sigmoid Function
  - Interpreting the Sigmoid Function
  - Cost function $J(\beta)$ for logistic regression
  - Gradient Descent
- Nonlinear Logistic Regression
- Plotting Decision Boundaries
- Multiclass Classification
- **Reading.** Lindholm, A., Wahlstrom, N., Lindsten, F., & Schon, T. B. (2022). Machine learning: a first course for engineers and scientists. Cambridge University Press.
  - **Chapter 3:** Section 3.2 - Classification and Logistic Regression. **Pages 45 to 57**.
- This lecture and slides focus on mathematics (high-level), concepts, and understanding of concepts.

Datasets admission.csv, iris2D.csv

Binary Classification





- **Task:** Learn a function/model $y = f(x), y \in 0, 1$ from a training set (x_i, y_i)
- **Goal:** Identify model $f(x)$ that can predict y_k for unseen instances x_k.
- **Expectation:** Precision will improve if more training samples are provided
- **Binary classification** since y is either 1/true/yes or 0/false/no.
- Separating 1 from 0 often involves a **decision boundary** (right-hand side figure)
- **Multiclass classification** $\Rightarrow$ y is $1, 2, 3, 4, \ldots, k$

Previously – Linear Regression (Degree 1)

- A dataset (x_i, y_i) with n observations
- Model (or hypothesis): $y = \beta_1 + \beta_2 x$ (Polynomial of degree 1)
- Problem: Find β_1 and β_2 that minimises the cost function

$$J(\beta_1, \beta_2) = \frac{1}{n} \sum_{i=1}^{n} ((\beta_1 + \beta_2 x_i) - y_i)^2$$

Solution 1 (Normal Equation): $\beta = (X_{ext}^T X_{ext})^{-1} X_{ext}^T y$:
Solution 2: Find β_1, β_2 using Gradient Descent

$$\beta_1^{j+1} = \beta_1^j - \gamma \frac{\partial J}{\partial \beta_1} = \beta_1^j - \frac{2\gamma}{n} \sum (\beta_1^j + \beta_2^j x_i - y_i)$$

$$\beta_2^{j+1} = \beta_2^j - \gamma \frac{\partial J}{\partial \beta_2} = \beta_2^j - \frac{2\gamma}{n} \sum x_i (\beta_1^j + \beta_2^j x_i - y_i)$$

Predict value y_p for given value x_p: $y_p = \beta_1 + \beta_2 x_p$ (Apply model)

Previously - Vectorized Solutions

All scenarios (linear, multivariate, polynomial) are treated the same way

- Dataset: $[X, y]$
- Extend X to fit scenario $X_{ext} = [ones(n, 1), X, ...]$
- Model: $y = X_{ext}\beta$
- Problem: Find β that minimises the cost function

$$J(\beta) = \frac{1}{n}(X_{ext}\beta - y)^T(X_{ext}\beta - y)$$

- Approximative solution: Find β using Gradient Descent

$$\beta^{j+1} = \beta^j - \gamma\nabla J(\beta) = \beta^j - \frac{2\gamma}{n}X_{ext}^T(X_{ext}\beta^j - y)$$

- Predict value y_p for given value x_p: $y_p = x_p\beta$ (Apply model)
- Apply feature normalization of X^i to speed up gradient descent
  1. Compute mean μ_i and standard deviation σ_i
  2. Compute normalized X^i as $X_n^i = (X^i - \mu_i)/\sigma_i$

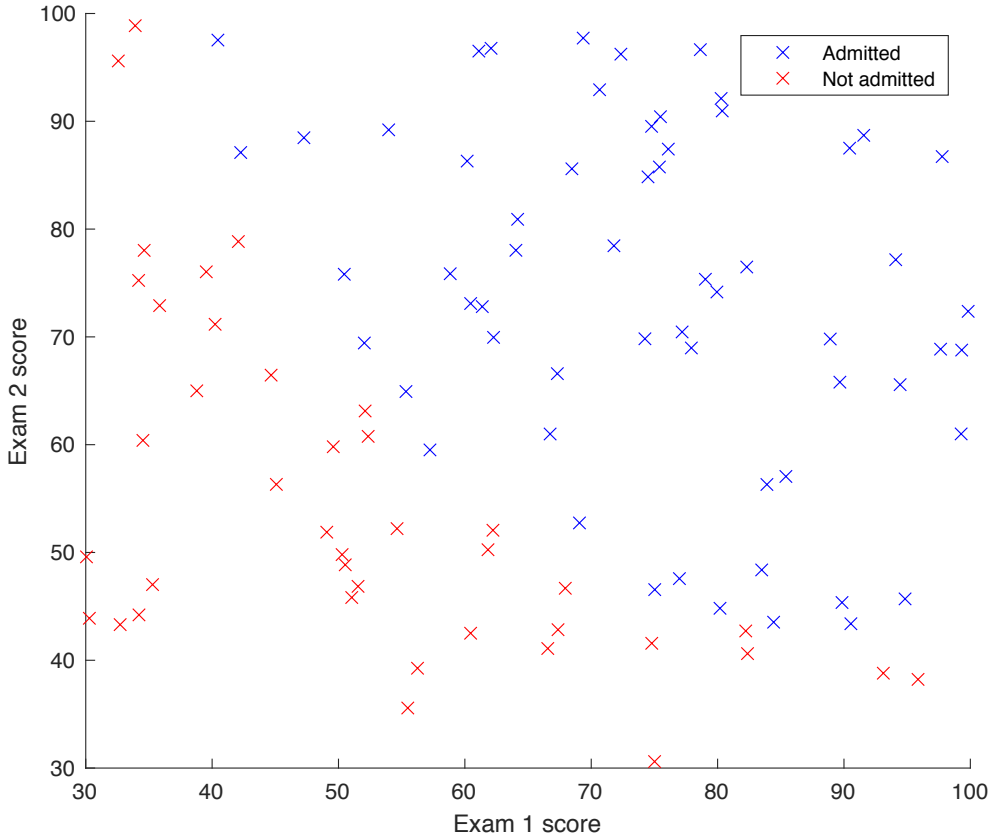
Example – Admission

All students are required to take two exams as a part of the admission procedure to a university.

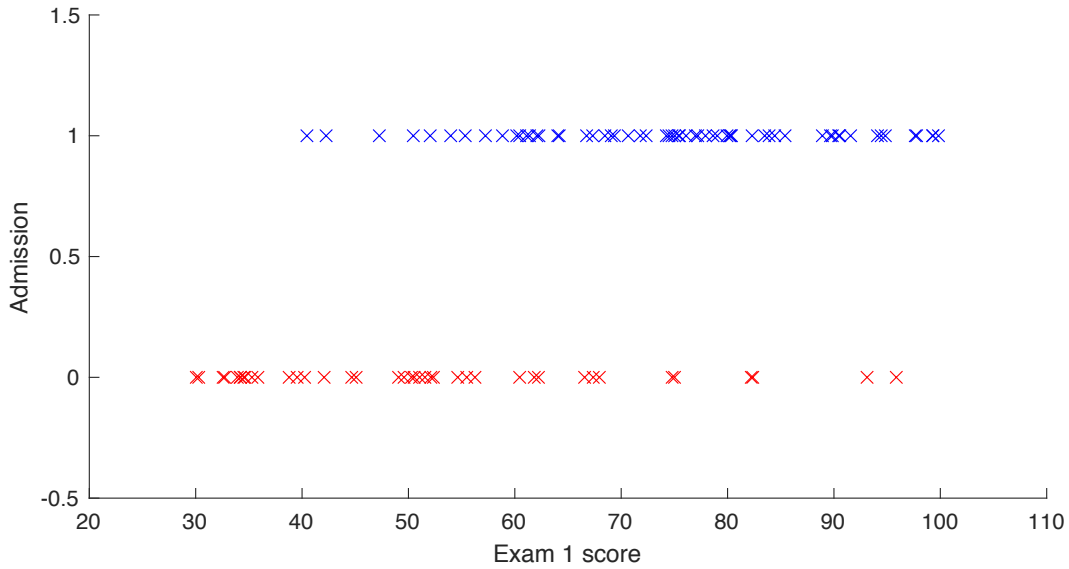
- Exam scores X_1, X_2
- Admission *yes* (1) or *no* (0)
- Observations: 100
- Problem: Find model $f(score_1, score_2) \rightarrow yes/no$

Q_1: What is the probability that a student with scores (45, 85) gets admitted?

Q_2: Visualize the corresponding decision boundary

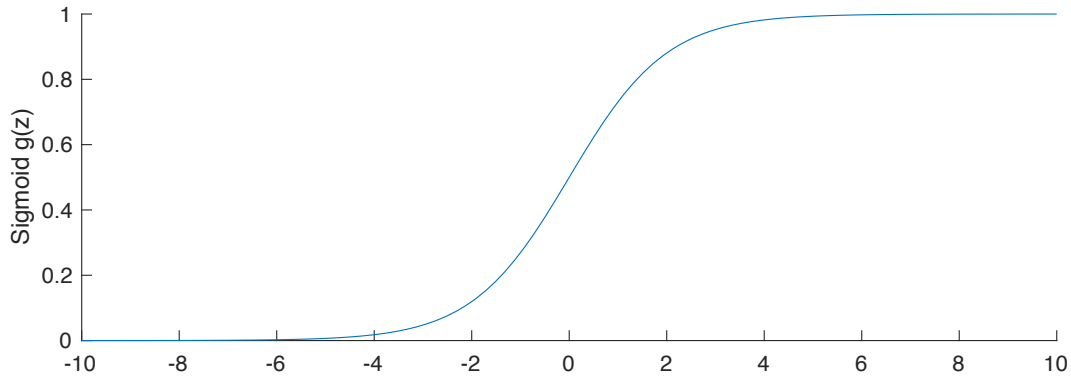


Admission – Only Test 1



We would like to find a function $f(x)$ that can be interpreted as **the probability** that a student with test score x will get admitted

Sigmoid Function (1)



Idea: Use training data to find β_0, β_1 such that $f_\beta(x)$ is the best possible fit where

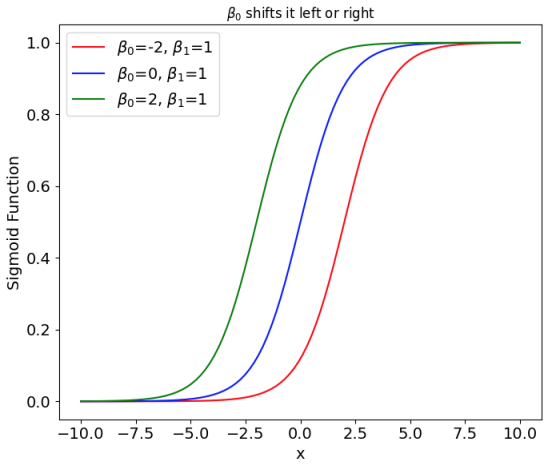
$$f_\beta(x) = g(X\beta) \qquad \text{(Our model, interpreted as probability)}$$

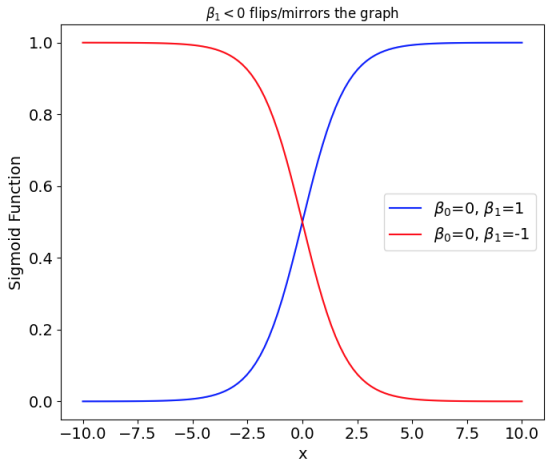
$$g(z) = \frac{1}{1+e^{-z}} \qquad \text{The Sigmoid Function}$$

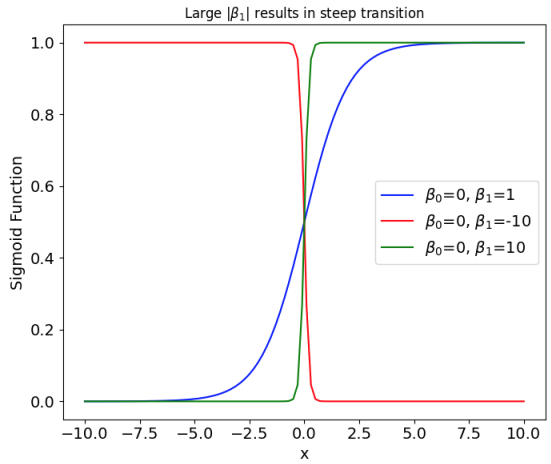
For high z : $\lim_{z\to\infty} g(z) = 1$(high positive values of z, the sigmoid function approaches 1)

For low z : $\lim_{z\to-\infty} g(z) = 0$(high negative values of z, the sigmoid function approaches 0)

Sigmoid Function (2)







$X\beta = \beta_0 + \beta_1 x$

β_0, β_1 changes the shape of $f_\beta(x)$. Roughly

- β_0 shifts it left or right
- $\beta_1 < 0 \Rightarrow$ 1 for large negative x (flips/mirrors the graph)
- Large $|\beta_1| \Rightarrow$ steep transition from 0 to 1

Maximum Likelihood (1)

$f_\beta(x)$ is interpreted as $p(x) \Rightarrow$ probability of being admitted

$$L = \prod_{i,y_i=1} p(x_i) \prod_{i,y_i=0} (1 - p(x_i)) \qquad \text{(Likelihood Function)}$$

It quantifies the likelihood of observing a given data set under a particular statistical model as a function of the model parameters.

Notice

- $y_i = 1 \Rightarrow$ positive cases $\Rightarrow p(x_i) \approx 1$
- $y_i = 0 \Rightarrow$ negative cases $\Rightarrow 1 - p(x_i) \approx 1$
- Good fit $\Rightarrow$ all factors are close to 1 $\Rightarrow$ Maximum L

The Likelihood function adapted to use our model $f_\beta(x_i)$ rather than $p(x_i)$

The goal is to find β_0, β_1 that maximizes the likelihood L

$$L(\beta_0, \beta_1) = \prod_{i,y_i=1} f_\beta(x_i) \prod_{i,y_i=0} (1 - f_\beta(x_i))$$

Maximum Likelihood (2)

Simple manipulations of the likelihood L

$$\begin{aligned} L(\beta_0, \beta_1) &= \prod_{i, y_i=1} f_\beta(x_i) \prod_{i, y_i=0} (1 - f_\beta(x_i)) \\ &= \prod_{i=1}^{n} f_\beta(x_i)^{y_i} \prod_{i=1}^{n} (1 - f_\beta(x_i))^{1-y_i} \\ &= \prod_{i=1}^{n} f_\beta(x_i)^{y_i} (1 - f_\beta(x_i))^{1-y_i} \end{aligned}$$

In the final steps we have made use of the fact $q^0 = 1$ and $q^1 = q$ for all q to merge all factors into a single product.

Hence, the goal is to find β_0, β_1 that maximizes the likelihood L

$$L(\beta_0, \beta_1) = \prod_{i=1}^{n} f_\beta(x_i)^{y_i} (1 - f_\beta(x_i))^{1-y_i}$$

The Logistic Cost Function (1)

First note:

- Find x maximizing $f(x) \Leftrightarrow$ Find x minimizing $-\log(f(x))$
  - $\log(f(x))$ has a max where $f(x)$ has a max
  - the minus sign turns the maximum problem into a minimization problem.

Recall.

$$L(\beta_0, \beta_1) = \prod_{i=1}^{n} f_\beta(x_i)^{y_i}(1 - f_\beta(x_i))^{1-y_i}$$

Hence, our goal is to find β_0, β_1 that minimizes the cost function

$$\begin{aligned} J(\beta_0, \beta_1) &= -\frac{\log(L(\beta_0, \beta_1))}{n} = \begin{bmatrix} log(ab) = \log(a) + \log(b) \\ log(a^b) = b\log(a) \end{bmatrix} \\ &= \dots \\ &= -\frac{1}{n}\sum_{i=1}^{n} y_i \log(f_\beta(x_i)) + (1 - y_i)\log(1 - f_\beta(x_i)) \end{aligned}$$

The Logistic Cost Function (Vectorized)

Non-vectorized Find β_0, β_1 that minimizes the cost function

$$J(\beta_0, \beta_1) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(f_\beta(x_i)) + (1 - y_i)\log(1 - f_\beta(x_i))$$

Vectorized Find β that minimizes the cost function

$$J(\beta) = -\frac{1}{n}(y^T \log(g(X\beta)) + (1 - y)^T \log(1 - g(X\beta))).$$

Notice

- y^T is a $1 \times n$ vector,
- $X\beta$ is a $n \times 1$ vector, and
- $\log(M)$ and the sigmoid $g(M)$ are matrix operations that applies $log(z)$ and $g(z)$ on each element in a matrix M

Finding β

We can not solve the equation $\nabla J(\beta) = 0$ to find β as we did in the Normal Equation. We must rely on numerical methods.

Gradient Descent

$$\begin{aligned} \beta_k^{j+1} &= \beta_k^j - \gamma \frac{\partial J}{\partial \beta_k} = \ldots = (\text{lengthy, dropped!}) \\ &= \beta_k^j - \frac{\gamma}{n} \sum_{i=0}^{n} (f_\beta(x^i) - y^i) x_k^i \end{aligned}$$

Vectorized

$$\beta^{j+1} = \beta_k^j - \frac{\gamma}{n} X^T (g(X\beta) - y).$$

Notice

- X^T is a $q \times n$ matrix where q is the number of columns in the extended matrix $[1, X_1, X_2, \ldots]$
- $g(X\beta)$ and y are $n \times 1$ vectors

Logistic Regression - Summary

Model:

$$\begin{aligned} f_\beta(x) &= g(X\beta) \\ g(z) &= \frac{1}{1+e^{-z}} \qquad \text{(The Sigmoid Function)} \end{aligned}$$

Cost function to minimize:

$$J(\beta) = -\frac{1}{n}(y^T \log(g(X\beta)) + (1-y)^T \log(1-g(X\beta))).$$

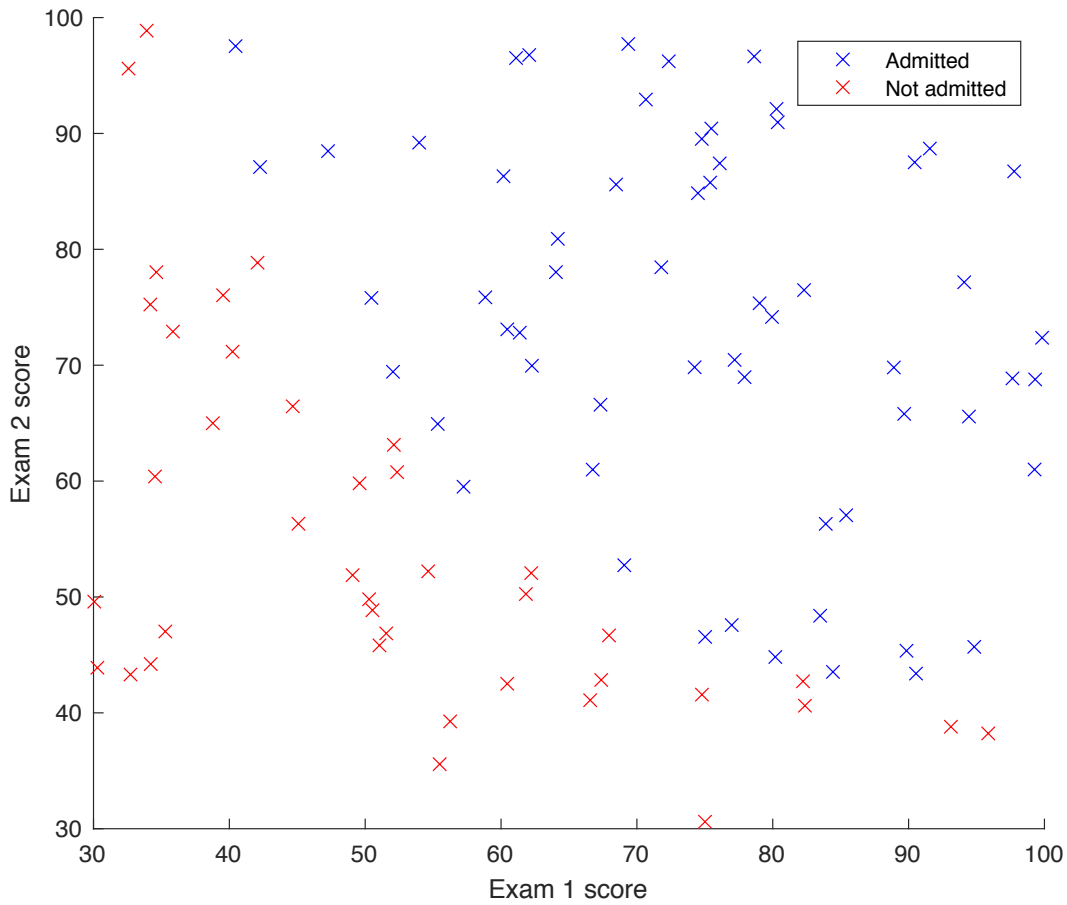
Find β using gradient descent

$$\beta^{j+1} = \beta^j - \frac{\alpha}{n} X^T (g(X\beta) - y).$$

Apart from changes in cost and gradient, very similar to gradient descent approach for linear regression

Admission Revisited

- Data X_1, X_2, y
- Linear Model:
  $f_\beta(x) = g(X\beta)$ with
  $X\beta = \beta_0 + \beta_1 X_1 + \beta_2 X_2$
- Algorithm
  1. Normalize $X \Rightarrow X_n$
  2. Construct
     $X_{ne} = [1, X_{1n}, X_{2n}]$
  3. Apply gradient descent with
     $\beta = (0, 0, 0)$ and
     suitable α, N
- Result
  1. $\beta_0 = 1.61$
  2. $\beta_1 = 3.77$
  3. $\beta_2 = 3.51$



Q: How to interpret the result?

Interpreting Admission

A simple Python code for prediction

```
#Predict student with score 45,85
S = np.array([45,85])
Sn = (S-mu)/sigma
Sne = np.c_[1,Sn[0],Sn[1]]    # Extended Sn
prob = logReg.sigmoid( np.dot(Sne, beta) )  # g(Sne*beta)
print(f"Admission probability with scores {S[0]}, {S[1]} is {prob[0]}")
```

A simple Python code for counting training errors

```
# Compute training errors
z =  np.dot(Xe, beta).reshape(-1,1)   # Compute X*beta
p = logReg.sigmoid( z )  # Probabilities in range [0,1]
pp = np.round(p)         # prediction
yy = y.reshape(-1,1)     # actual
print("Training errors: ",(np.sum(yy!=pp)))
```

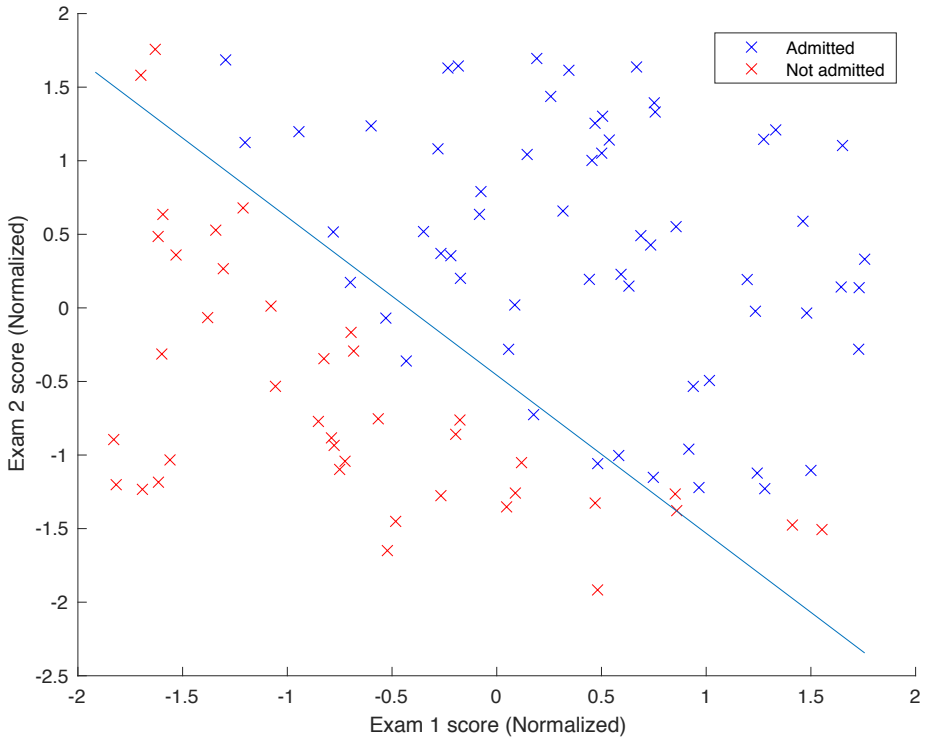
Notice: `reshape(-1,1)` turns an array of shape (say) `(100,)` into shape `(100,1)` required by my sigmoid implementation `logReg.sigmoid( p )`.

A 10 minute break?

Coffee Break ...

Admission - Straight Line Decision Boundary

- Data X_1, X_2, y
- Linear Model: $X\beta = \beta_0 + \beta_1 X_1 + \beta_2 X_2$
- Find $\beta_0, \beta_1, \beta_2$ minimizing cost function using gradient descent
- Result
  1. $\beta_0 = 1.61$
  2. $\beta_1 = 3.77$
  3. $\beta_2 = 3.51$
- Q: Visualize the corresponding decision boundary



Probability 0.5 gives the boundary between admission or not in our model

$p = 0.5 \Rightarrow g(X\beta) = 0.5 \Rightarrow X\beta = 0$ since $g(0) = 1/(1+1)$

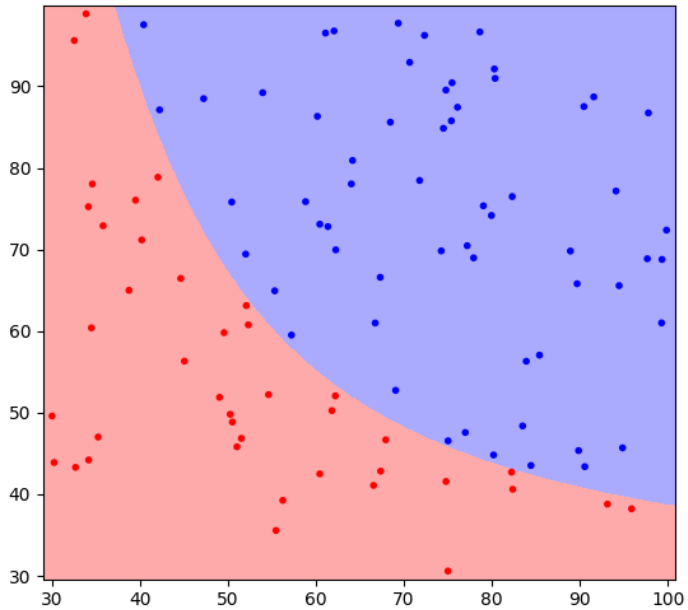
Hence $X\beta = \beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$ defines the decision boundary

Plot $x_2 = -(\beta_0 + \beta_1 x_1)/\beta_2$ for $x_1 \in [min(X_1), max(X_1)]$ to display the boundary

Admission – Nonlinear Boundaries

- The straight line fit for the Admission data was not optimal.
- A polynomial of degree 2 or higher would be better
- Model: $X\beta = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_1 X_2 + \beta_5 X_2^2$
- Algorithm
  1. Normalize $X \Rightarrow X_n$
  2. Construct $X_{ne} = [1, X_{1n}, X_{2n}, X_{1n}^2, X_{1n}X_{2n}, X_{2n}^2]$
  3. Gradient descent with $\beta = (0, 0, 0, 0, 0, 0)$ and suitable α, N
  4. Result: $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5$
- Q: How to plot the decision boundary?

Plotting Boundaries - The Meshgrid Approach



Basic Idea

- Divide x-axis into (say) 200 subintervals $30, ..., x_j, ..., 100$
- Divide y-axis into (say) 200 subintervals $30, ..., y_j, ..., 100$
- Classify each possible pair $(x_i, y_j) \Rightarrow 0$ or 1 (We have 40,000 such pairs!)
- Assign each position (x_i, y_j) a color (blue for 1, red for 0)

Plotting Boundaries in Python

```
h = .01  # step size in the mesh
x_min, x_max = X1.min()-0.1, X1.max()+0.1
y_min, y_max = X2.min()-0.1, X2.max()+0.1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))  # Mesh Grid
x1,x2 = xx.ravel(), yy.ravel() # Turn to two Nx1 arrays

XXe = A2.mapFeature(x1,x2,2)  # Extend matrix for degree 2

p = A2.sigmoid( np.dot(XXe, beta) )    # classify mesh ==> probabilities
classes = p>0.5                        # round off probabilities
clz_mesh = classes.reshape(xx.shape)   # return to mesh format

cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF']) # mesh plot
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])  # colors

plt.figure(2)
plt.pcolormesh(xx, yy, clz_mesh, cmap=cmap_light)
plt.scatter(X1,X2,c=y, marker='.',cmap=cmap_bold)
plt.show()
```

mapFeature(X_1,X_2, D)

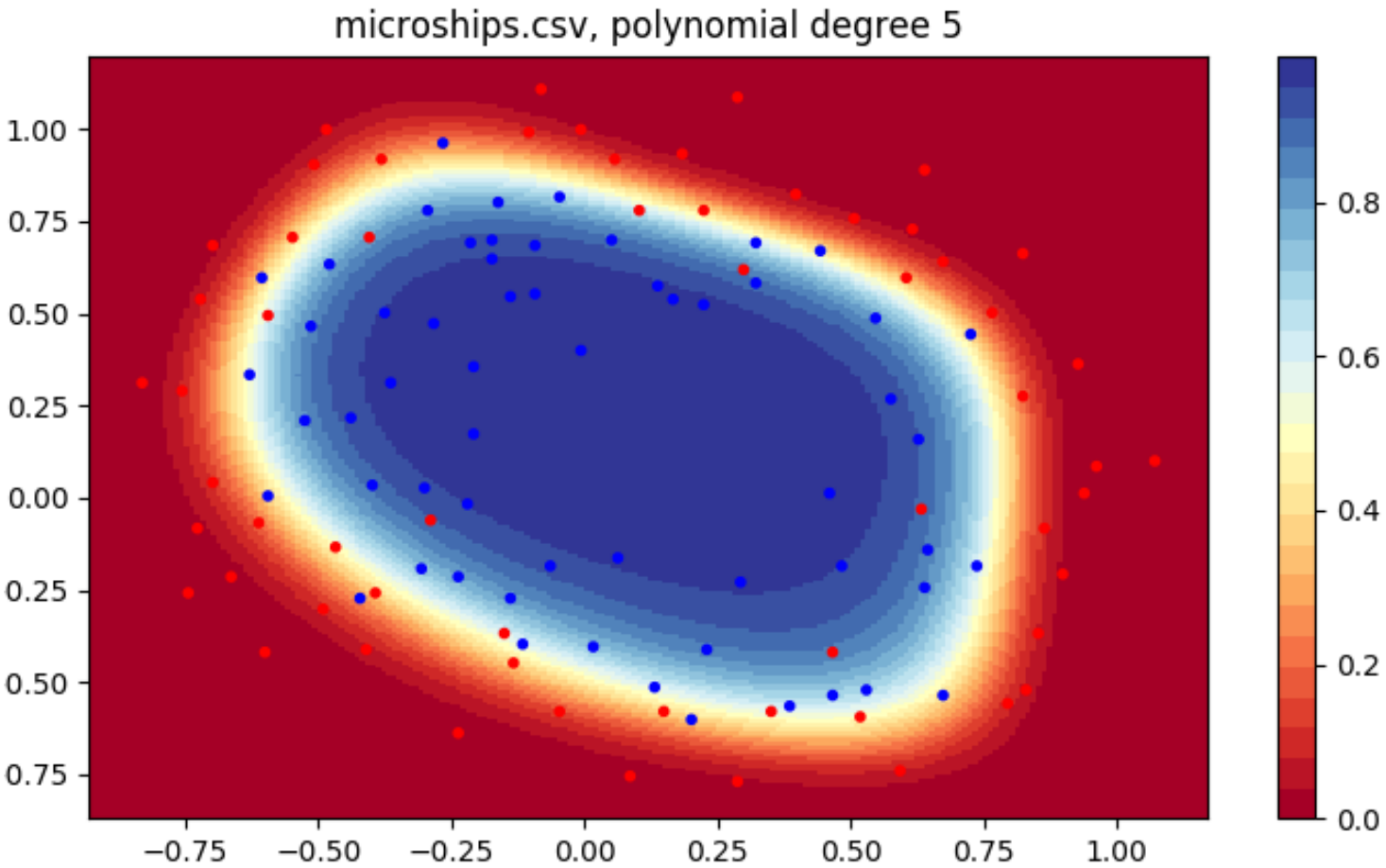
mapFeature maps X_1, X_2 to $[1, X_1, X_2, X_1^2, X_1X_2, X_2^2, \ldots]$ for any degree D
$\Rightarrow$ very useful for non-linear two-feature problems

```
def mapFeature(X1,X2,D):      # Pyton
    one = np.ones([len(X1),1])
    Xe = np.c_[one,X1,X2]         # Start with [1,X1,X2]
    for i in range(2,D+1):
        for j in range(0,i+1):
            Xnew =  X1**(i-j)*X2**j    # type (N)
            Xnew = Xnew.reshape(-1,1)  # type (N,1) required by append
            Xe = np.append(Xe,Xnew,1)  # axis = 1 ==> append column
    return Xe
```

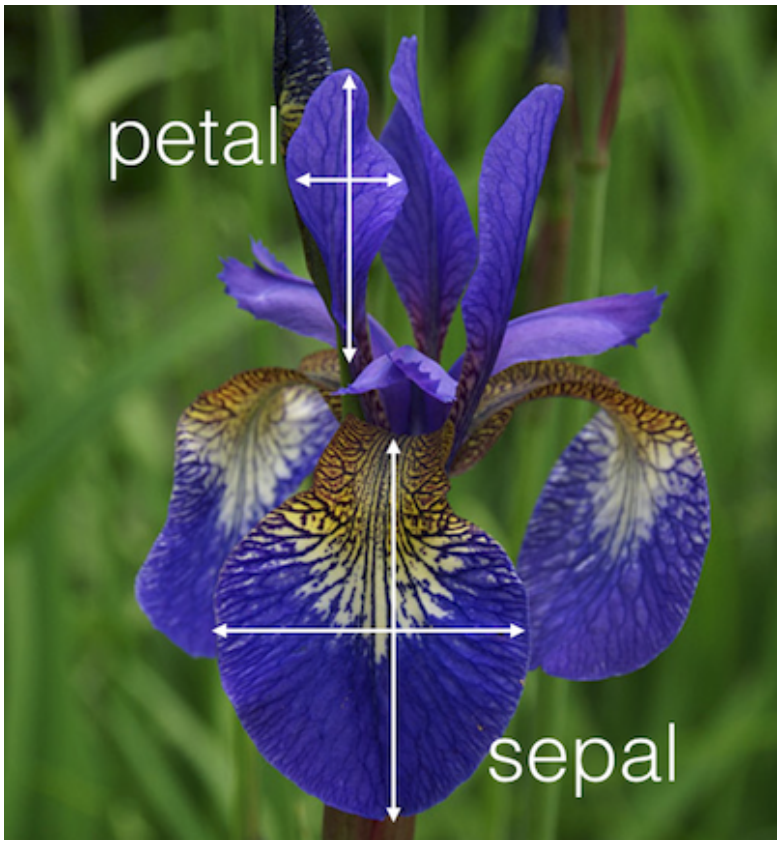
Summary: Algorithm for Binary Classification

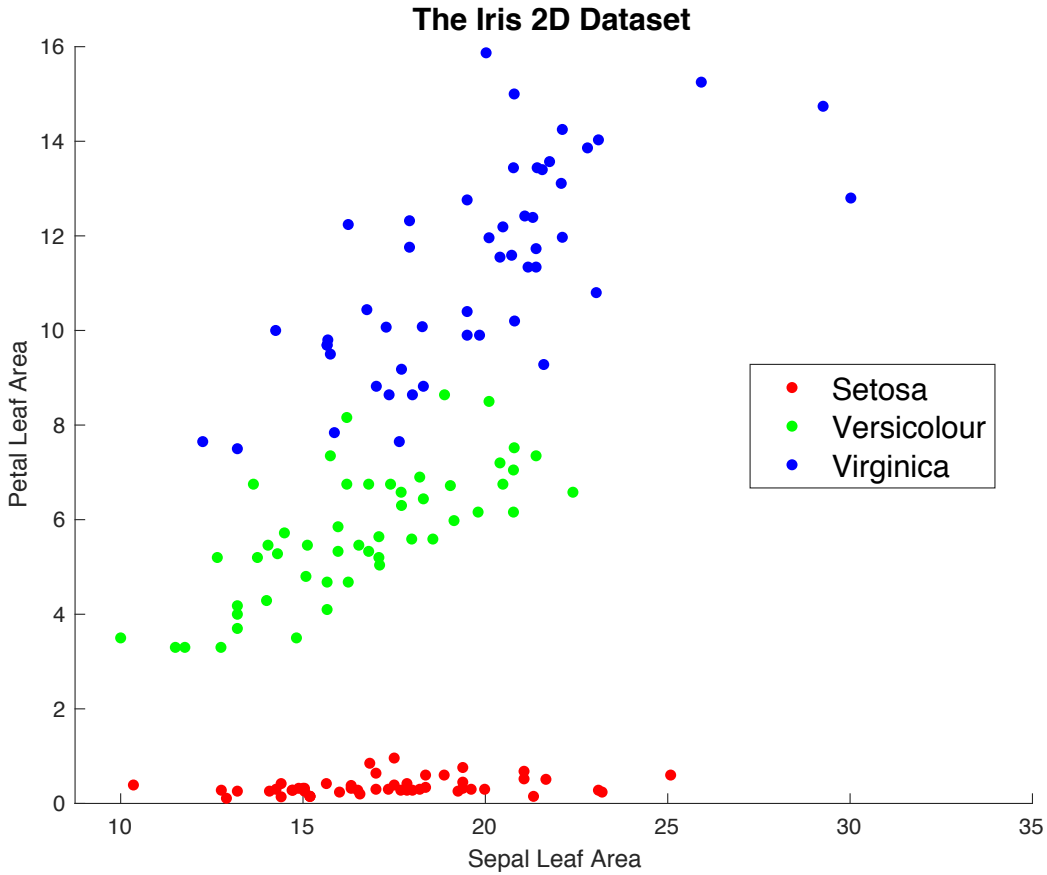
1. Assume labeled data: X_1, X_2, y
2. Normalize $X \Rightarrow X_n$
3. Select model $\Rightarrow$ polynomial of degree D
4. Create extended X_{ne} using `mapFeature(`X_1`,`X_2`, D)`
5. Find β using gradient descent
6. Plot decision boundary using the meshgrid approach

A nice decision boundary figure!



The Iris2D Dataset





Problem: Identify three subspecies of Iris based on petal and sepal leaf areas.

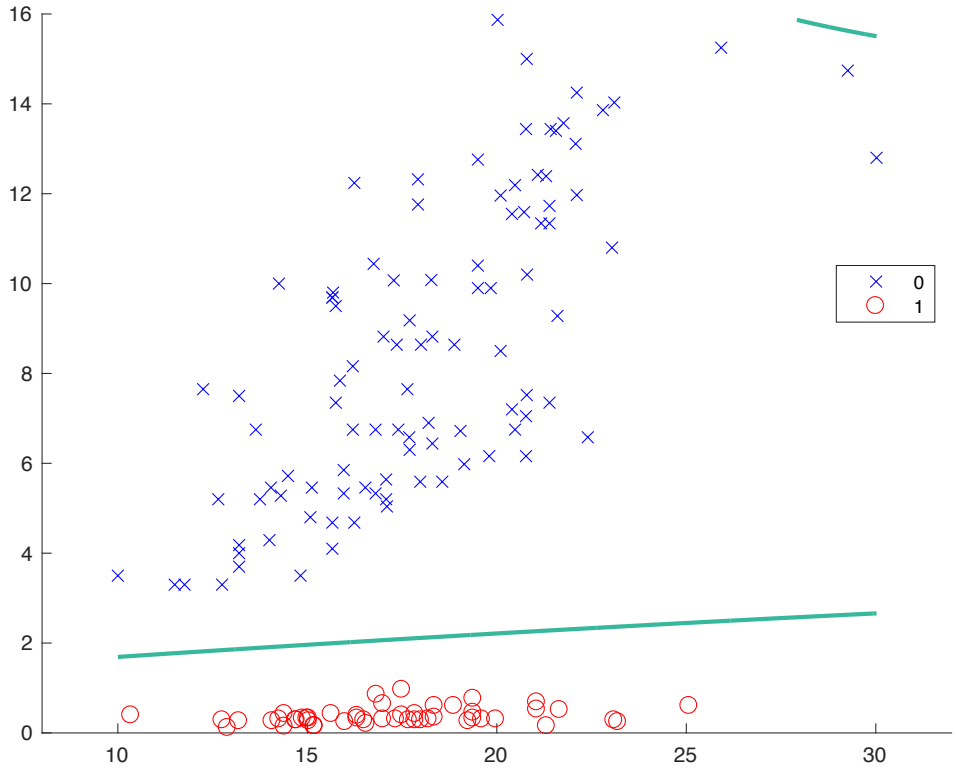
Multiclass – One-vs-all

- The Iris2D dataset has three classes: A (1), B (2), C (3)
  where A = Setosa, B = Versicolor, C = Virginica
- The sigmoid approach will not work! Only works in binary case.
- Basic Idea: One-vs-all
  - Train 3 binary classifiers $f_\beta^i(x)$, $i = 1, 2, 3$
  - For example, train $f_\beta^1(x)$ for A by using 1 for A, and 0 for B and C.
    $\Rightarrow$ A classifier that can recognise flowers of type A.
  - Classifying x: Apply all three classifiers $\Rightarrow$ for example,
    $p_A(x) = 0.8, p_B(x) = 0.4, p_C(x) = 0.05$,
  - Select the class with the highest probability $\Rightarrow$ class A
- Thus, we turn multiclass problems into several binary problems
- Rather tedious for datasets with many classes (e.g. MNIST).

Classifier 1: Recognise A (1)

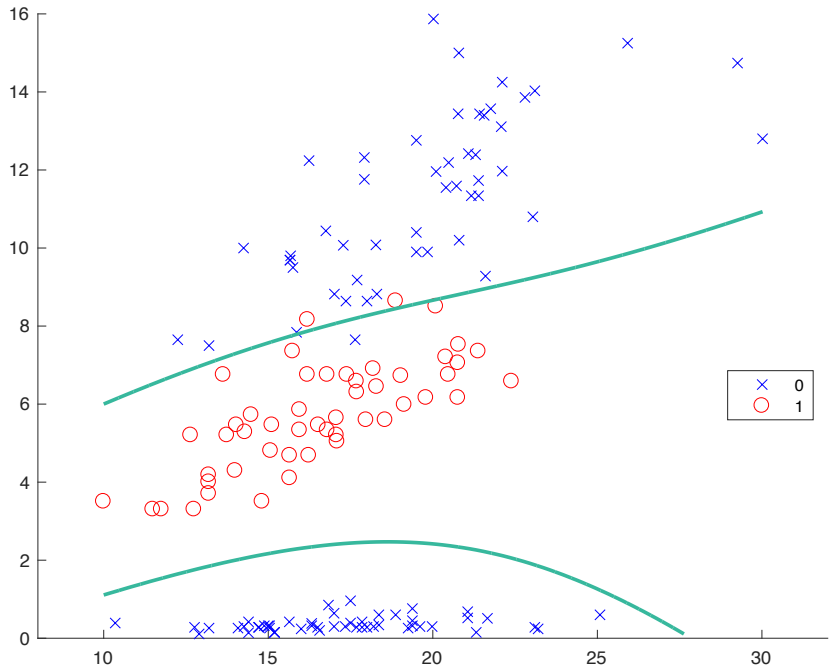
In what follows we use polynomials of degree 3.

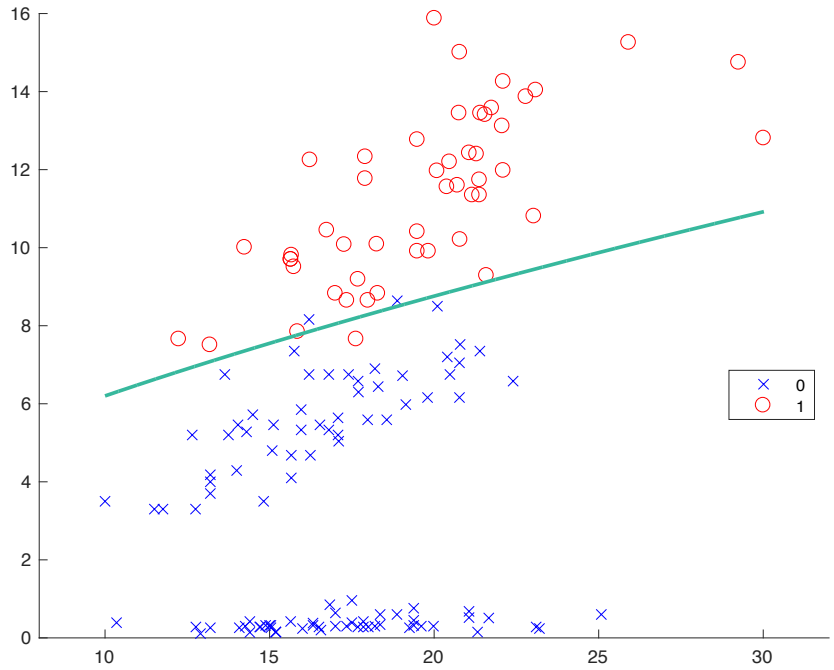
- Use 1 for A in y, and 0 for B and C
- Normalize X $\Rightarrow X_n$
- Extend X_n and apply gradient descent $\Rightarrow \beta_A$
- Plot decision boundary for β_A



Classifier 2 and 3: Recognise B (2) and C(3)

Repeat the same procedure for B and C $\Rightarrow \beta_B$ and β_C



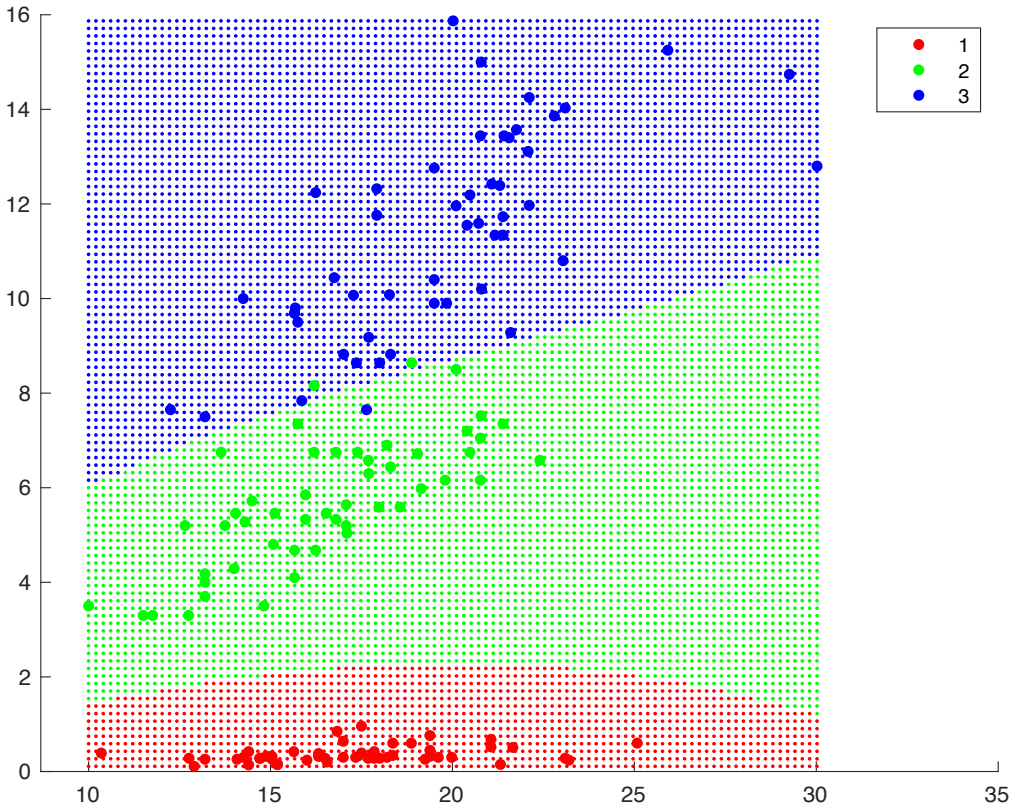


Iris2D – Classify Grid and and Plot

OK, we have three beta vectors β_A, β_B and β_C
and three classifiers $f_\beta^i(x)$, $i = A, B, C$

1. For each point (x_1^i, x_2^j) in the grid
   - 1.1 Apply all three classifiers $\Rightarrow$ $p_A(x_1, x_2), p_B(x_1, x_2), p_C(x_1, x_2)$
   - 1.2 Find largest probability and corresponding flower type:
     1 (A), 2 (B), or 3 (C)
   - 1.3 Associate the grid point (x_1^i, x_2^j) with corresponding value (1,2, or 3)
2. Plot the grid

Iris2D - Multiclass Decision Boundaries



This is a Matlab plot, expect similar plot in Python.

Notice: Three training errors, green (blue) data points inside the blue (green) territory

Multiclass - Summary

Assume a multiclass problem with (say) five classes A, B, C, D, E

One-vs-all

- We build five binary classifiers $f_\beta^i(x)$, $i = A, B, C, D, E$
- Train $f_\beta^A(x)$ for A by using 1 for A, and 0 fort he others
- Train $f_\beta^B(x)$ for B by using 1 for B, and 0 for the others
- ...
- Classifying x: Apply all five classifiers $\Rightarrow$ for example

$$p_A(x) = 0.7, p_B(x) = 0.4, p_C(x) = 0.05, p_D(x) = 0.82, p_E(x) = 0.08$$

- Select the class with the highest probability $\Rightarrow$ class D

One-vs-all is one possible approach for multiclass problems. Other approaches are possible and will be presented later on.

Assignment Summary - Lecture 3

Logistic Regression

- **Part 1.**
  - **Implement the Logistic Regression Model class. It must work for 2 classes only.**
  - **Implement the Non Linear Logistic Regression Model class.**
    This method should use the *mapFeatures* introduced in this class and works only for 2 features. It should work only for 2 classes only.
  - You must also implement the method *plotDecisionBoundary(X1, X2, y, model)* inside the Decision Boundary file. The code provided in class gives many hints but won't work for any model. It is up to you to modify it and make it produce the plot in a generalized way.
  - The code snippets for the models' cost function, gradient descent, forecast, and on were not given. It is part of the assignment for you to use Numpy properly to create those.
- **Part 2.**
  - Use these three classes you implemented to answer the questions from this part carefully.
  - Simple answers without an in-depth description of your findings won't receive full marks.
  - Try to describe your findings and what is going on. Whenever plots support your findings, discuss how and why you created them.