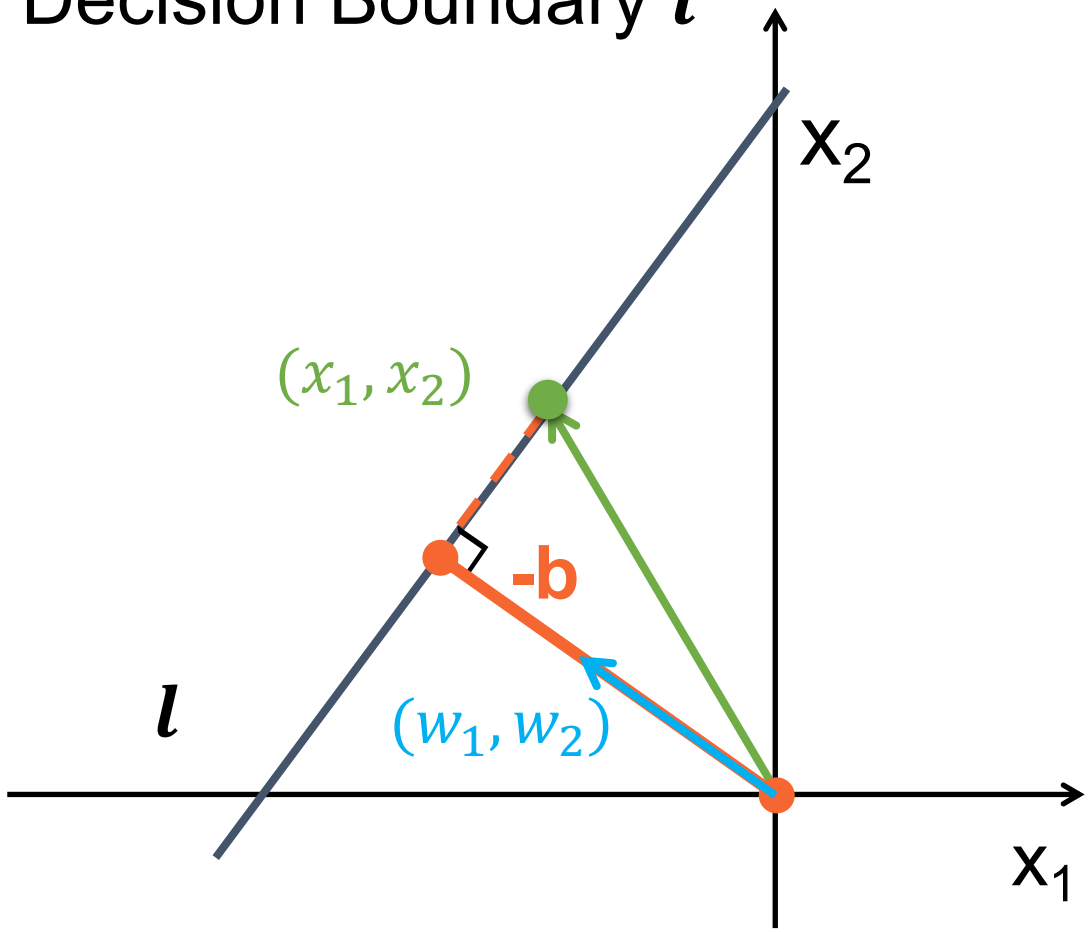
Support Vector Machine

Examples

Nianyi Li, Ph.D.
Assistant Professor
Clemson University

Review: Linear Classifier

- Decision Boundary $\boldsymbol{l}$



(w_1, w_2) is the normal vector of $\boldsymbol{l}$

(x_1, x_2) is feature points

-b is distance from origin to $\boldsymbol{l}$

$\boldsymbol{l}$: $\boldsymbol{w^T x = -b}$

$\boldsymbol{w^T x + b = 0}$

Recap (SVM)

Goal: Find the maximum classification margin

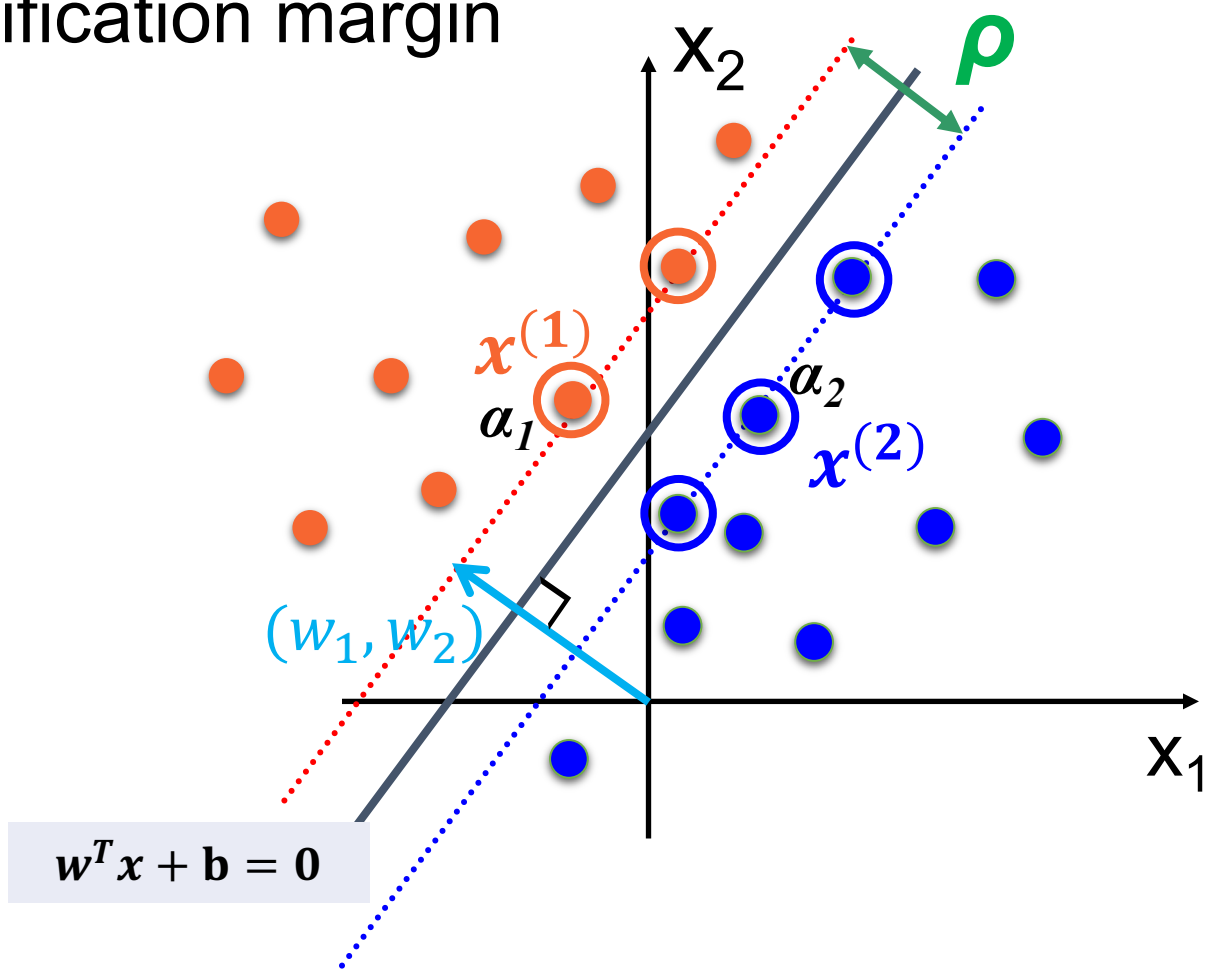
$$\boldsymbol{min}\ \|\vec{w}\|^2$$

$$\boldsymbol{w} = \Sigma \boldsymbol{\alpha_i} \boldsymbol{y}^{(i)} \mathbf{x}^{(i)}$$

$$b = \text{mean}(y_k - \boldsymbol{w}^{\mathbf{T}} \mathbf{x}_k)$$

for any $\alpha_k > 0$

$\boldsymbol{\alpha}$ are sparse vector with most elements of zero value



Review (SVM)

- Cost Function:

Find **w** and *b* such that

$\Phi(\mathbf{w}) = ||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w}$ is **minimized**

and for all $(\mathbf{x}^{(i)}, y^{(i)})$, $i=1..n$: $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1$

Lagrange duality

Find $\alpha_1 \ldots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y^{(i)} y^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle$ is maximized and

(1) $\Sigma\alpha_i y^{(i)} = 0$

(2) $0 \leq \alpha_i \leq C$ for all α_i

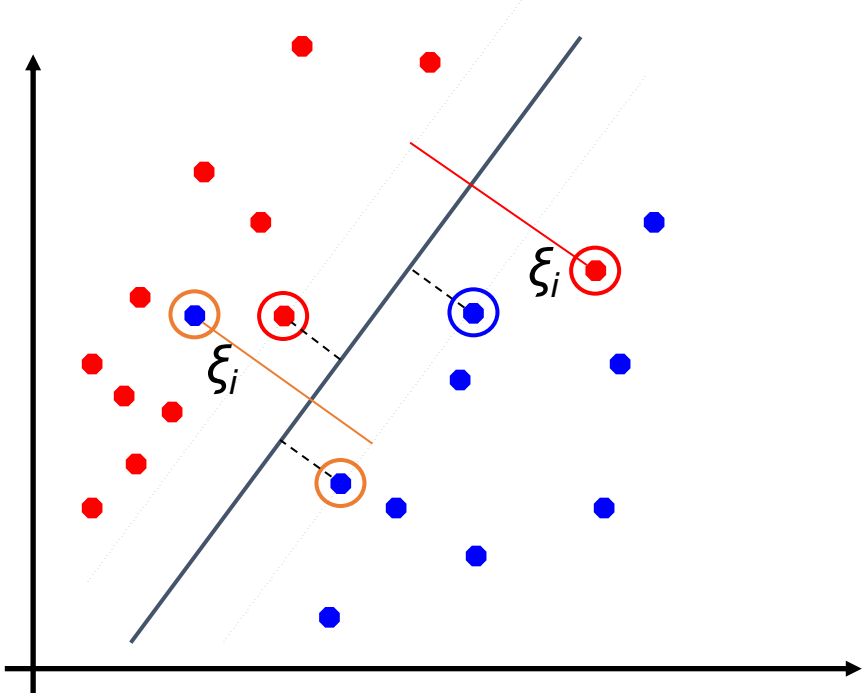
$\mathbf{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$

Gradient Decent

Kernel Functions

Soft Margin Classification

- What if the training set is not linearly separable?



Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*.

Soft Margin Classification

- The old formulation:

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$, $\xi_i \geq 0$

Parameter *C* can be viewed as a way to control overfitting: it "trades off" the relative importance of maximizing the margin and fitting the training data.

Soft Margin Classification

- Dual problem is identical to separable case
- Again, $\mathbf{x}_i$ with non-zero α_i will be support vectors.
- Solution to the dual problem is:

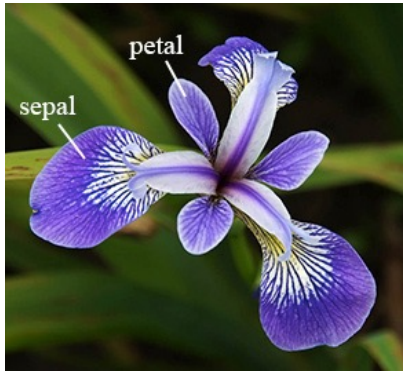
$$\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x}_i$$
$$b = y_k(1 - \xi_k) - \Sigma \alpha_i y_i \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_k \quad \text{for any } k \text{ s.t. } \alpha_k > 0$$

- We don't need to compute **w** explicitly for classification

$$f(\mathbf{x}) = \Sigma \alpha_i y_i \mathbf{x}_i^{\mathrm{T}} \mathbf{x} + b$$

About your midterm Project

- Project 1: plot the iris flower dataset
  - 150 records with attributes petal length, petal width, sepal length, and sepal width and type

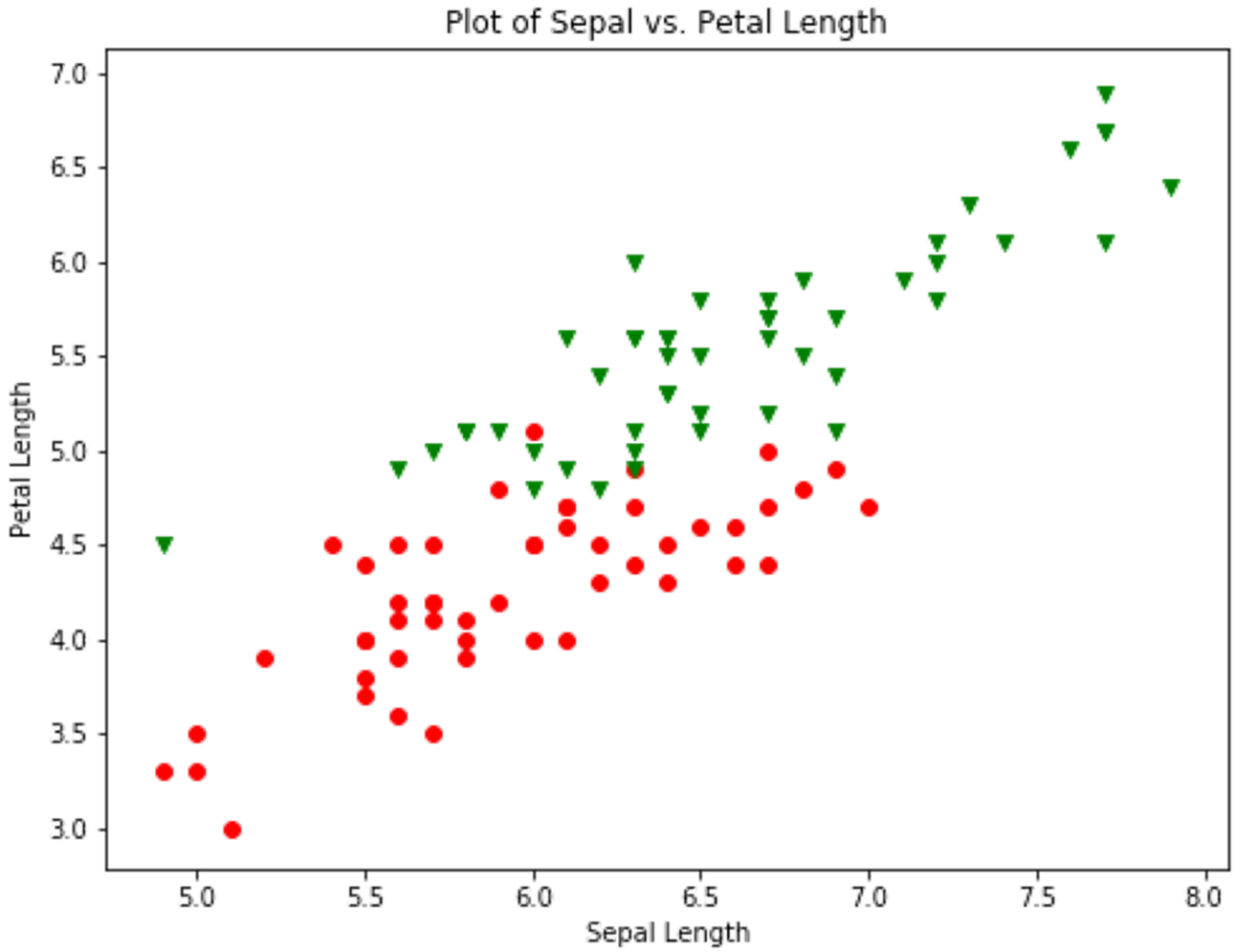


Let's implement Logistic Regression to determine if it can distinguish between Versicolor and Virginica flowers using just Sepal Length and Petal Length. (The same example we used for kNN)

Logistic regression with GD and two features

- Total Data file:
  - # of input:
    - 100
  - Features:
    - Sepal length
    - Petal length
  - Classes:
    - Versicolor
    - Virginica

100	2	
7	4.7	versicolor
6.4	4.5	versicolor

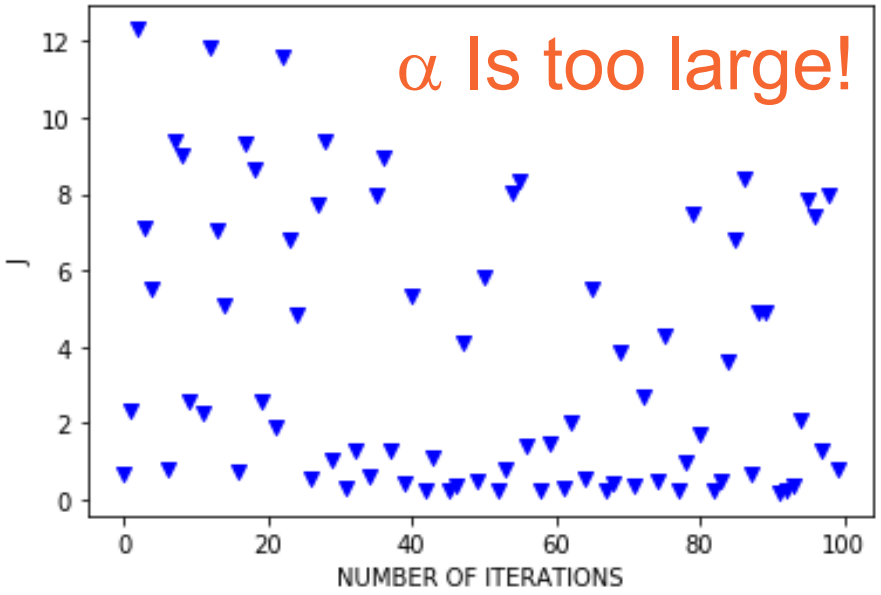


Logistic regression with GD and two features

- Data process
  1. Randomize the order of the records
  2. Replace the flower names with a code ("versicolor" = 0, "virginica" = 1)
  3. put 80% of the data in a training data set (**TrainingSet.txt**) and 20% of the data in a test set (**TestSet.txt**)
- GD steps (repeat certain times):
  1. Chose initial values for w's and α, and the total number of iterations
  2. Generate a plot of J versus number of iterations for initial evaluation of whether we have reasonable values for the weights.
  3. Try those weights with the TestSet.txt and compute, FN, FP, TN, TP.

Logistic regression with GD and two features

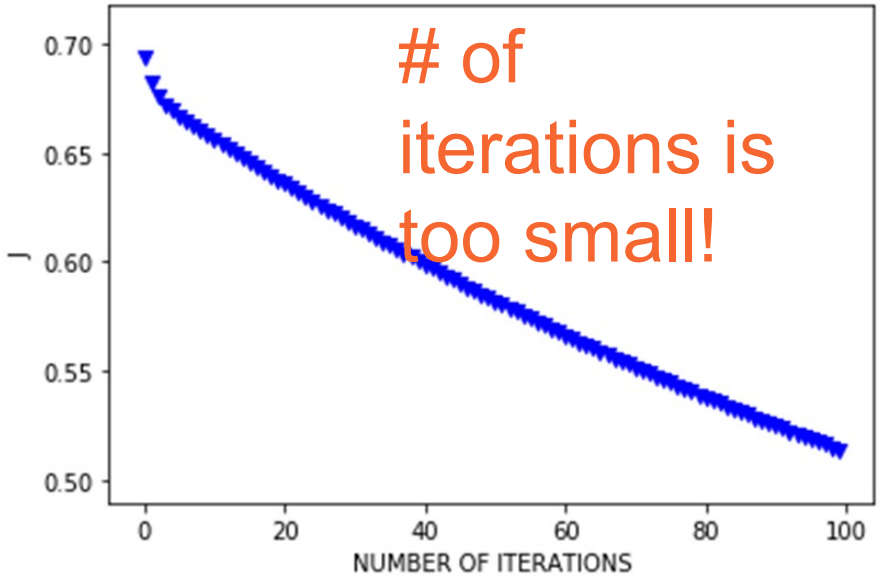
- Initialization
  - number of iterations = 100, α = 1 and w_0= 0, w_1 = 0 , w_2 = 0.
- After 100 iterations we get: $w = \begin{bmatrix} -2.88572301 \\ -5.11452316 \\ 10.44360012 \end{bmatrix}, J = 0.769361180363796$



Actual Class \ Predicted Class	N	Y
N	True Negatives (TN=0)	False Positives (FP=14)
Y	False Negatives (FN=0)	True Positives (TP=6)

Logistic regression with GD and two features

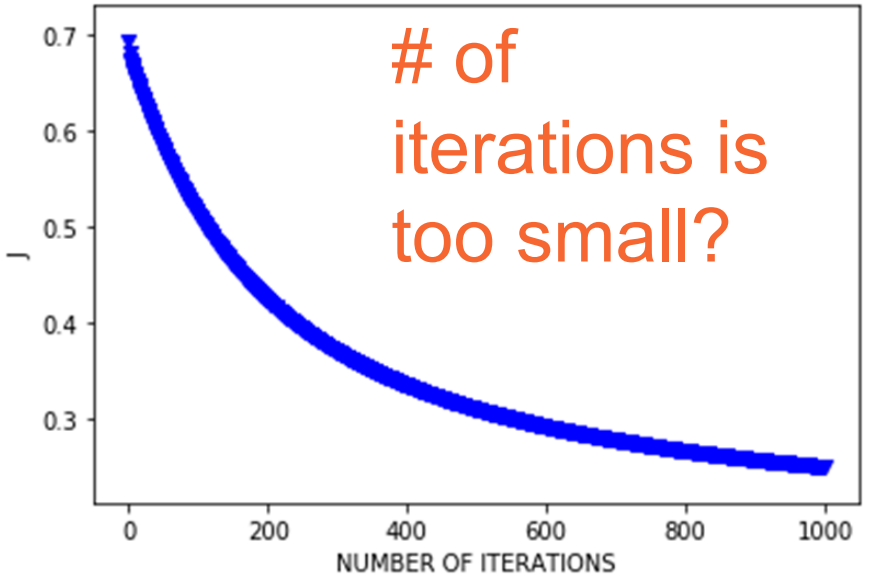
- Initialization
  - number of iterations = 100, $\alpha = 0.1$ and $w_0 = 0$, $w_1 = 0$, $w_2 = 0$.
- After 100 iterations we get: $w = \begin{bmatrix} -0.31979309 \\ -0.70181083 \\ 1.02304884 \end{bmatrix}, J = 0.513945631739894$



Actual Class \ Predicted Class	N	Y
N	True Negatives (TN=9)	False Positives (FP=5)
Y	False Negatives (FN=0)	True Positives (TP=6)

Logistic regression with GD and two features

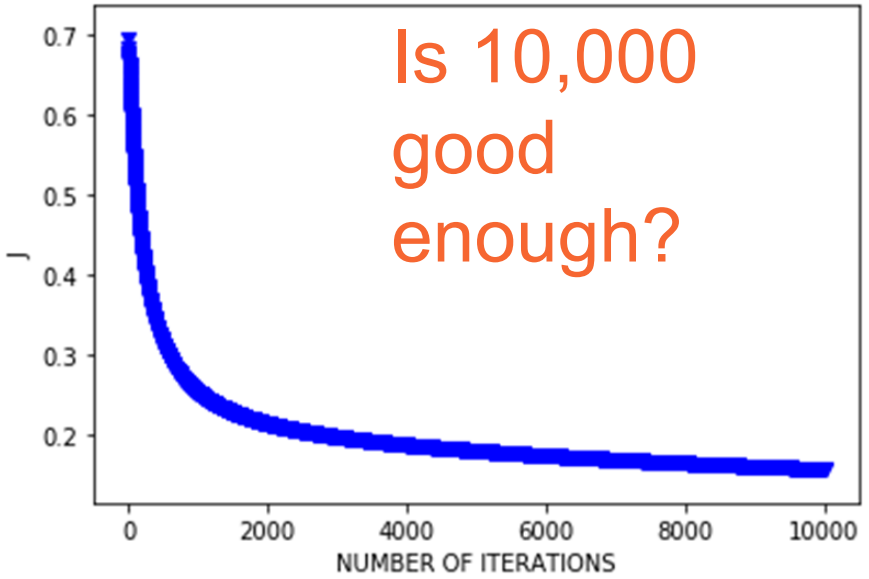
- Initialization
  - number of iterations = **1000**, α = 0.1 and w_0= 0, w_1 = 0 , w_2 = 0.
- After 1000 iterations we get: $w = \begin{bmatrix} -1.93807308 \\ -3.10019015 \\ 4.40804456 \end{bmatrix}, J = 0.250968703120496 2$



Actual Class \ Predicted Class	N	Y
N	True Negatives (TN=12)	False Positives (FP=2)
Y	False Negatives (FN=0)	True Positives (TP=6)

Logistic regression with GD and two features

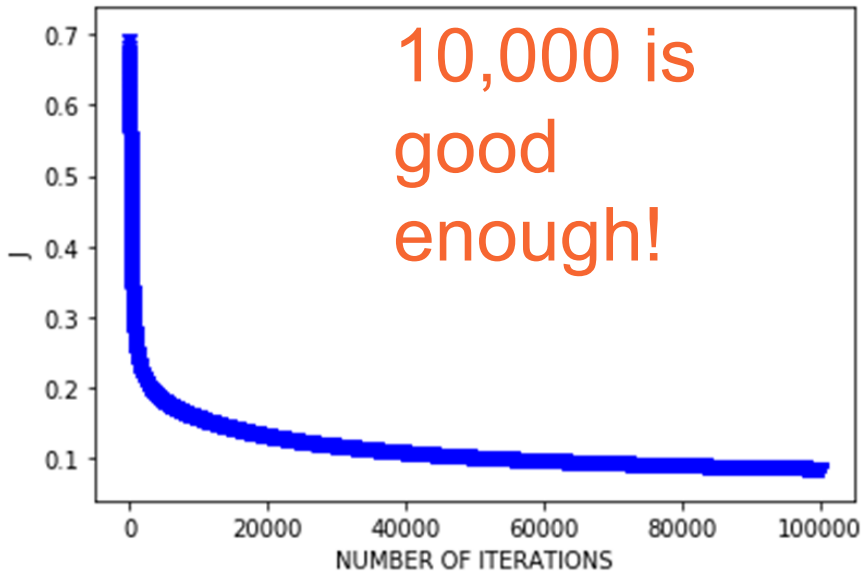
- Initialization
  - number of iterations = **10,000**, α = 0.1 and w_0= 0, w_1 = 0 , w_2 = 0.
- After 10,000 iterations we get: $w = \begin{bmatrix} -8.92966373 \\ -4.82507983 \\ 8.03599199 \end{bmatrix}, J = 0.1559449423214590 3$



Actual Class \ Predicted Class	N	Y
N	**True Negatives (TN=13)**	**False Positives (FP=1)**
Y	**False Negatives (FN=0)**	**True Positives (TP=6)**

Logistic regression with GD and two features

- Initialization
  - number of iterations = **100,000**, α = 0.1 and w_0= 0, w_1 = 0 , w_2 = 0.
- After 100,000 iterations we get: $w = \begin{bmatrix} 31.67979612 \\ -4.37788585 \\ 12.16406009 \end{bmatrix}, J = 0.0849833426353569 1$



Actual Class \ Predicted Class	N	Y
N	**True Negatives (TN=13)**	**False Positives (FP=1)**
Y	**False Negatives (FN=0)**	**True Positives (TP=6)**

Logistic regression with GD and two features

- Final result

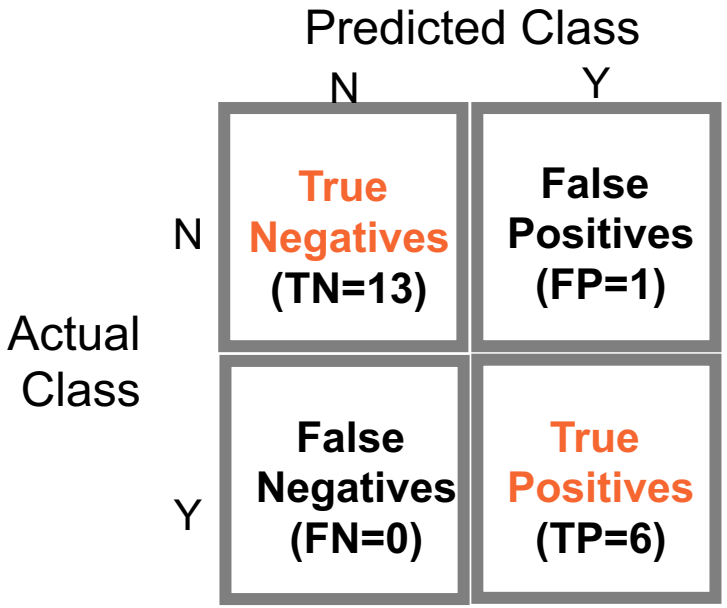
"versicolor" = 0, "virginica" = 1

$$\boldsymbol{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} = 95\%$$

$$\boldsymbol{Precision} = \frac{TP}{TP+FP} = 86\%$$

$$\boldsymbol{Recall} = \frac{TP}{TP+FN} = 100\%$$

$$\boldsymbol{F1} = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}} = 92\%$$

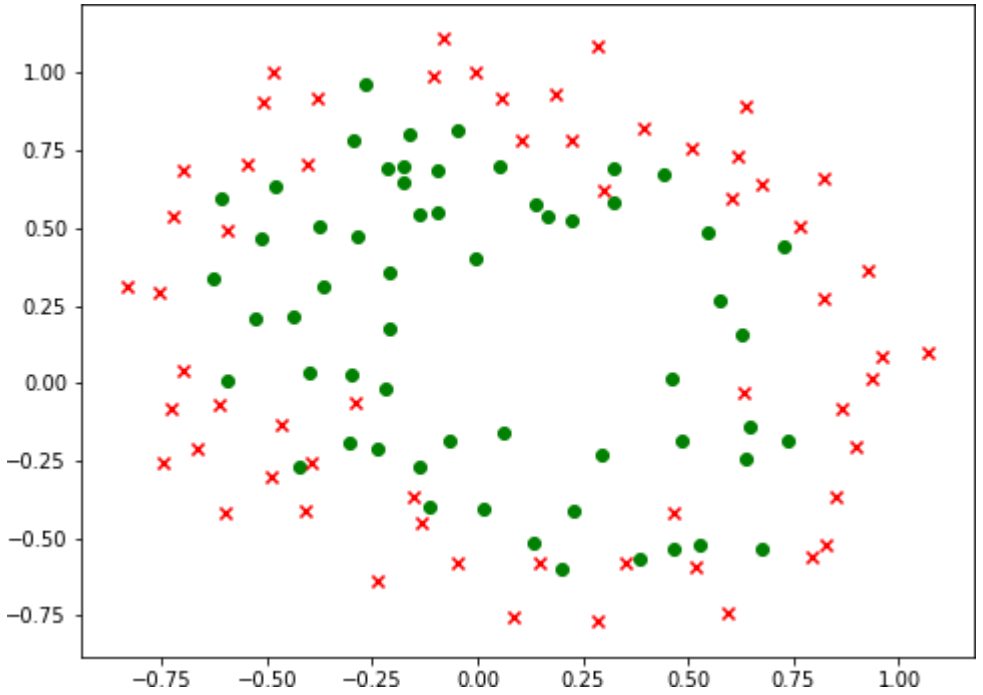


About the midterm project

- Programs submitted after the deadline have 20% points deducted from their score, if the submission is within 24 hours after the due date. Then, for every day after the first 24 hours, your penalized points will be double (40%, 80%, then 100%).
- No grace period is allowed
- Don’t use logistic regression library function, as you need to implement them from scratch of the logistic regression

About the midterm project

- Your assignment is to create a binary classifier to predict whether each capacitor in the test set will pass quality control



Obviously, a straight-line decision boundary is not going to work for this data. You will need to add new features to model a more complex boundary.

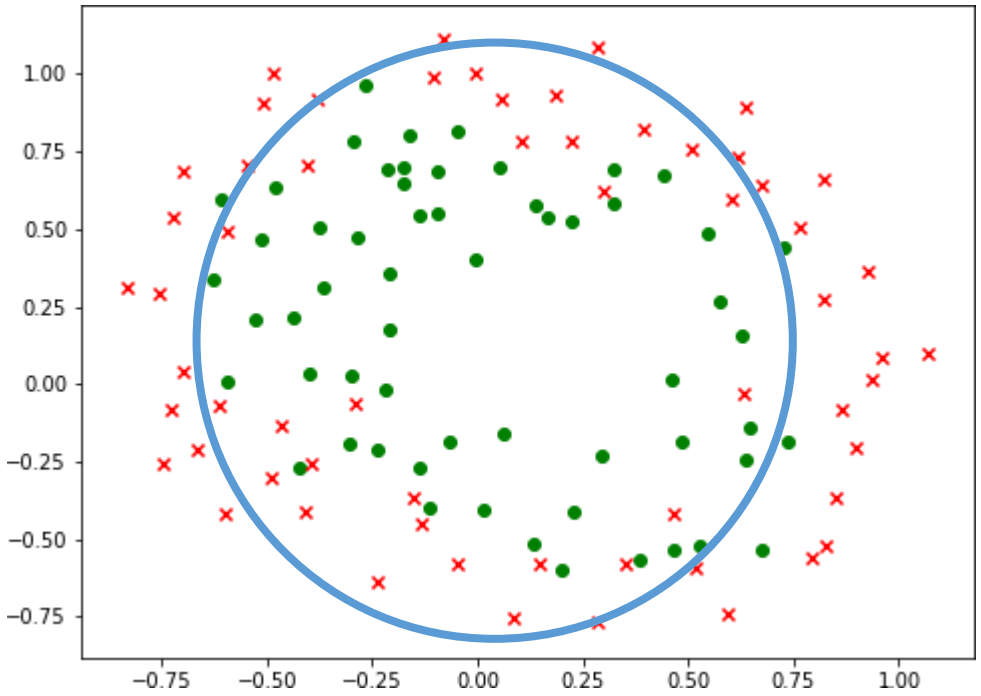
Currently the training data file looks like:

85 2

x_1 x_2 y

⋮

About the midterm project



If you thought that a circle might work, you might create a new training set file that looks like:

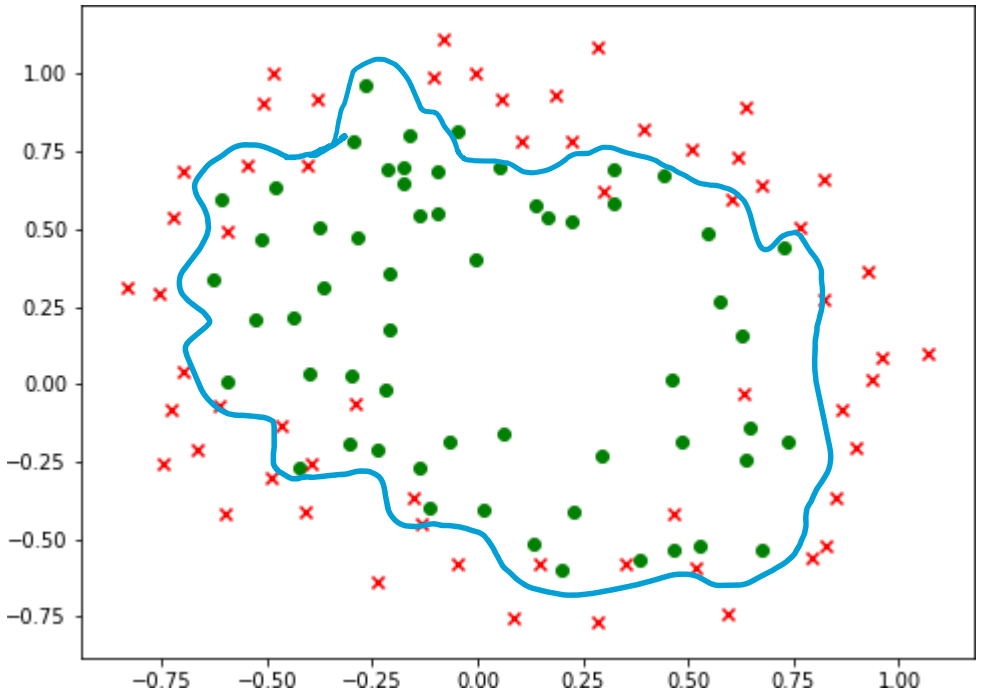
85 4

x_1 x_1^2 x_2 x_2^2 y

⋮

An important point is that your logistic regression program should be the same regardless of the type of model you choose

About the midterm project



If you thought that a circle might work, you might create a new training set file that looks like:

85 4

x_1 x_1^2 x_2 x_2^2 y

⋮

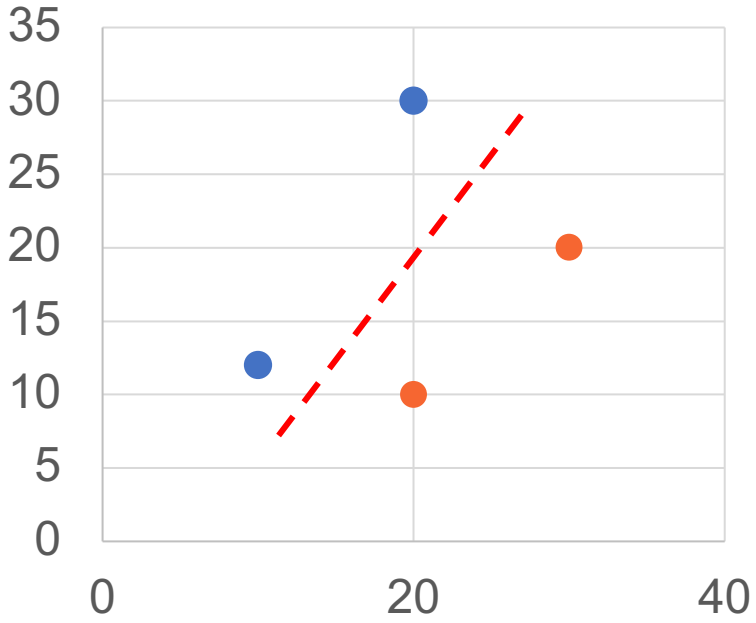
An important point is that your logistic regression program should be the same regardless of the type of model you choose

My advice

- First get your Logistic regression code working on a simple set of data
- Then, start applying it to the capacitor data set.
- For example:

Input data:

4	2	
10	12	1
20	10	0
30	20	0
20	30	1



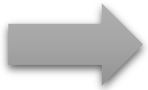
	Initially	1 Iteration	2 Iterations
w0	0	0	0.000568
w1	0	-0.025	-0.03631
w2	0	0.015	0.039159
J	-	0.693147	0.618463
Alpha	0.01	0.01	0.01
FP	-	0	0
TP	-	0	2
FN	-	2	0
TN	-	2	2

...

My advice

- Once you think your code works, then you can create new features for the capacitor data set to determine a good model. For example, you can use the following python program to automatically create models with different powers of the x_1 and x_2.

```
thePower = 2
for j  in  range(thePower+1):
   for i in range(thePower+1):
      temp = (x1**i)*(x2**j) if
      (temp != 1):
         fout1.write(str(temp)+"\t")
fout1.write(str(y)+"\n")
```



New Features	From Original Features
x_1	x^1
x_2	x_1^2
x_3	x_2
x_4	x_1x_2
x_5	$x_1x_2^2$
x_6	x_2^2
x_7	$x_1^2x_2$
x_8	$x_1^2x_2^2$

About the midterm project

- SVM
  - Can use sklearn package
  - Need to clarify your kernel choice
  - Plot the decision boundary