Do not distribute course material

You may not and may not allow others to reproduce or distribute lecture notes and course materials publicly whether or not a fee is charged.

Topic 4
Linear Classification & Logistic Regression

- http://cs229.stanford.edu/notes2020fall/notes2020fall/cs229-notes1.pdf
- https://eight2late.wordpress.com/2017/07/11/a-gentle-introduction-to-logistic-regression-and-lasso-regularisation-using-r/

PROF. LINDA SELLIE

Some slides/approaches used are from Prof. Rangan
Many approaches used are from CMU 18-661

Learning objectives

- Know how to use a hyperplane for binary classification
- Use the sigmoid function to scale a number in the range $[-\infty, \infty]$ into $[0,1]$
- Apply the principle of maximum likelihood estimation (MLE) to learn the parameters of a probabilistic model
- Derive the conditional log-likelihood
- How to apply gradient ascent to find the parameters of the the conditional log-likelihood
- Evaluate performance with different measures
- Create more complex models by feature transformation
- Understand how to add L1 and L2 regularization to the objective function
- Know how to interpret the output of soft-max

Logistic Regression

Data: $(\mathbf{x}^{(i)}, y^{(i)}), i = 1,2,\ldots,N$ where $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{0,1\}$

model: Logistic function applied to $\mathbf{w}^T\mathbf{x}$

$$p(y = 1 \mid \mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})}$$

Learning: find parameters that maximizes the **objective function:**

$$\mathbf{w}* = \arg\max_{\mathbf{w}} \left(\frac{1}{N} \sum_{i=1}^{N} y^{(i)} \ln(\sigma(\mathbf{w}^T\mathbf{x}^{(i)})) + (1 - y^{(i)}) \ln\left(1 - \sigma(\mathbf{w}^T\mathbf{x}^{(i)})\right) \right)$$

where $\sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})}$

Maximum Likelihood estimator (MLE) **w***

Next we will show how to find the optimal **w***

Prediction: either $\hat{y} = p(y \mid \mathbf{x}; \mathbf{w})$ or $\hat{y} = \arg\max_{y \in \{0,1\}} p(y \mid \mathbf{x}; \mathbf{w})$

$$\frac{\partial}{\partial w_j}\ell(\mathbf{w}) = \sum_{i=1}^{N}\left(y^{(i)} - \sigma(\mathbf{w}^T\mathbf{x}^{(i)})\right)x_j^{(i)}$$

Gradient Ascent Algorithm

$$\frac{1}{N}\ell(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\left[y^{(i)}\ln\sigma(\mathbf{w}^T\mathbf{x}^{(i)}) + (1-y^{(i)})\ln\left(1-\sigma(\mathbf{w}^T\mathbf{x}^{(i)})\right)\right]$$

for k = 1 to num_iter

$$temp0 = w_0 + \alpha\frac{\partial\ell(\mathbf{w})/N}{\partial w_0} = w_0 + \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - \sigma(\mathbf{w}^T\mathbf{x}^{(i)}))x_0^{(i)}$$

$$temp1 = w_1 + \alpha\frac{\partial\ell(\mathbf{w})/N}{\partial w_1} = w_1 + \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - \sigma(\mathbf{w}^T\mathbf{x}^{(i)}))x_1^{(i)}$$

$$\vdots$$

$$tempd = w_d + \alpha\frac{\partial\ell(\mathbf{w})/N}{\partial w_d} = w_d + \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - \sigma(\mathbf{w}^T\mathbf{x}^{(i)}))x_d^{(i)}$$

$x^{(i)}_0=1$

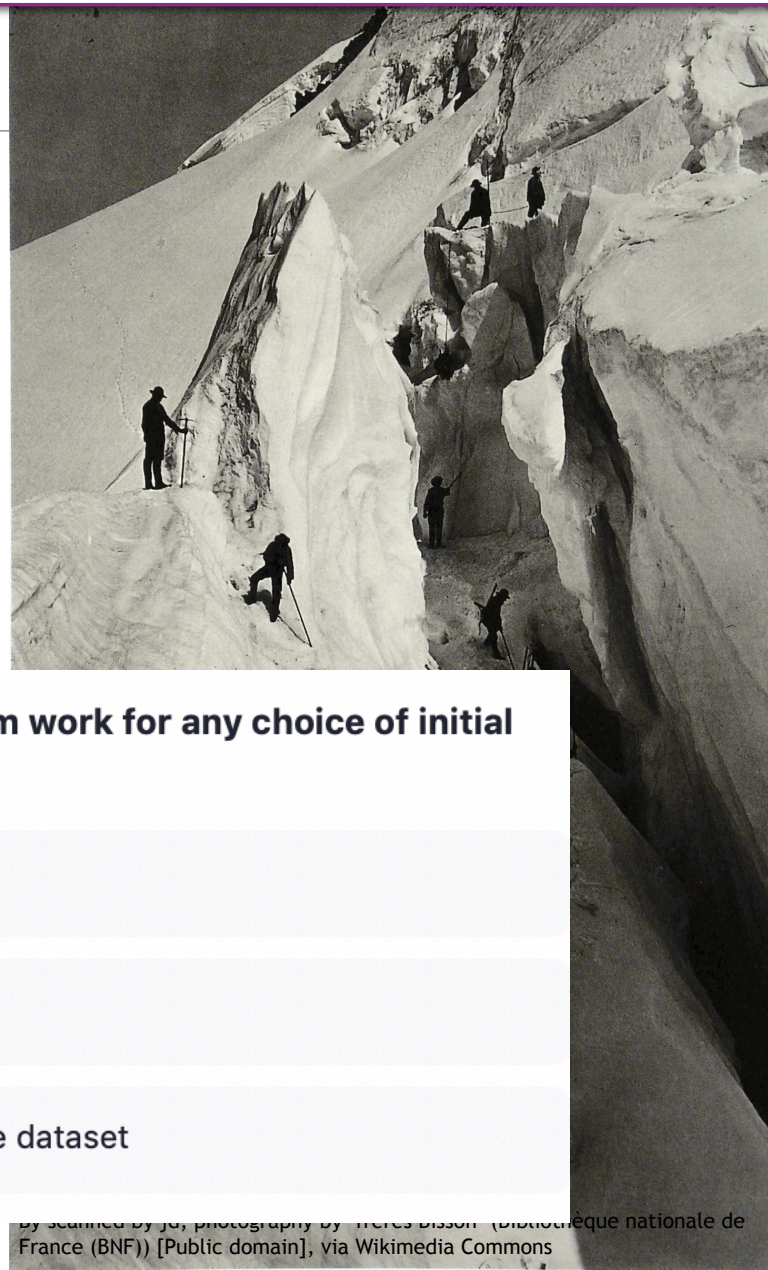
$$w_0 = temp0$$

$$w_1 = temp1$$

$$\vdots$$

$$w_d = tempd$$

Simultaneous update



1. Does this algorithm work for any choice of initial values for w?

- Yes
- No
- It depends on the dataset

Gradient Ascent Step

$$X = \begin{bmatrix} 1 & 3 & 4 \\ 1 & 3 & 5 \\ 1 & 4 & 1 \\ 1 & 3 & 1.5 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow X\mathbf{w} = \begin{bmatrix} 1 & 3 & 4 \\ 1 & 3 & 5 \\ 1 & 4 & 1 \\ 1 & 3 & 1.5 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \hat{\mathbf{y}} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

for k = 1 to num_iter

$$temp0 = w_0 + \frac{\alpha}{N}\frac{\partial \ell(\mathbf{w})}{\partial w_0} = w_0 + \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - \sigma(\mathbf{w}^T\mathbf{x}^{(i)}))x_0^{(i)} = w_0 + \frac{0.2}{4}\left[\left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(1)}\right)\right) + \left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(2)}\right)\right) + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(3)}\right)\right) + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(4)}\right)\right)\right]$$

-0.5 -0.5 0.5 0.5

$$temp1 = w_1 + \frac{\alpha}{N}\frac{\partial \ell(\mathbf{w})}{\partial w_1} = w_1 + \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - \sigma(\mathbf{w}^T\mathbf{x}^{(i)}))x_1^{(i)} = w_1 + \frac{0.2}{4}\left[\left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(1)}\right)\right)x_1^{(1)} + \left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(2)}\right)\right)x_1^{(2)} + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(3)}\right)x_1^{(3)}\right) + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(4)}\right)x_1^{(4)}\right)\right]$$

(-0.5)3 (-0.5)3 (0.5)4 (0.5)3

$$temp2 = w_2 + \frac{\alpha}{N}\frac{\partial \ell(\mathbf{w})}{\partial w_2} = w_2 + \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - \sigma(\mathbf{w}^T\mathbf{x}^{(i)}))x_2^{(i)} = w_2 + \frac{0.2}{4}\left[\left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(1)}\right)\right)x_2^{(1)} + \left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(2)}\right)\right)x_2^{(2)} + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(3)}\right)x_2^{(3)}\right) + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(4)}\right)x_2^{(4)}\right)\right]$$

(-0.5)4 (-0.5)5 (0.5)1 (0.5)(1.5)

$w_0^{new} = temp0$ 0

$w_1^{new} = temp1$ 0.025

$w_2^{new} = temp2$ -0.1625

Gradient Ascent Example

$$X = \begin{bmatrix} 1 & 3 & 4 \\ 1 & 3 & 5 \\ 1 & 4 & 1 \\ 1 & 3 & 1.5 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 0 \\ 0.025 \\ -0.1625 \end{bmatrix} \Rightarrow X\mathbf{w} = \begin{bmatrix} 1 & 3 & 4 \\ 1 & 3 & 5 \\ 1 & 4 & 1 \\ 1 & 3 & 1.5 \end{bmatrix} \begin{bmatrix} 0 \\ 0.025 \\ -0.1625 \end{bmatrix} = \begin{bmatrix} -0.575 \\ -0.7375 \\ -0.0625 \\ -0.16875 \end{bmatrix} \Rightarrow \hat{\mathbf{y}} = \begin{bmatrix} 0.3601 \\ 0.3236 \\ 0.4844 \\ 0.4579 \end{bmatrix}$$

for k = 1 to num_iter

$$temp0 = w_0 + \frac{\alpha}{N}\frac{\partial \ell(\mathbf{w})}{\partial w_0} = w_0 + \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - \sigma(\mathbf{w}^T\mathbf{x}^{(i)}))x_0^{(i)} = w_0 + \frac{0.2}{4}\left[\left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(1)}\right)\right) + \left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(2)}\right)\right) + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(3)}\right)\right) + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(4)}\right)\right)\right]$$

(0-0.3601) (0-0.3236) (1-0.4844) (1-0.4579)

$$temp1 = w_1 + \frac{\alpha}{N}\frac{\partial \ell(\mathbf{w})}{\partial w_1} = w_1 + \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - \sigma(\mathbf{w}^T\mathbf{x}^{(i)}))x_1^{(i)} w_1 + \frac{0.2}{4}\left[\left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(1)}\right)\right)x_1^{(1)} + \left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(2)}\right)\right)x_1^{(2)} + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(3)}\right)x_1^{(3)}\right) + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(4)}\right)x_1^{(4)}\right)\right]$$

(0-0.3601)*3 (0-0.3236)*3 (1-0.4844)*4 (1-0.4579)*3

$$temp2 = w_2 + \frac{\alpha}{N}\frac{\partial \ell(\mathbf{w})}{\partial w_2} = w_2 + \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - \sigma(\mathbf{w}^T\mathbf{x}^{(i)}))x_2^{(i)} = w_2 + \frac{0.2}{4}\left[\left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(1)}\right)\right)x_2^{(1)} + \left(0 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(2)}\right)\right)x_2^{(2)} + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(3)}\right)x_2^{(3)}\right) + \left(1 - \sigma\left(\mathbf{w}^T\mathbf{x}^{(4)}\right)x_2^{(4)}\right)\right]$$

(0-0.3601)*4 (0-0.3236)*5 (1-0.4844)*1 (1-0.4579)*(1.5)

$w_0^{new} = temp0$ 0.019

$w_1^{new} = temp1$ 0.107

$w_2^{new} = temp2$ -0.249

Vectorized Prediction

For this slide, we want $\hat{y} = p(y \mid \mathbf{x}; \mathbf{w})$

For $X = \begin{bmatrix} 1 & 3 & 4 \\ 1 & 3 & 5 \\ 1 & 4 & 1 \\ 1 & 3 & 1.5 \end{bmatrix}$ $\mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$ and initial weights $\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$,

we predict $\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \hat{y}^{(3)} \\ \hat{y}^{(4)} \end{bmatrix} = \begin{bmatrix} \sigma(\mathbf{w}^T \mathbf{x}^{(1)}) \\ \sigma(\mathbf{w}^T \mathbf{x}^{(2)}) \\ \sigma(\mathbf{w}^T \mathbf{x}^{(3)}) \\ \sigma(\mathbf{w}^T \mathbf{x}^{(4)}) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} = \text{sigmoid}(X\mathbf{w})$

Vectorized Gradient

$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)} = \sum_{i=1}^{N} (y^{(i)} - \sigma(\mathbf{w}^T \mathbf{x}^{(i)})) x_j^{(i)}$$

$$\nabla \ell(\mathbf{w}) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \cdots & \vdots \\ x_d^{(1)} & x_d^{(2)} & \cdots & x_d^{(N)} \end{bmatrix} \begin{bmatrix} y^{(1)} - \hat{y}^{(1)} \\ y^{(2)} - \hat{y}^{(2)} \\ \vdots \\ y^{(N)} - \hat{y}^{(N)} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)}) x_0^{(i)} \\ \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)}) x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)}) x_d^{(i)} \end{bmatrix} = \begin{bmatrix} \frac{\partial \ell(\mathbf{w})}{\partial w_0} \\ \frac{\partial \ell(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial \ell(\mathbf{w})}{\partial w_d} \end{bmatrix} = X^T (y - \text{sigmoid}(X\mathbf{w}))$$

Example: $X = \begin{bmatrix} 1 & 3 & 4 \\ 1 & 3 & 5 \\ 1 & 4 & 1 \\ 1 & 3 & 1.5 \end{bmatrix}$ $\mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

$$\nabla \ell(\mathbf{w}) = \begin{bmatrix} \frac{\partial \ell(\mathbf{w})}{\partial w_0} \\ \frac{\partial \ell(\mathbf{w})}{\partial w_1} \\ \frac{\partial \ell(\mathbf{w})}{\partial w_2} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 3 & 3 & 4 & 3 \\ 4 & 5 & 1 & 1.5 \end{bmatrix}}_{X^T} \underbrace{\begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \hat{y}^{(4)} \\ \hat{y}^{(4)} \end{bmatrix} \end{bmatrix}}_{(\mathbf{y} - \hat{\mathbf{y}})} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 3 & 3 & 4 & 3 \\ 4 & 5 & 1 & 1.5 \end{bmatrix}}_{X^T} \underbrace{\begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} - \text{sigmoid} \left(\begin{bmatrix} 1 & 3 & 4 \\ 1 & 3 & 5 \\ 1 & 4 & 1 \\ 1 & 3 & 1.5 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \right) \end{bmatrix}}_{(\mathbf{y} - \text{sigmoid}(X\mathbf{w}))}$$

Vectorized Gradient Ascent Algorithm

for k = 1 to num_iter

$$\mathbf{w} = \mathbf{w} + \frac{\alpha}{N}\left(X^T(\mathbf{y} - \text{sigmoid}(X\mathbf{w}))\right)$$

Example where $\alpha = 0.2$

for k = 1 to num_iter

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} + \frac{0.2}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 3 & 3 & 4 & 3 \\ 4 & 5 & 1 & 1.5 \end{bmatrix} \left[\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} - \text{sigmoid}\left(\begin{bmatrix} 1 & 3 & 4 \\ 1 & 3 & 5 \\ 1 & 4 & 1 \\ 1 & 3 & 1.5 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \right) \right]$$

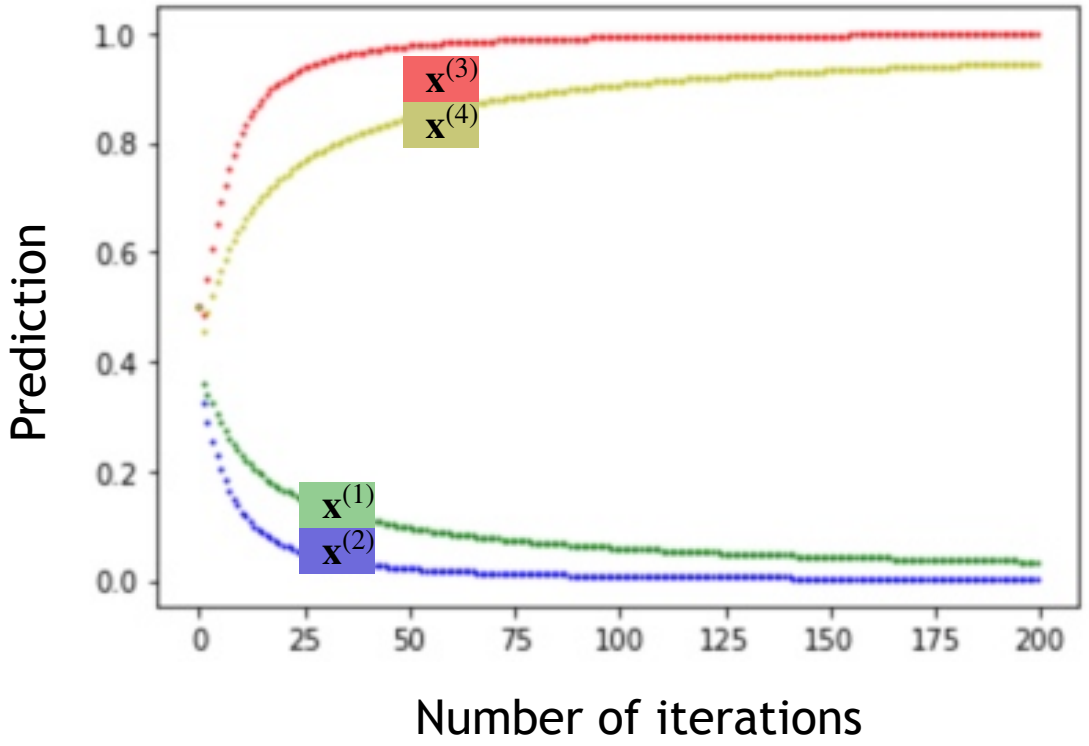
Gradient Ascent

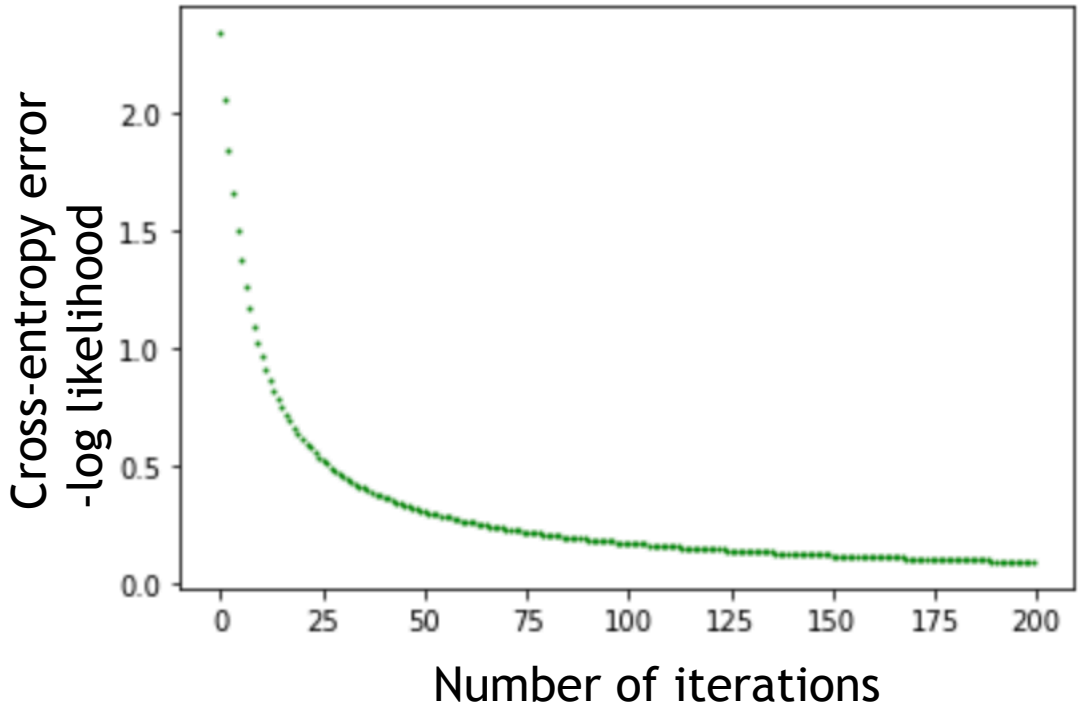
$\mathbf{x}^{(1)} = [1 \;\; 3 \;\; 4]^T$
$\mathbf{x}^{(2)} = [1 \;\; 3 \;\; 5]^T$

Label 0 examples

$\mathbf{x}^{(3)} = [1 \;\; 4 \;\; 1]^T$
$\mathbf{x}^{(4)} = [1 \;\; 3 \;\; 1.5]^T$

Label 1 examples





Prediction

After 200 iterations of gradient with a learning rate of 0.2, $\mathbf{w} = \begin{bmatrix} 0.551 \\ 2.011 \\ -2.48 \end{bmatrix}$

Given a feature vector for a new flower $\mathbf{x} = \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix}$ we predict the probability of being a Setosa Iris is $\sigma(-0.856)$, i.e. the flower has a 0.3 chance of being a Setosa Iris.

If we had to predict which flower we would predict it is a Versicolor Iris.

Outline

- Motivating example: How can we classify? — How can we use a hyperplane for a classification problem?
- Estimating probabilities — Can we predict not only which class an example belongs to - but a confidence score of that classification
- Maximum likelihood — How can we find the most likely hyperplane? Could we write a function to describe how likely a hyperplane was to have generated the dataset?
  - Iterative approach - gradient ascent — Maximizing the function
- Thinking about different types of error — Some errors are more costly than other errors. Can we modify our predictions to decrease one type of error (and perhaps increase another type of error?
  - Accuracy
  - Motivating example
- Transformation of the features — Extending our algorithm to nonlinear decision boundaries
- Multiple classes — What if we have more than two classes?

Most Data not Perfectly Separable

- Generally cannot find a separating hyperplane
- Always, some points that will be mis-classified
- Algorithms attempt to find "good" hyper-planes
  - Reduce the number of mis-classified points
  - Or, some similar metric

What type of measurements do you think we should calculate?

Evaluating Performance

- Classification error: # mistakes made on the training set $\frac{\#\text{mistakes}}{\#\text{examples}}$
- Accuracy: fraction of correct examples $\frac{\#\text{correct}}{\#\text{examples}} = 1 - \frac{\#\text{mistakes}}{\#\text{examples}}$
  - Problem: Unbalanced datasets. Eg. Suppose your data consist examples, and 1% negative examples.

1. Given an unbalanced dataset containing 99% positive examples (class 1) and 1% negative examples (class 0). Do you think it is possible to find a classifier that has 99% accuracy?

- No
- Yes
- Maybe - depends on the algorithm

Evaluating Performance

- Classification error: # mistakes made on the training set $\frac{\#\text{mistakes}}{\#\text{examples}}$
- Accuracy: fraction of correct examples $\frac{\#\text{correct}}{\#\text{examples}} = 1 - \frac{\#\text{mistakes}}{\#\text{examples}}$
  - Problem: Unbalanced datasets. Eg. Suppose your data consist of 99% positive examples, and 1% negative examples.

    If you predict positive (without looking at **x**), your accuracy is 99%!

What other error measurements make sense?

Hard Decisions

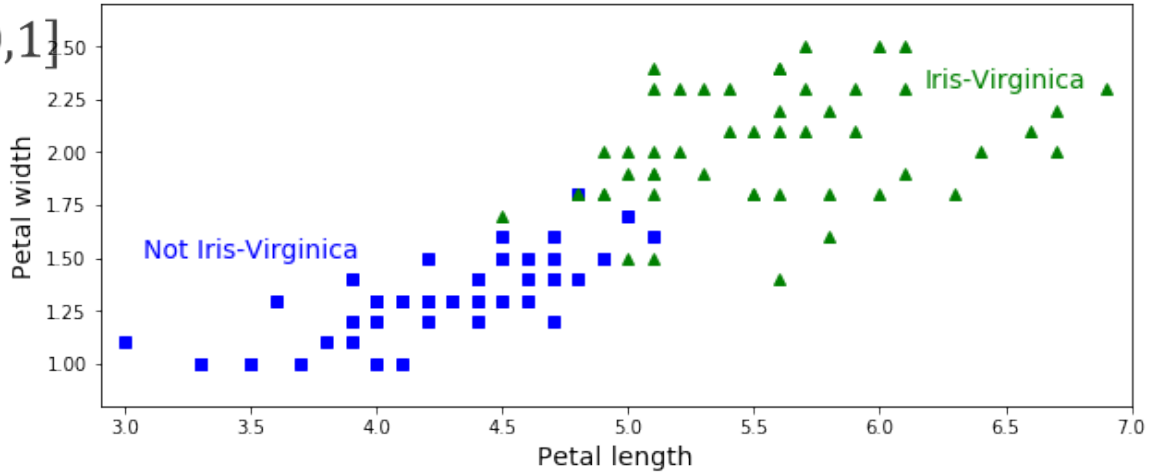

Iris virginica

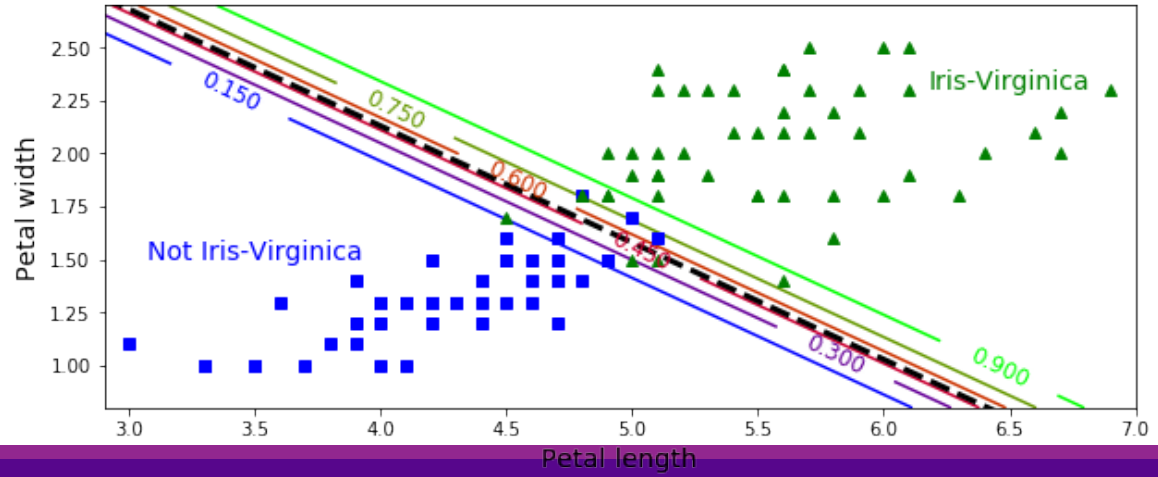

Iris versicolor


Iris setosa

iris pictures from https://en.wikipedia.org/wiki/Iris_flower_data_set

- Logistic classifier outputs a soft label: $P(y = 1|x) \in [0,1]$
  - $P(y = 1|x) \approx 1 \Rightarrow y = 1$ more likely
  - $P(y = 0|x) \approx 1 \Rightarrow y = 0$ more likely
- Can obtain a hard label by thresholding:
  - Set $\hat{y} = 1 \Leftrightarrow \ P(y = 1|x) > t$
  - t = Threshold
- How to set threshold?
  - Set $t = \frac{1}{2} \Rightarrow$ Minimizes overall error rate
  - Increasing $t \Rightarrow$ Decreases false positives
  - Decreasing $t \Rightarrow$ Decreases missed detections



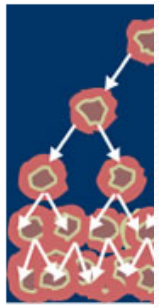


New example

Breast Cancer Wisconsin (Diagnostic) Data Set

Download: Data Folder, Data Set Description

Abstract: Diagnostic Wisconsin Breast Cancer Database



Data Set Characteristics:	Multivariate	**Number of Instances:**	569	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	32	**Date Donated**	1995-11-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	442524

Attribute Information:

1) ID number
2) Diagnosis (M = malignant, B = benign)
3-32)

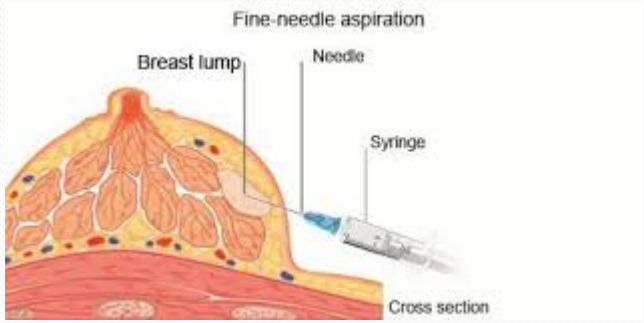
Ten real-valued features are computed for each cell nucleus:

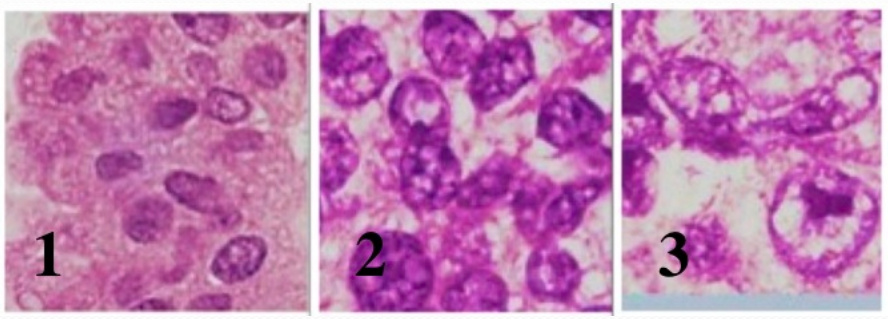
a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)

- First publication: O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995

Diagnosing Breast Cancer

- Fine needle aspiration of suspicious lumps
- Cytopathologist visually inspects cells
  - Sample is stained and viewed under microscope
- Determines if cells are benign or malignant and furthermore provides grading if malignant
- Uses many features:
  - Size and shape of cells, degree of mitosis, differentiation, …
- Diagnosis is not exact
- If uncertain, use a more comprehensive biopsy
  - Additional cost and time
  - Stress to patient
- Can machine learning provide better rules?





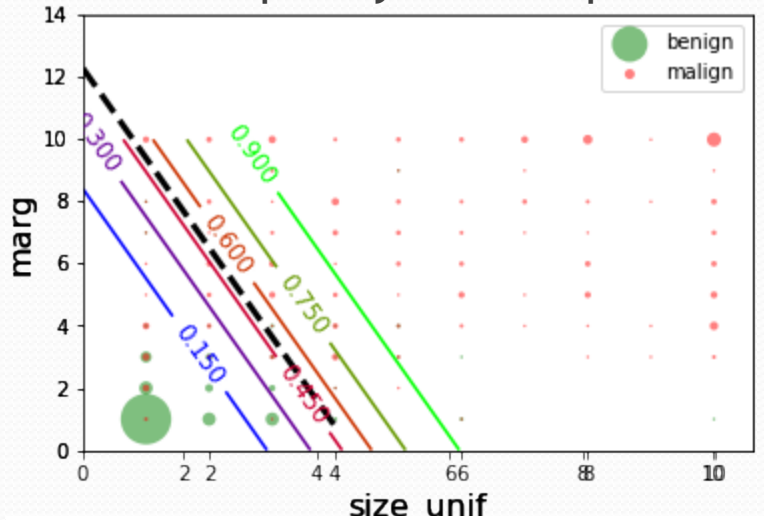
Grades of carcinoma cells
http://breast-cancer.ca/5a-types/

Example - predicting breast cancer

We have 683 examples

	id	thick	size_unif	shape_unif	marg	cell_size	bare	chrom	normal	mit	class
0	1000025	5	1	1	1	2	1.0	3	1	1	2
1	1002945	5	4	4	5	7	10.0	3	2	1	2
2	1015425	3	1	1	1	2	2.0	3	1	1	2
3	1016277	6	8	8	1	3	4.0	3	7	1	2
4	1017023	4	1	1	3	2	1.0	3	1	1	2
5	1017122	8	10	10	8	7	10.0	9	7	1	4

- Scatter plot of points from each class
- Plot not informative
  - Many points overlap
  - Relative frequency at each point

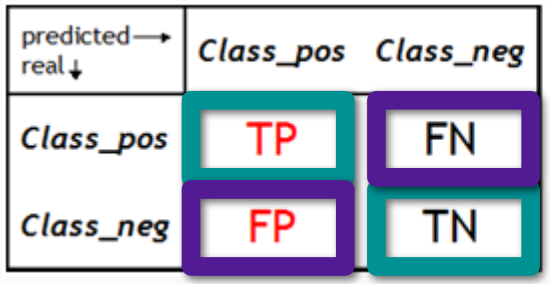


`w=[-5.50017433,1.29037381 0.44738685]`

Of course we can have a more accurate prediction if we use more of the features

Errors in Binary Classification

- Doctors want to know if a patient has breast cancer
- This is a binary classification problem
- There are two types of **errors** an example can have:
  - Type 1 error. The doctor predicts the patient has cancer but the patient doesn't. **False positive (FP)**
  - Type 2 error. The doctor predicts the patient doesn't have cancer, but the patient does. **False negative (FN)**
- Their are two types of **correctly** classified examples:
  - The doctor *correctly* predicts the patient has cancer. **True positive (TP)**
  - The doctor *correctly* predicts the patient doesn't have cancer. **True negative (TN)**
- Confusion matrix
  - The diagonal contains TP and TN
  - The off diagonal contains the FP and FN
- Uses of a confusion matrix:
  - Shows if the classes are imbalanced
  - Shows the different types of errors



$$\text{TPR (sensitivity)} = \frac{TP}{TP + FN}$$

$$\text{FPR (1-specificity)} = \frac{FP}{TN + FP}$$

Errors in Binary Classification

- Doctors want to know if a patient has breast cancer
- This is a binary classification problem
- There are two types of **errors** an example can have:
  - Type 1 error. The doctor predicts the patient has cancer but the patient doesn't. **False positive (FP)**
  - Type 2 error. The doctor predicts the patient doesn't have cancer, but the patient does. **False negative (FN)**
- Their are two types of **correctly** classified examples:
  - The doctor *correctly* predicts the patient has cancer. **True positive (TP)**
  - The doctor *correctly* predicts the patient doesn't have cancer. **True negative (TN)**
- Confusion matrix
  - The diagonal contains TP and TN
  - The off diagonal contains the FP and FN
- Uses of a confusion matrix:
  - Shows if the classes are imbalanced
  - Shows the different types of errors

There is often a trade off between the TPR and the FPR.

predicted → / real ↓	*Class_pos*	*Class_neg*
Class_pos	TP	FN
Class_neg	FP	

We can plot the FPR and TPR for different cutoff values - e.g. 0.2, 0.3, 0.4, 0.5, …, 0.9

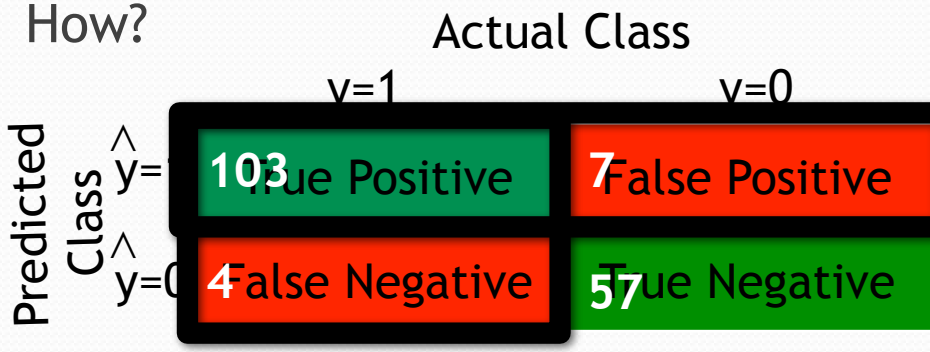
By choosing a value different than 0.5 as our cutoff we can change the number of FP and FN

$$\text{y}) = \frac{\text{FP}}{\text{TN}+\text{FP}}$$

Evaluating Errors: Precision and Recall

Notice we changed the labels on the conclusion matrix Always check the labels

- How can we evaluate the performance of our classifier?
- For binary classification, we can easily get half the answers right! How?
- What the class is 90% positive and 10% negative.
  - Can you get a 90% accurate answer?
- Precision and Recall
  - Precision- fraction of predicted positive answers that were really positive
  - Recall - fraction of positive answers you correctly calculated (Aka *TPR*, *sensitivity*)



Best is 1 and worst is 0

$$\frac{\#\text{ True Positive}}{\#\text{ True Positive} + \#\text{ False Positive}} \qquad \frac{103}{103+7}$$

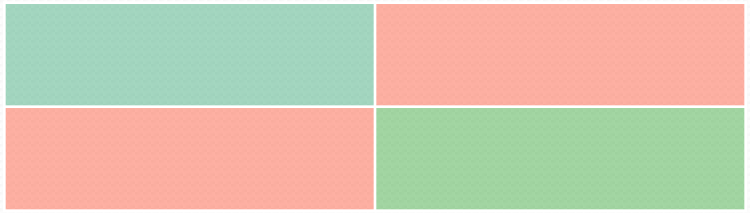
$$\frac{\#\text{ True Positive}}{\#\text{ True Positive} + \#\text{False Negative}} \qquad \frac{103}{103+4}$$

To learn more watch https://www.youtube.com/watch?v=eU4_3dzU7yw&t=216

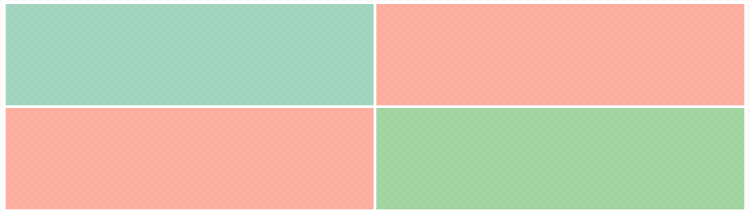
Toy example

True Positive	False Positive
False Negative	True Negative

- Balancing precision and recall:
- if $t \geq 0.5$



- if $t \geq 0.55$



Y	h(x)	>=0.5	>0.55
0	0.44	False	False
0	0.74	True	True
0	0.51	True	False
1	0.54	True	False
1	0.85	True	True
0	0.31	False	False
0	0.38	False	False
1	0.56	True	True
1	0.48	False	False
0	0.37	False	False

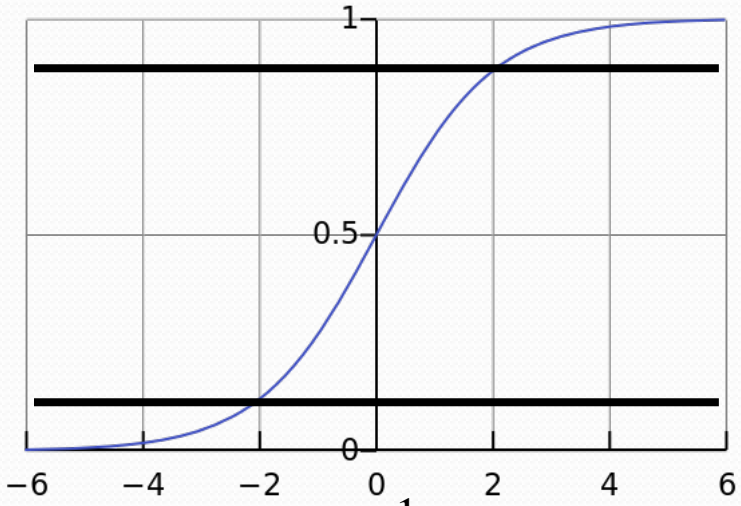
Trade off between Precision and Recall

- Optimistic classifier - almost all examples are labeled 1
- Negative classifier - almost all examples are labeled 0
- Balancing precision and recall:

recall =1, but precision is low

very few examples will be classified as 1
precision will be high, recall will be low

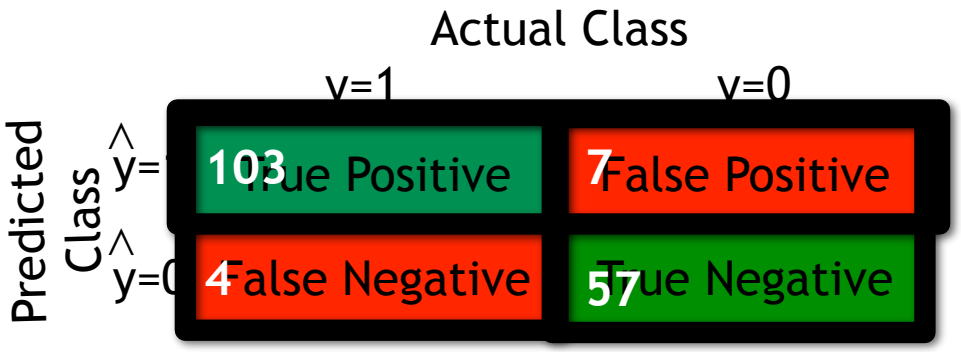
very few examples will be classified as 0
precision will be low, recall will be high



$$h(\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x})}}$$

Overview

$$\text{Accuracy} = \frac{\#\,\text{correct}}{\#\,\text{examples}} = \frac{103 + 57}{103 + 57 + 4 + 7}$$



Precision $\dfrac{\#\,\text{True Positive}}{\#\,\text{True Positive} + \#\,\text{False Positive}}$ $\dfrac{103}{103+7}$

Recall, *TPR*, *sensitivity*

$$\frac{\#\,\text{True Positive}}{\#\,\text{True Positive} + \#\text{False Negative}}$$

$$\frac{103}{103+4}$$

Specificity

$$\frac{\#\text{true negative}}{\#\text{true negative} + \#\text{false positive}}$$

$$\frac{57}{57 \quad +7}$$

Pair share:

If you raise the threshold, how does recall change?

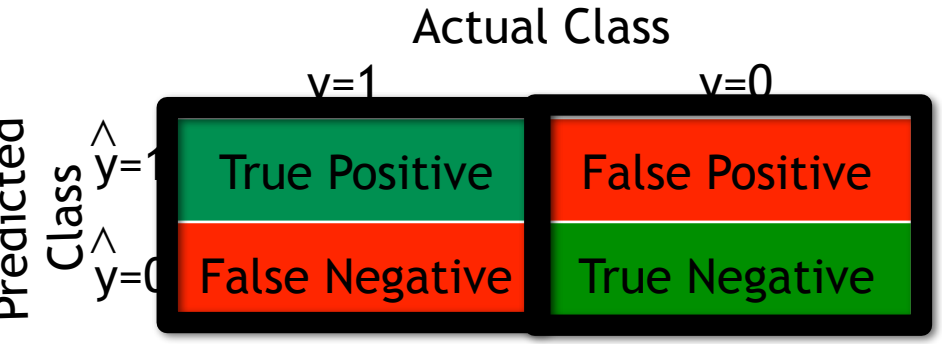
If you raise the threshold, how does precision change?

https://www.youtube.com/watch?v=21Igj5Pr6u4
https://www.youtube.com/watch?v=4jRBRDbJemM

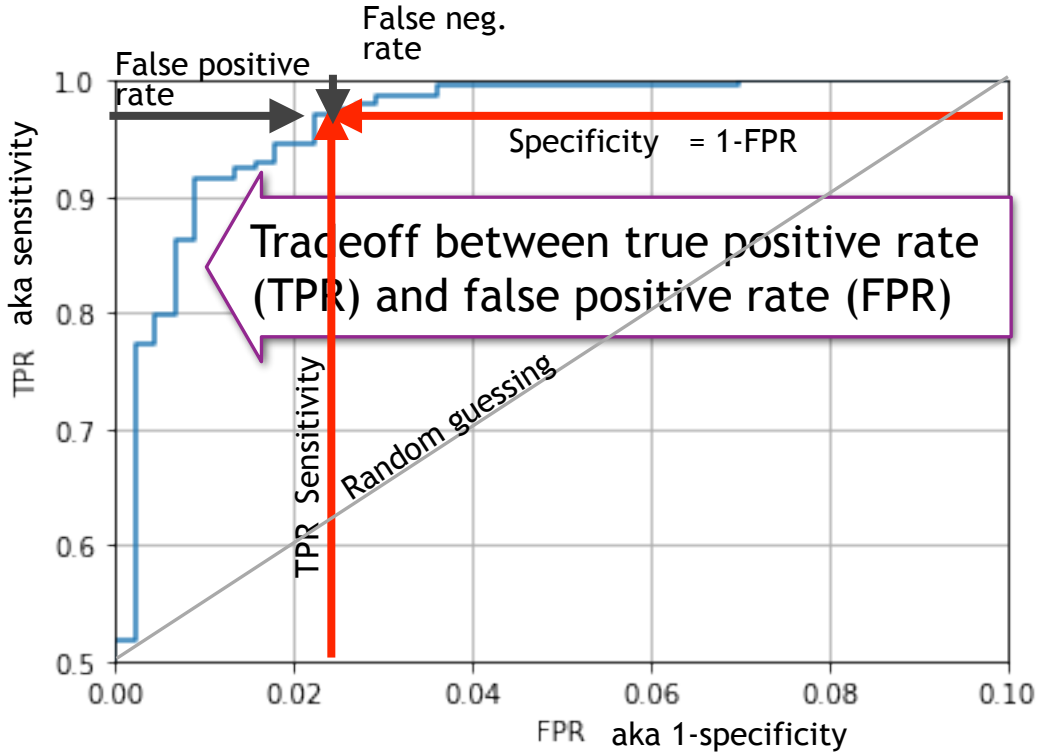
Receiver Operating Characteristic (ROC) curve

How many false positives we tolerate to get a certain number of true positives

- Varying threshold obtains a set of classifiers
- Trades off FPR and TPR for different hyper parameter settings (in this case the threshold t)
- Can visualize with ROC curve
  - Receiver operating curve
  - Term from digital communications



$$TPR = \frac{\#TP}{\#FN + \#TP} \qquad FPR = \frac{\#FP}{\#FP + \#TN}$$



https://www.youtube.com/watch?v=21Igj5Pr6u4
https://www.youtube.com/watch?v=4jRBRDbJemM

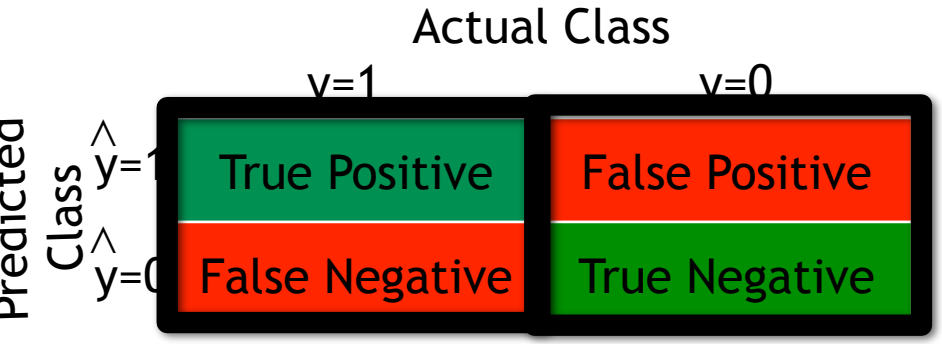
Receiver Operating Characteristic (ROC

If the data set is unbalanced, Precision recall curve is better than using accuracy

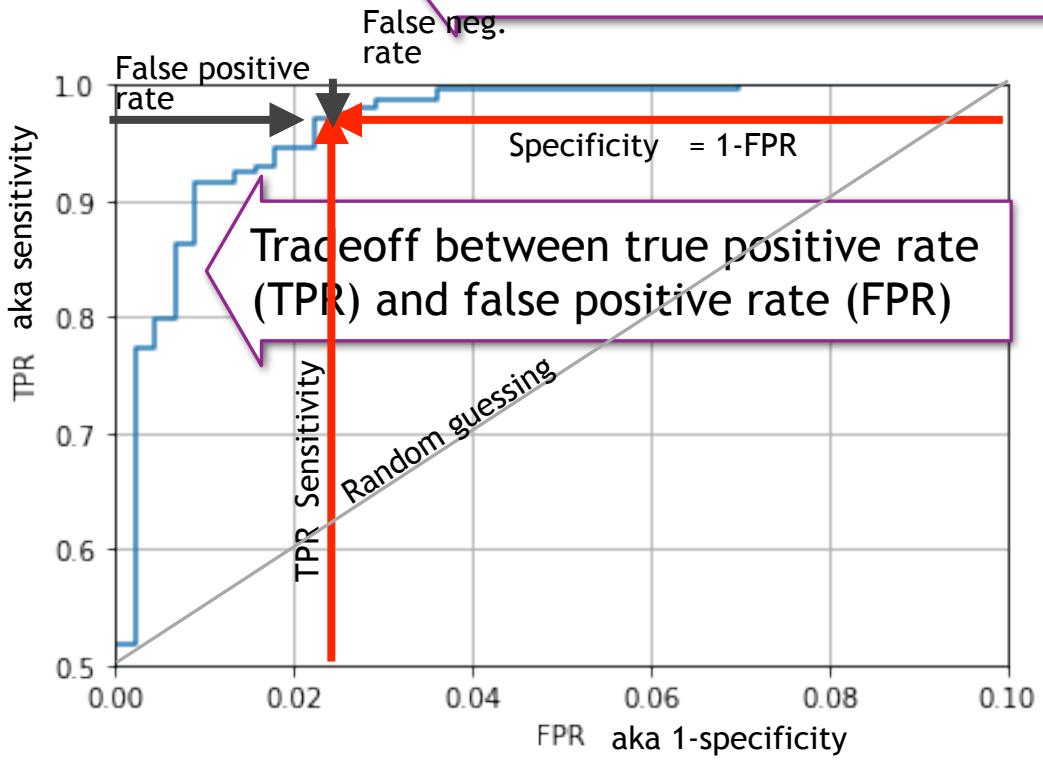
How many false positives we tolerate to get a certain number of true positives

- Varying threshold obtains a set of classifiers
- Trades off FPR and TPR for different hyper parameter settings (in this case the threshold t)
- Can visualize with ROC curve
  - Receiver operating curve
  - Term from digital communications

Useful for unbalanced classes.



$$TPR = \frac{\#TP}{\#FN + \#TP} \qquad FPR = \frac{\#FP}{\#FP + \#TN}$$

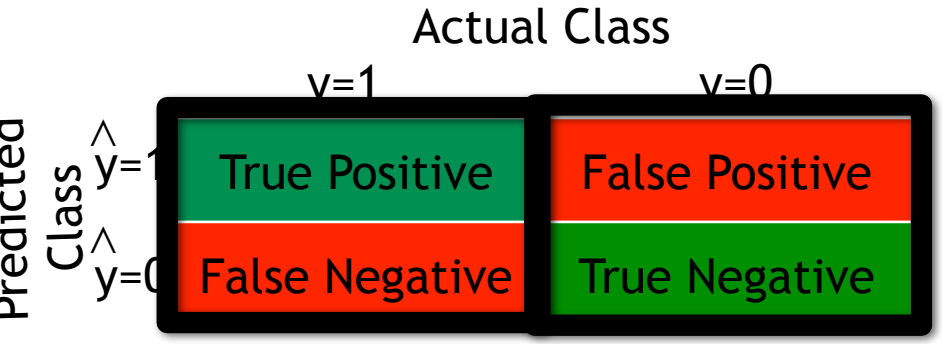


https://www.youtube.com/watch?v=21Igj5Pr6u4
https://www.youtube.com/watch?v=4jRBRDbJemM

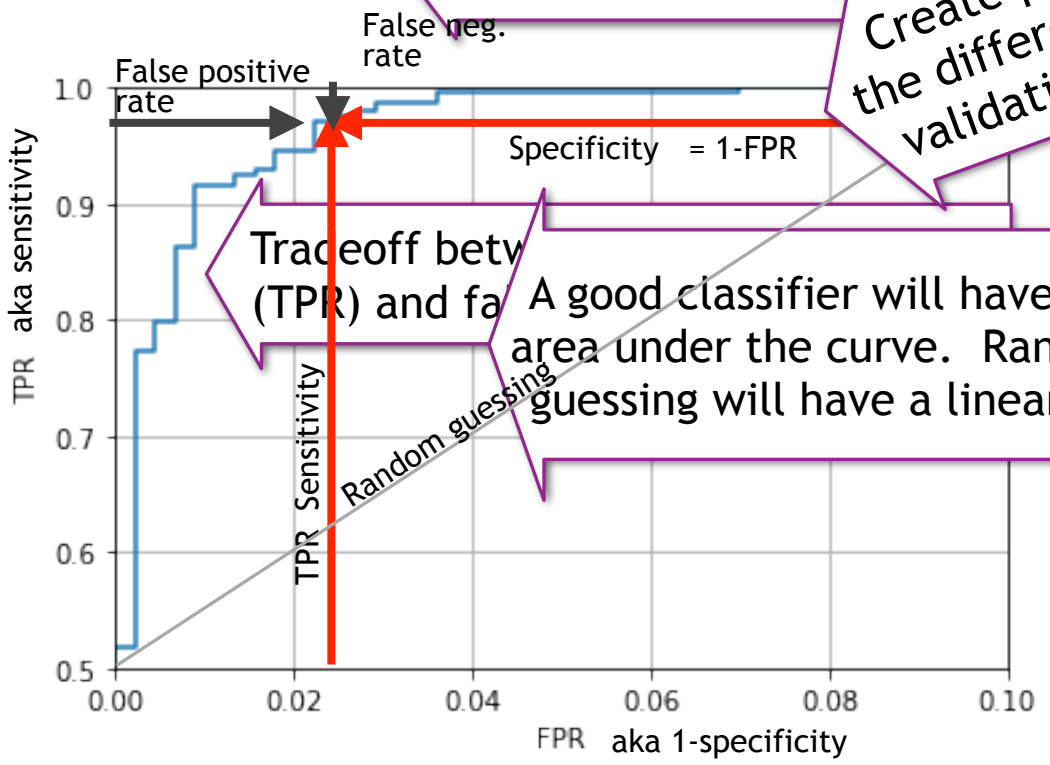
Receiver Operating Characteristic (ROC

How many false positives we tolerate to get a certain number of true positives

- Varying threshold obtains a set of classifiers
- Trades off FPR and TPR for different hyper parameter settings (in this case the threshold t)
- Can visualize with ROC curve
  - Receiver operating curve
  - Term from digital communications



$$TPR = \frac{\#TP}{\#FN + \#TP} \qquad FPR = \frac{\#FP}{\#FP + \#TN}$$



If the data set is unbalanced, Precision recall curve is better than using accuracy

Useful for unbalanced cl

Create plot by evaluating the different thresholds using validation set

Tradeoff betw (TPR) and fa

A good classifier will have a large area under the curve. Random guessing will have a linear ROC

F1 Score

https://en.wikipedia.org/wiki/F1_score

$$\text{Precision} = \frac{\#\,\text{True Positive}}{\#\,\text{True Positive} + \#\,\text{False Positive}}$$

$$\text{Recall} = \frac{\#\,\text{True Positive}}{\#\,\text{True Positive} + \#\text{False Negative}}$$

The F1 score balances precision and recall

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

The advantage of using F1 is we have a single number to evaluate the performance of an algorithm. The F1 score is between 0 and 1

Unlike accuracy, which can be misleading if the classes are unbalanced. F1 score gives one number indicating a score that is a balance between precision and recall

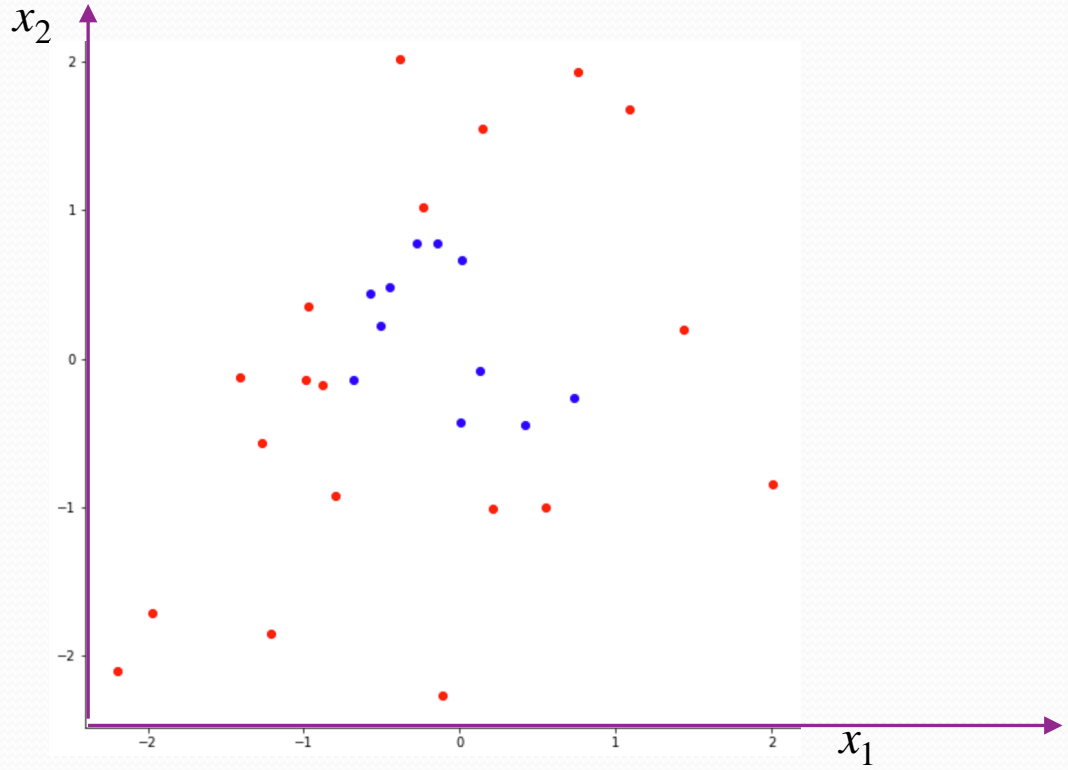
This may not be what you want if you care more about precision than recall, or if you care more about recall than precision

Outline

- Motivating example: How can we classify? — How can we use a hyperplane for a classification problem?
- Estimating probabilities — Can we predict not only which class an example belongs to - but a confidence score of that classification
- Maximum likelihood — How can we find the most likely hyperplane? Could we write a function to describe how likely a hyperplane was to have generated the dataset?
  - Iterative approach - gradient ascent — Maximizing the function
- Thinking about different types of error — Some errors are more costly than other errors. Can we modify our predictions to decrease one type of error (and perhaps increase another type of error?
  - Accuracy
  - Motivating example
- Transformation of the features — Extending our algorithm to nonlinear decision boundaries
- Multiple classes — What if we have more than two classes?

Non-linear Decision Boundary

Could our logistic regression function do well on the dataset?



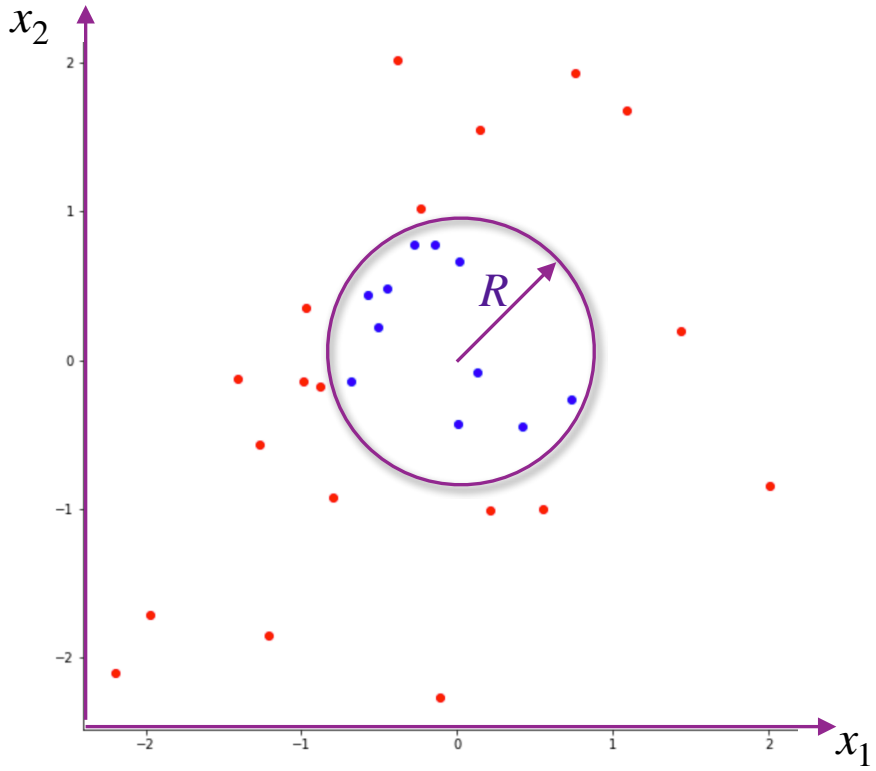
What do we do if our data had a more complex decision boundary?

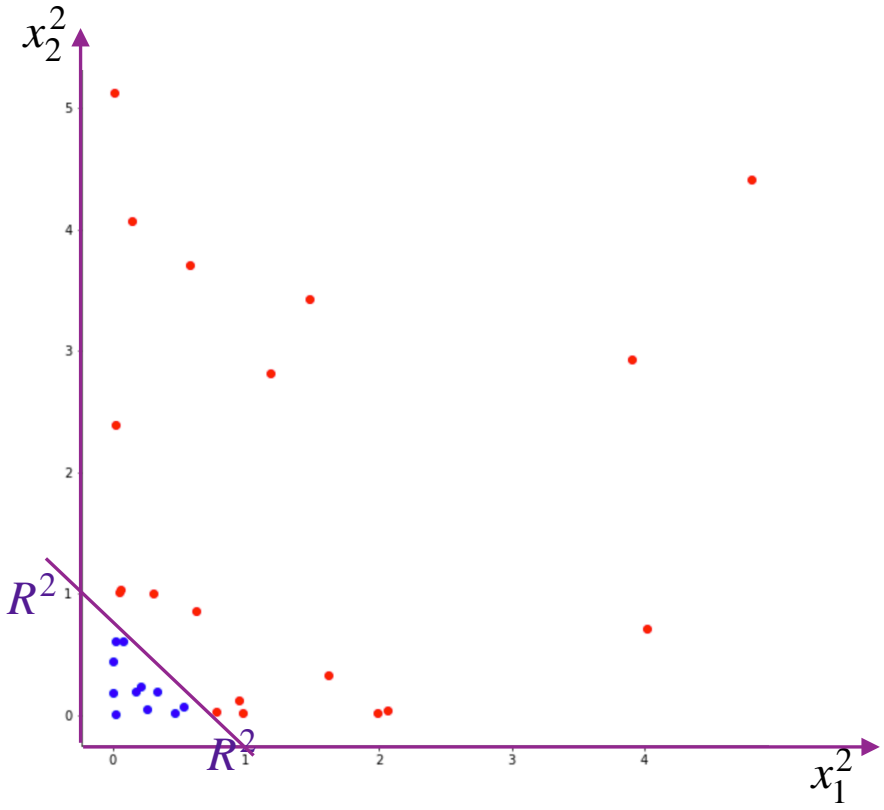
Transforming the Feature Space

Z Space

Just as we did in linear regression, we can expand the model logistic model by transforming the features:

If we add features, can we separate the data?

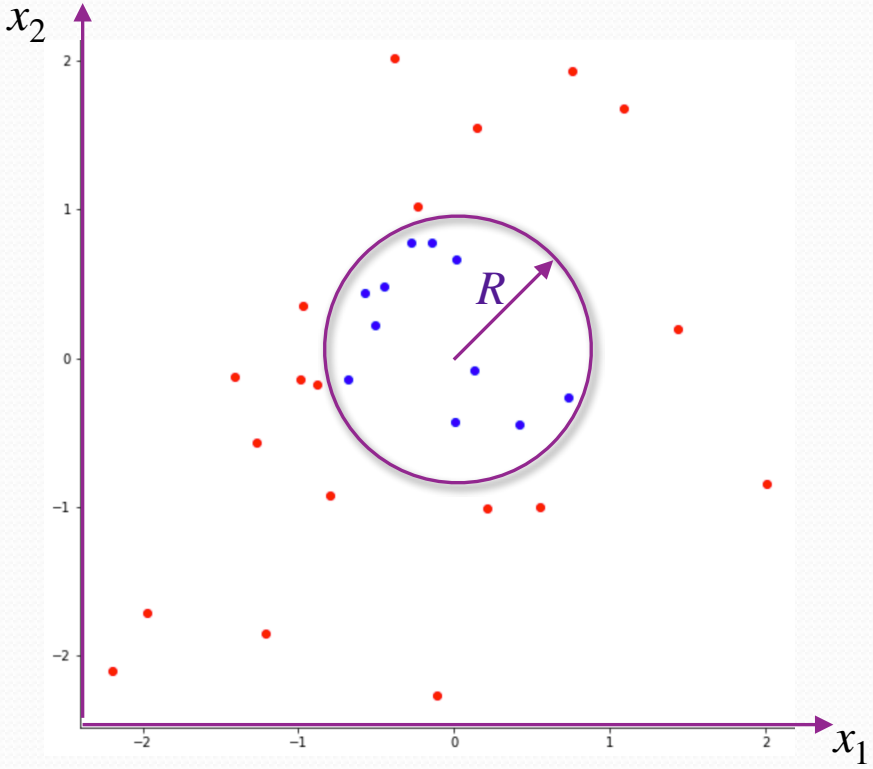


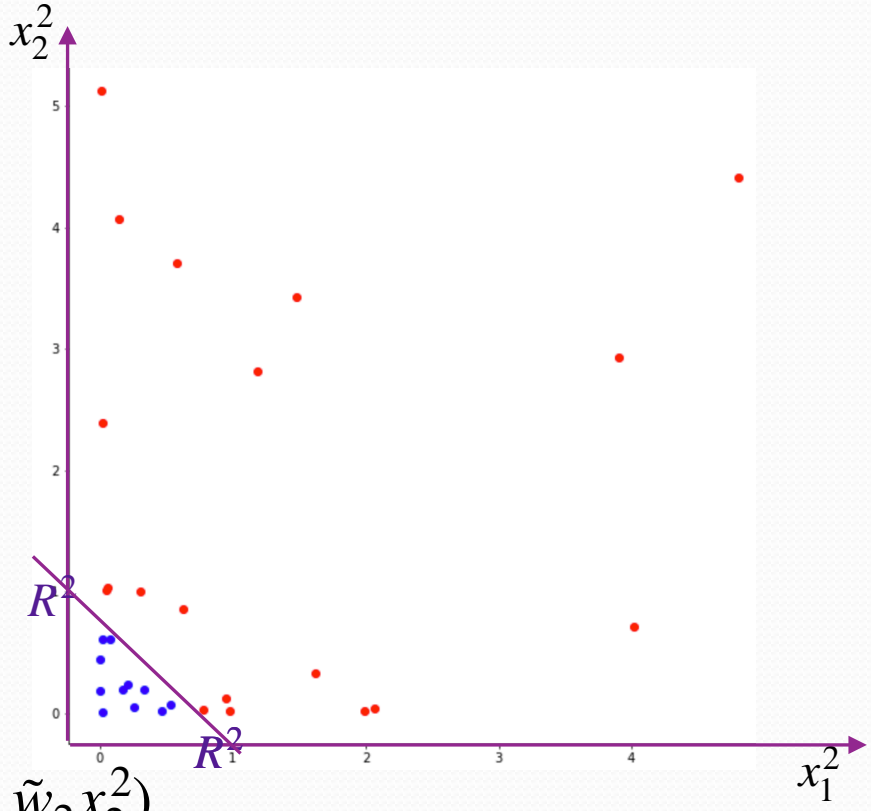


$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

$$\Phi(\mathbf{x}) = \begin{bmatrix} 1 \\ (x_1)^2 \\ (x_2)^2 \end{bmatrix} = \begin{bmatrix} 1 \\ z_1 \\ z_2 \end{bmatrix} \qquad \tilde{\mathbf{w}} = \begin{bmatrix} -R^2 \\ 1 \\ 1 \end{bmatrix}$$

Prediction

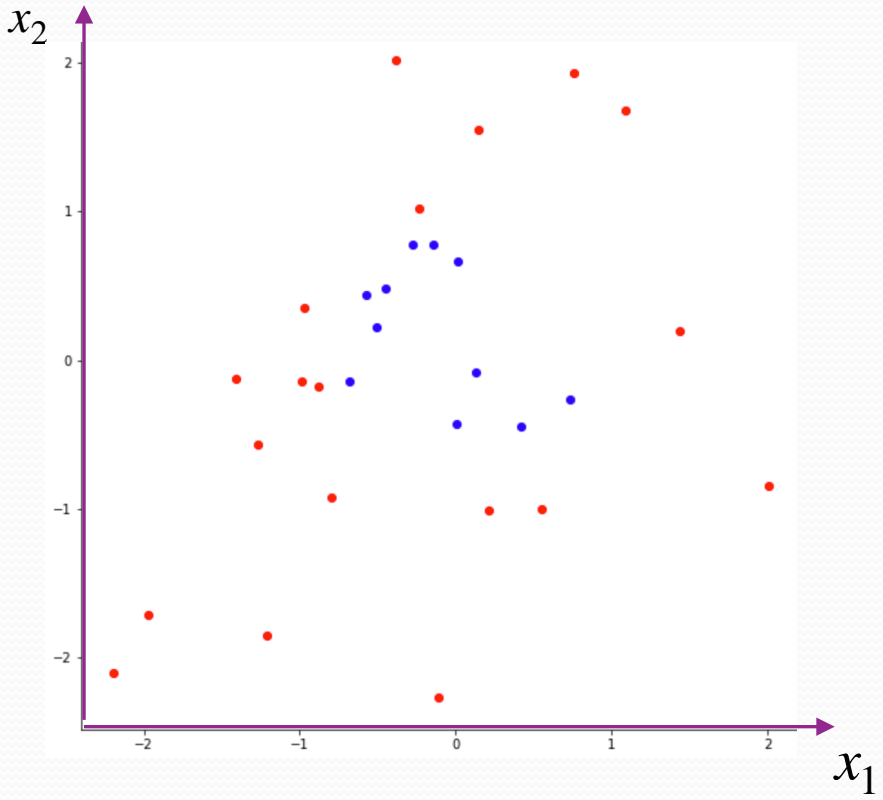


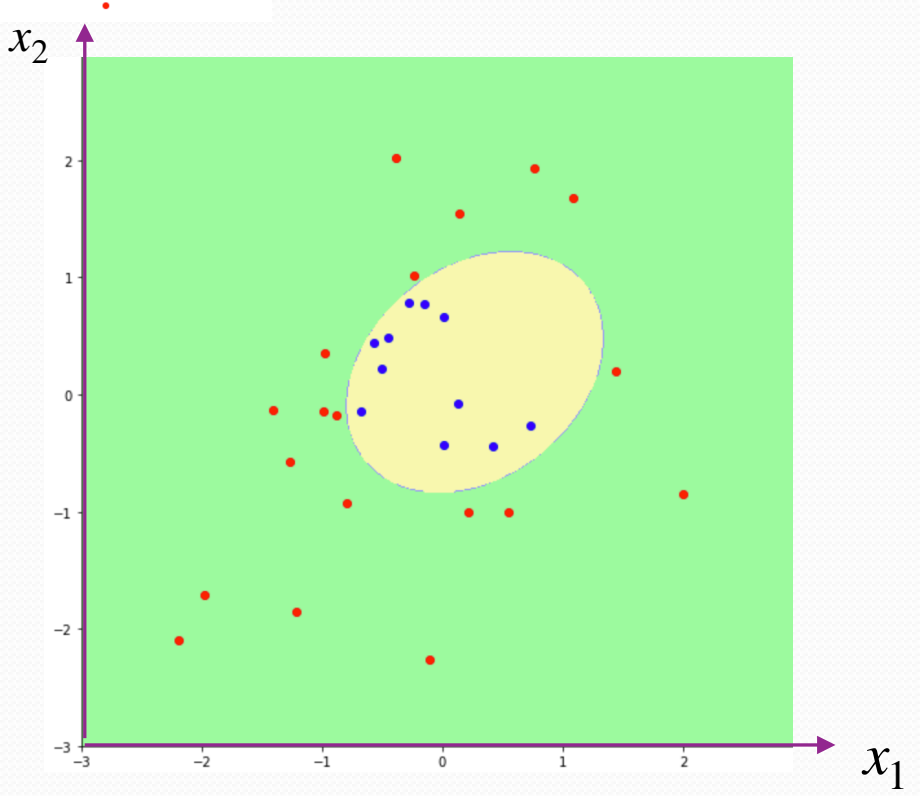


$$pr(y = 1 \mid \mathbf{x}) = \sigma(\tilde{w}_0 + +\tilde{w}_2 x_1^2 + \tilde{w}_2 x_2^2)$$

If $pr(y = 1 \mid \mathbf{x}) = \sigma(-R^2 + x_1^2 + x_2^2) \leq 0$ .5 predict blue

If $pr(y = 1 \mid \mathbf{x}) = \sigma(-R^2 + x_1^2 + x_2^2)$ > 0.5 predict red





Transform examples into Z-space:

$$\Phi_2(\mathbf{x}) = \Phi_2([1, x_1, x_2]^T) = [1, x_1, x_1^2, x_2, x_2^2, x_1 x_2]^T = [1, z_1, z_2, z_3, z_4, z_5]^T$$

Learn in z-space:

$$\tilde{\mathbf{w}} = \begin{bmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \tilde{w}_2 \\ \tilde{w}_3 \\ \tilde{w}_4 \\ \tilde{w}_5 \end{bmatrix}$$

$$pr(y = 1 \mid \mathbf{x}) = \sigma(\tilde{w}_0 + \tilde{w}_1 x_1 + \tilde{w}_2 x_1^2 + \tilde{w}_3 x_2 + \tilde{w}_4 x_2^2 + \tilde{w}_5 x_1 x_2)$$

$$\text{sigmoid}(\tilde{\mathbf{w}}^T \Phi(\mathbf{x}^{(i)}))$$

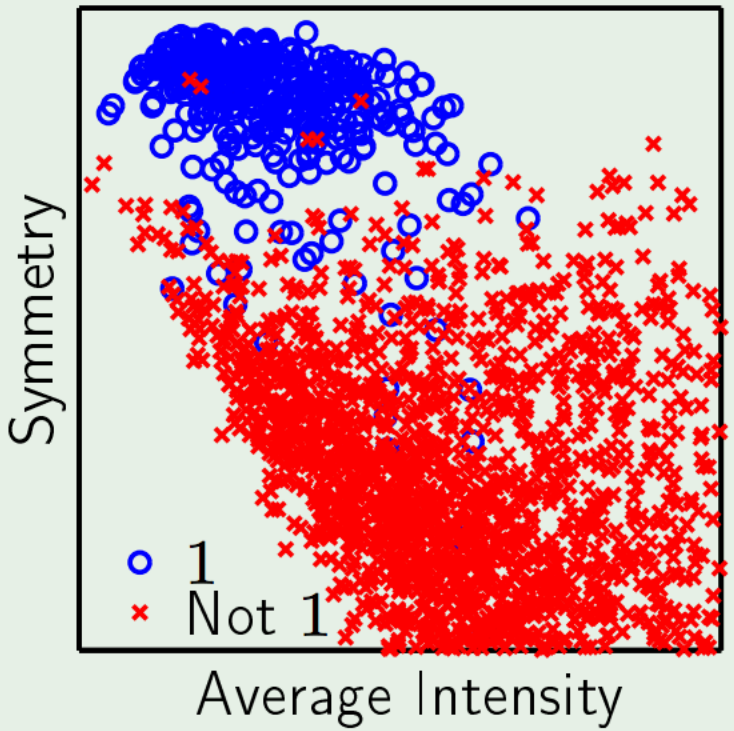
Overfitting...

1. What do you think we should do to prevent overfitting?

- [ ] model selection
- [ ] L1 regularization
- [ ] L2 regularization
- [ ] Do not transform the features
- [ ] Find another algorithm

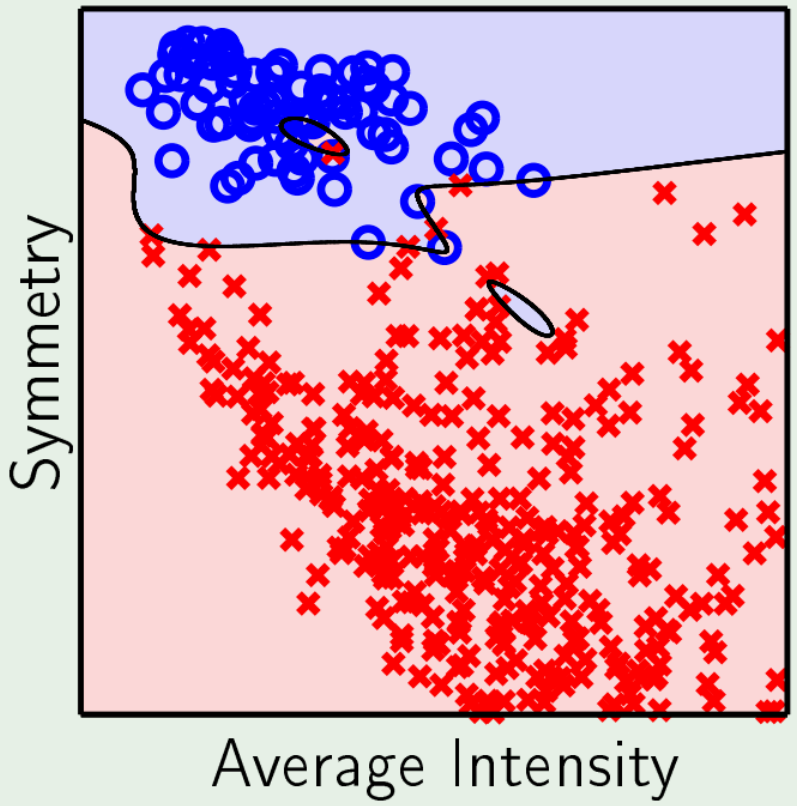
Cross validation in action

Digits classification task



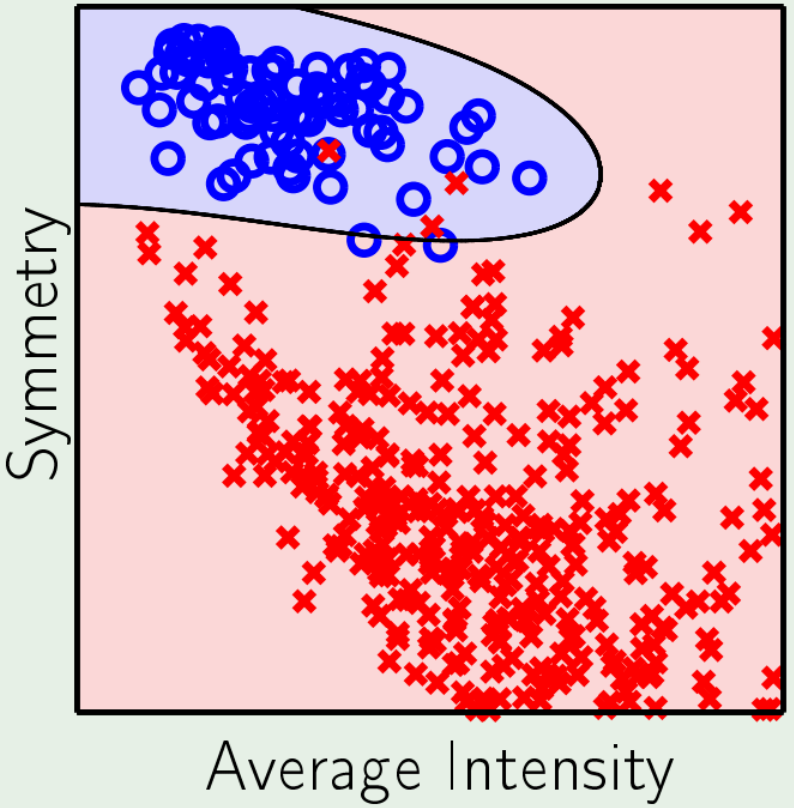
The result

without validation



$E_{\text{in}} = 0\%$ $E_{\text{out}} = 2.5\%$

with validation



$E_{\text{in}} = 0.8\%$ $E_{\text{out}} = 1.5\%$

Regularization

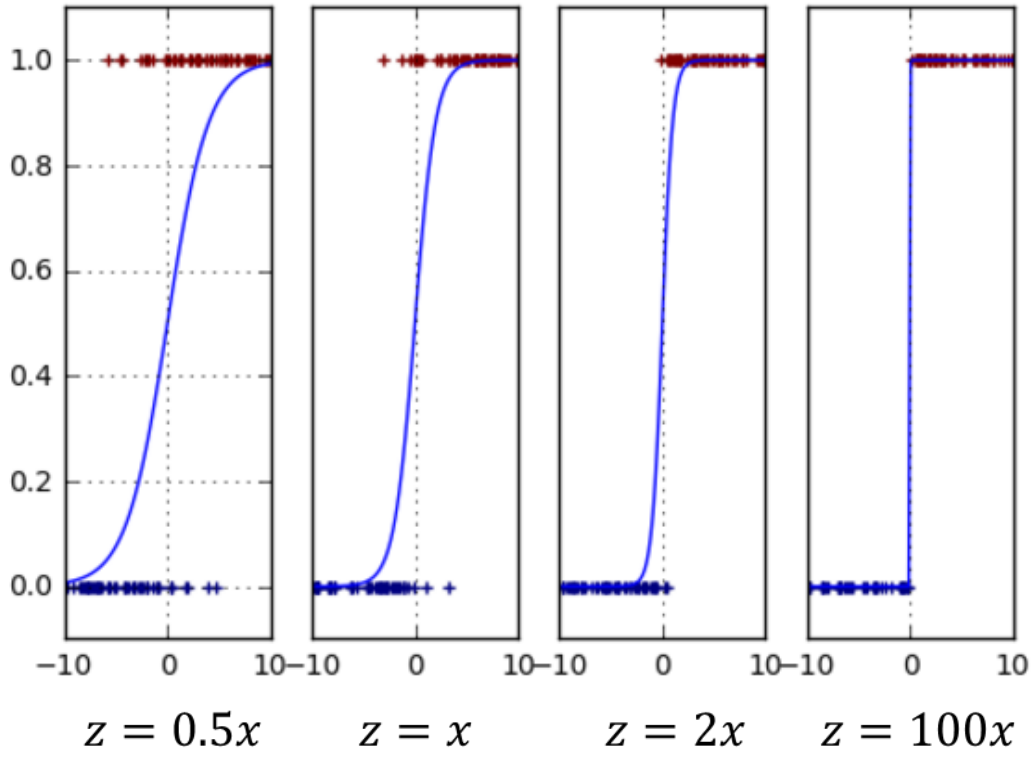
HOW WOULD YOU ADD REGULARIZATION?

THINK OF HOW WE ADDED REGULARIZATION TO LINEAR REGRESSION

$E_{in}(\mathbf{w}) + \lambda(\|\mathbf{w}\|_2^2)$ $E_{in}(\mathbf{w}) + \lambda(\|\mathbf{w}\|_1)$

How you would you add regularization to logistic regression? Use $\ell(w)$ to be the log likelihood function

Logistic Model as a "Soft" Classifier



- Plot of

$$P(y = 1|x) = \frac{1}{1 + e^{-z}}, \qquad z = w_1 x$$

  - Markers are random samples

- Higher w_1: prob transition becomes sharper
  - Fewer samples occur across boundary

- As $w_1 \to \infty$ logistic becomes "hard" rule

$$P(y = 1|x) \approx \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

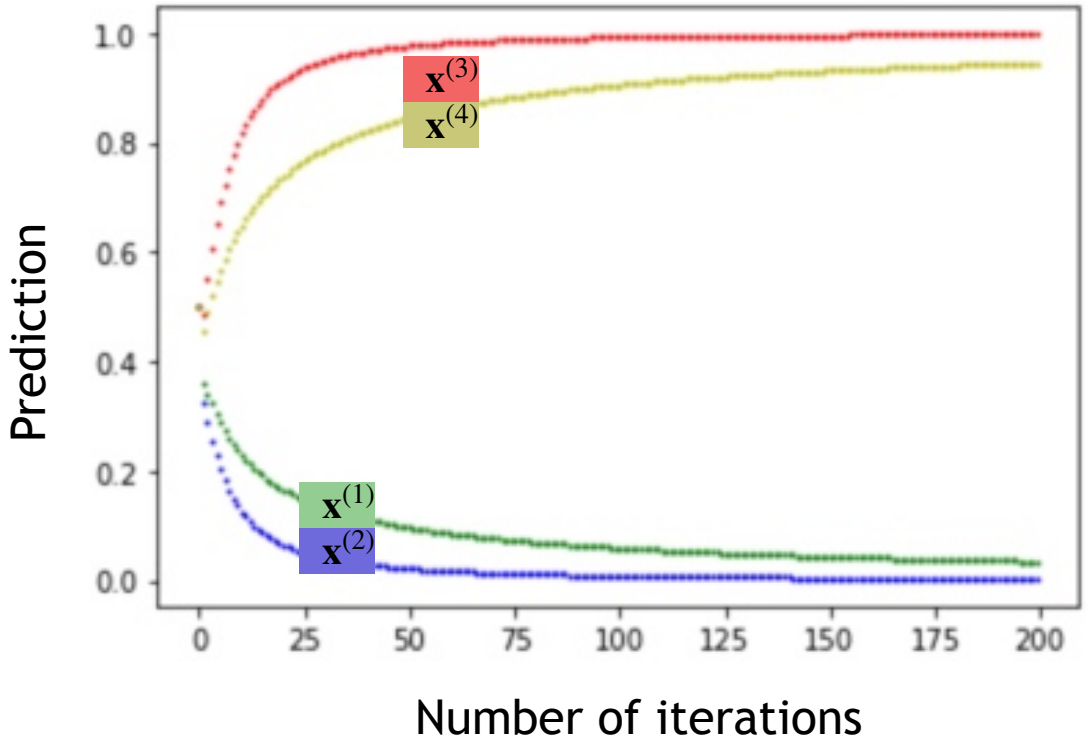
Gradient Ascent

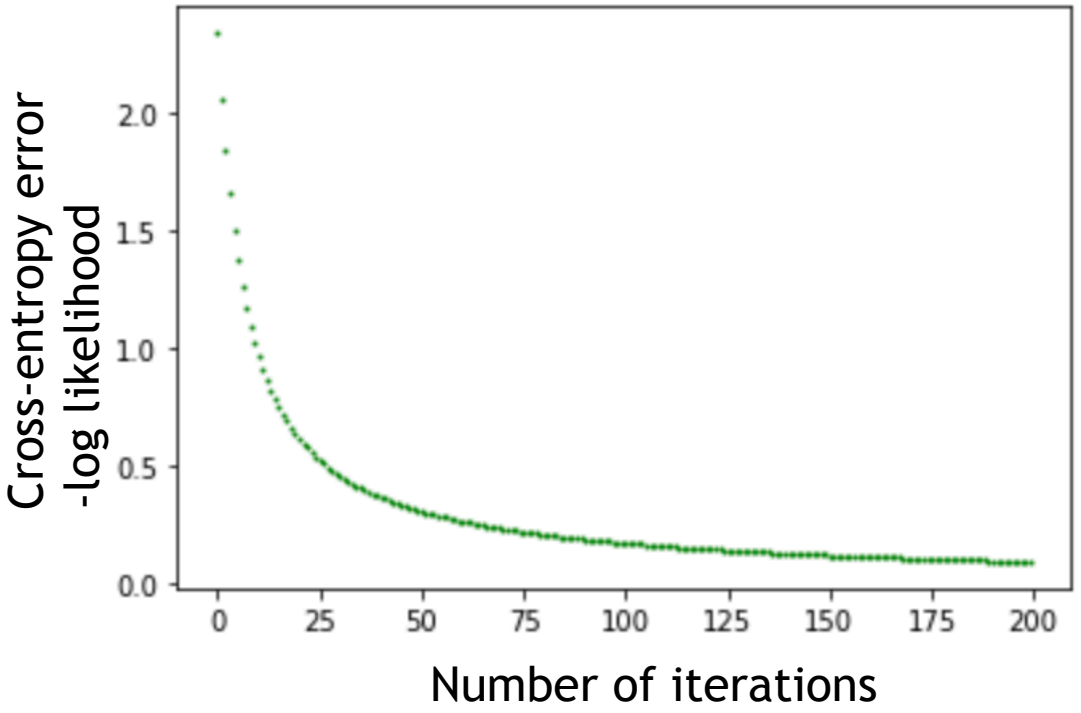
$\mathbf{x}^{(1)} = [1 \;\; 3 \;\; 4]^T$
$\mathbf{x}^{(2)} = [1 \;\; 3 \;\; 5]^T$

Label 0 examples

$\mathbf{x}^{(3)} = [1 \;\; 4 \;\; 1]^T$
$\mathbf{x}^{(4)} = [1 \;\; 3 \;\; 1.5]^T$

Label 1 examples





Adding L2 regularization: $\frac{1}{N}\ell(\mathbf{w}) - \lambda(\|\mathbf{w}_{1:d}\|_2^2)$

- The log likelihood function for logistic regression

Fit

Regularization term

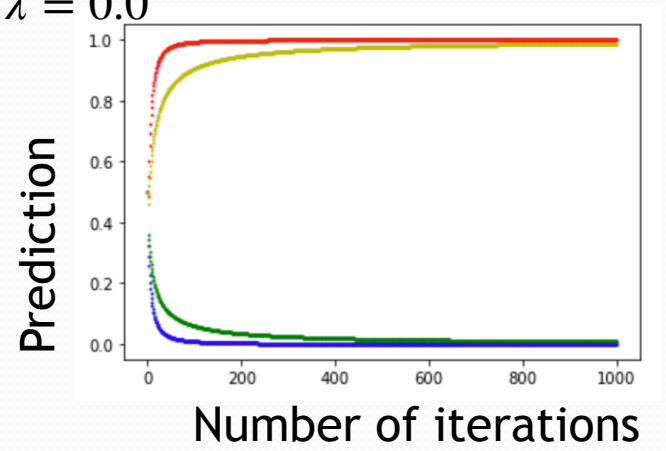
$$\ell_{\text{ridge}}(\mathbf{w}) = \frac{1}{N}\log L(\mathbf{w}) - \lambda(\|\mathbf{w}_{1:d}\|_2^2)$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}\log\sigma(\mathbf{w}^T\mathbf{x}^{(i)}) + (1-y^{(i)})\log(1-\sigma(\mathbf{w}^T\mathbf{x}^{(i)}))\right) - \lambda(\|\mathbf{w}_{1:d}\|_2^2)$$

- Perform gradient ascent on this regularized function

- Typically, we don't want to restrict the size of the intercept term

$\lambda = 0.0$



$\mathbf{x}^{(1)} = [1 \;\; 3 \;\; 4]^T$

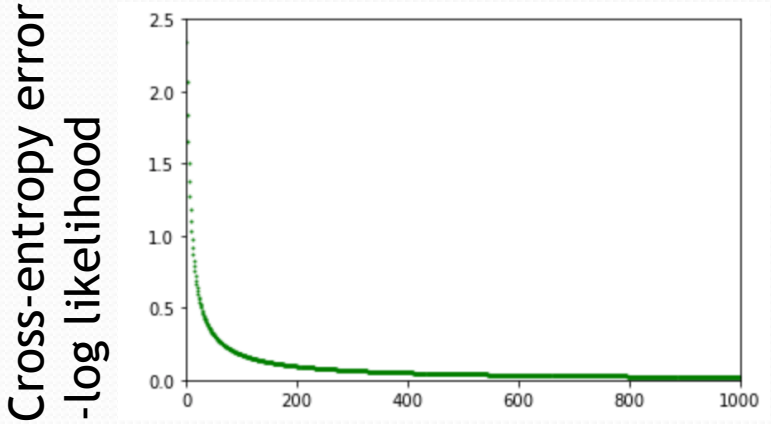
$\mathbf{x}^{(2)} = [1 \;\; 3 \;\; 5]^T$

Label 0 examples

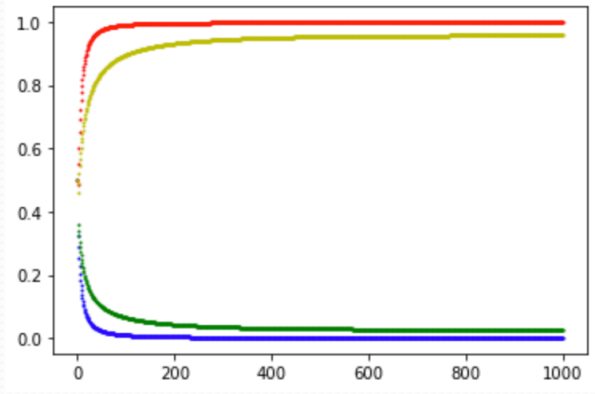
$\mathbf{x}^{(3)} = [1 \;\; 4 \;\; 1]^T$

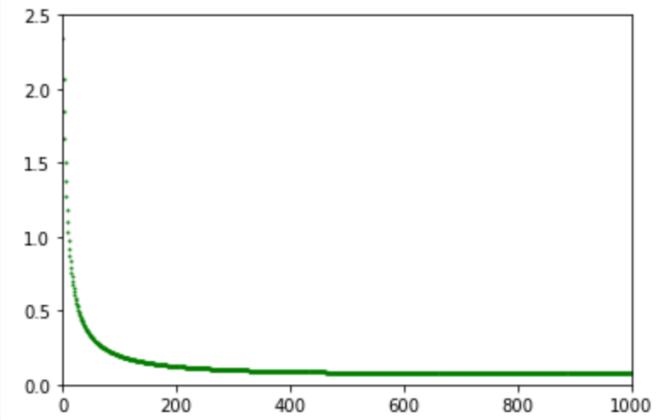
$\mathbf{x}^{(4)} = [1 \;\; 3 \;\; 1.5]^T$

Label 1 examples

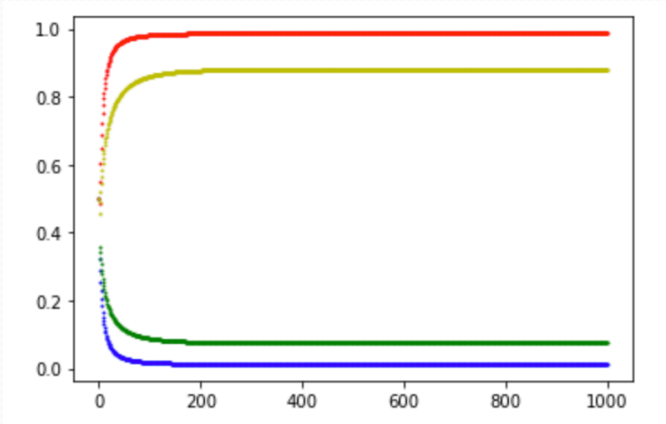


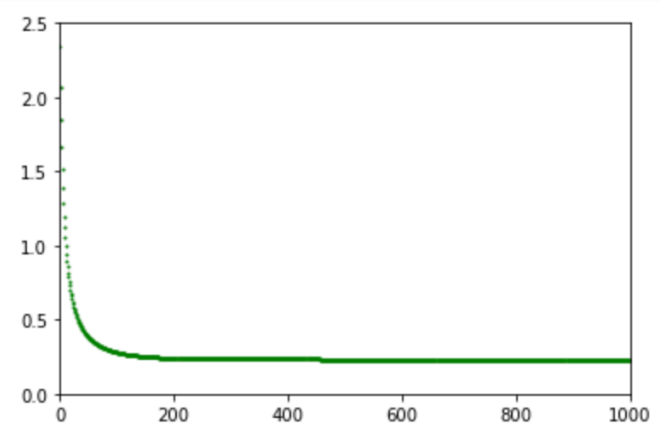
$\lambda = 0.01$





$\lambda = 0.05$





Outline

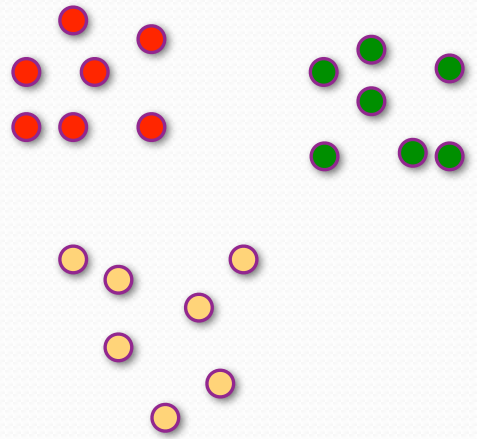
- Motivating example: How can we classify? — How can we use a hyperplane for a classification problem?
- Estimating probabilities — Can we predict not only which class an example belongs to - but a confidence score of that classification
- Maximum likelihood — How can we find the most likely hyperplane? Could we write a function to describe how likely a hyperplane was to have generated the dataset?
  - Iterative approach - gradient ascent — Maximizing the function
- Thinking about different types of error — Some errors are more costly than other errors. Can we modify our predictions to decrease one type of error (and perhaps increase another type of error?
  - Accuracy
  - Motivating example
- Transformation of the features — Extending our algorithm to nonlinear decision boundaries
- Multiple classes — What if we have more than two classes?

- Multi-class: every example belongs to **one** class.
- Multi-label: every example can belong to **one or more** classes

What if there is more than two classes?

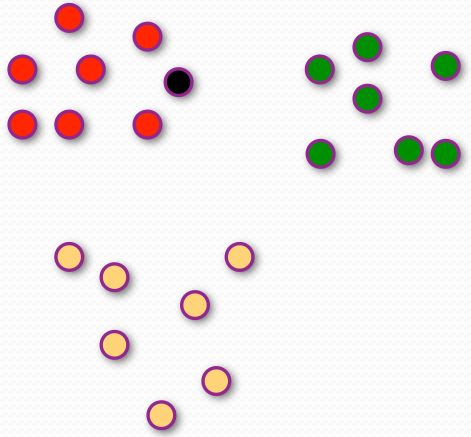
Multi-class classification

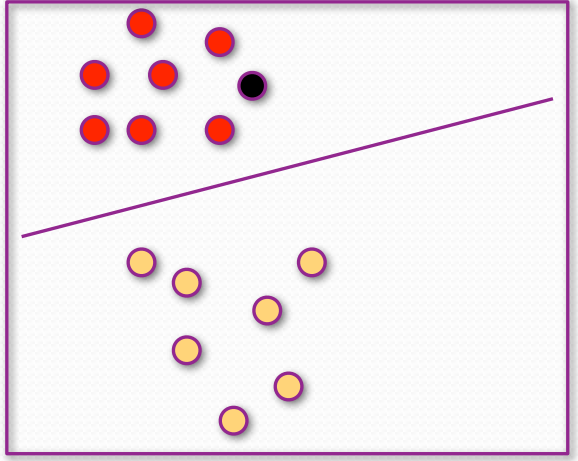
- Versicolor Iris vs Setosa Iris vs Veronica Iris
- 0 vs 1 vs 2 vs 3 … vs 9
- Dog vs Cat vs Mouse
- …



One-versus-One Approach

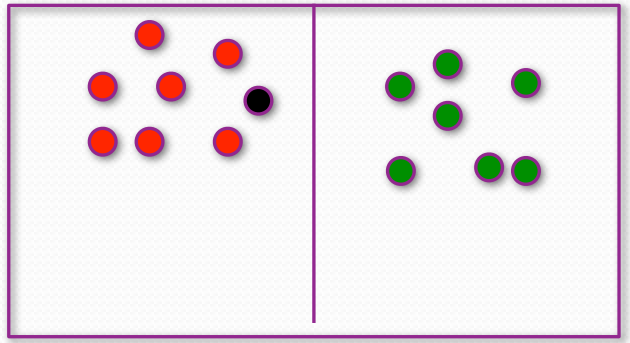
Turn multiple classes problem $C_1, C_2, \ldots, C_K$ into $\frac{K(K-1)}{2}$ binary classification problems





For each pair C_i, C_j of classes

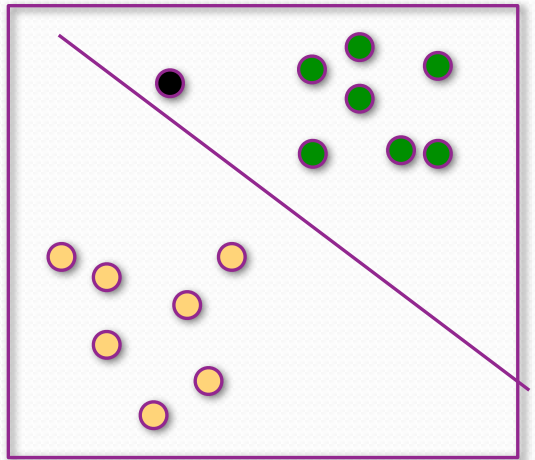
- Relabel training examples with label C_i into '1'
- Relabel training examples with label C_j into '0'
- Remove all other training examples



How do we predict given $\frac{K(K-1)}{2}$ binary classifiers? (i.e. K decision boundaries)

Step 1) Given a new example $\mathbf{x}$, predict the class that wins "majority of votes"

Step 2) If there is a tie, use confidence scores to resolve ties

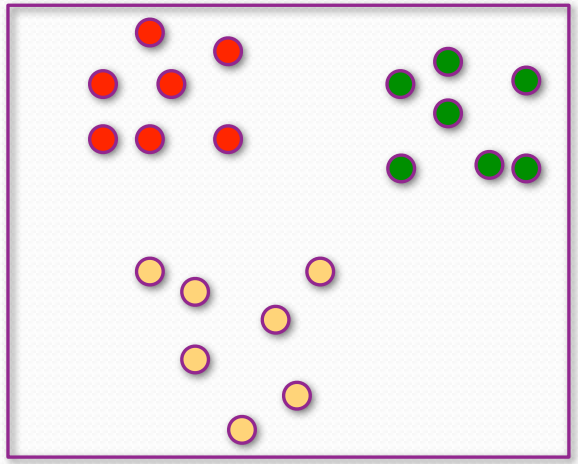


One-versus-Rest or One-Versus All Approach

Turn multiple classes problem $C_1, C_2, \ldots, C_K$ into K binary classification problems

- Relabel training examples with label C_i into '1'
- Relabel all other training examples into '0'

Repeat K times using logistic regression to classify

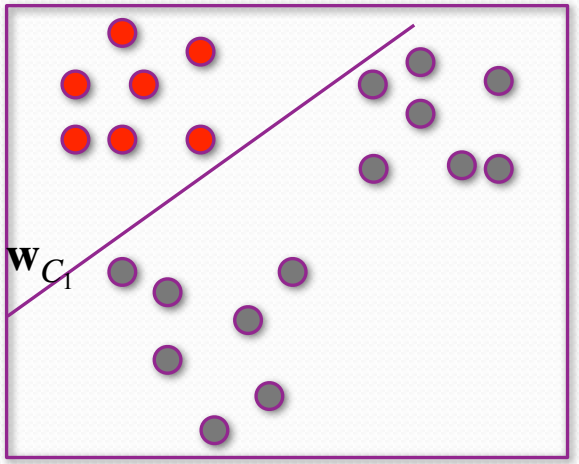


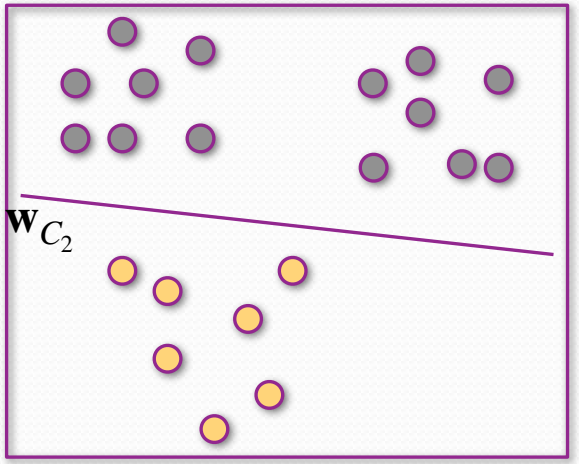
How do we predict given K binary classifiers? (i.e. K decision boundaries)

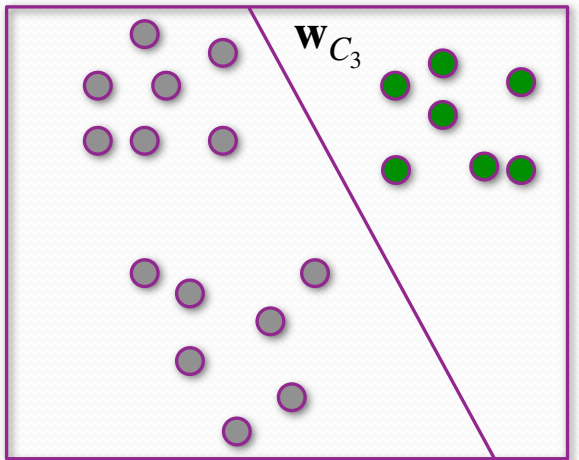
Use confidence estimates $p(y = C_i \mid \mathbf{x})$ for $i = 1 \ldots K$

Predict C_i^* where

$$i = \arg \max_{i \in 1,\ldots,K} p(y = C_i \mid \mathbf{x}) = \sigma(\mathbf{w}_{C_i}^T \mathbf{x})$$







Works well in practice

Learning over local correctness - does not guarantee good global performance

One-versus-One Approach

K(K-1)/2 classifiers. Slow if K is large

Trained on a smaller subset of data which can result in variance

One-versus-Rest or One-Versus All Approach

K classifiers

In-balance in number of class 1 and class 0 training examples

Could our algorithm directly estimate the probability of label belonging to each of the K classes?
(i.e. don't resort to a binary classification problem)

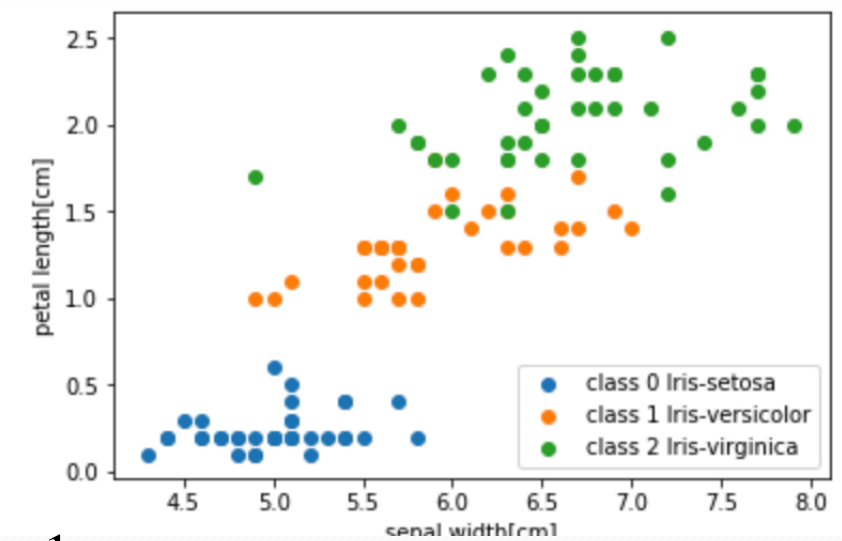
Computing $\mathbf{w}_1$, $\mathbf{w}_3$, $\mathbf{w}_2$, …, $\mathbf{w}_K$ separately doesn't work since $\sum_{i=1}^{K} p(y = C_i \mid \mathbf{x}) = \sum_{i=1}^{K} \sigma(\mathbf{w}_i^T \mathbf{x}) \neq 1$

We can add a Normalization term to make the probabilities sum to 1 and learn the probabilities jointly (to keep the sum =1)

Setup

Predict when multiple classes: $C_1, C_2, \ldots, C_K$ (i.e. K different classes)

- Versicolor Iris vs Setosa Iris vs Veronica Iris
- 0 vs 1 vs 2 vs 3 … vs 9
- Dog vs Cat vs Mouse
- …



Goal:

Estimate $p(y = C_i \mid \mathbf{x})$ for $i = 1 \ldots K$ such that $\sum_{i=1}^{K} p(y = C_i \mid \mathbf{x}) = 1$

$$h(\mathbf{x}) = \begin{bmatrix} p(y = C_1 \mid \mathbf{x}) \\ p(y = C_2 \mid \mathbf{x}) \\ p(y = C_3 \mid \mathbf{x}) \\ \cdots \\ p(y = C_K \mid \mathbf{x}) \end{bmatrix}$$

$$h(\mathbf{x}) = \begin{bmatrix} p(y = C_1 \mid \mathbf{x}) \\ p(y = C_2 \mid \mathbf{x}) \\ p(y = C_3 \mid \mathbf{x}) \\ \cdots \\ p(y = C_K \mid \mathbf{x}) \end{bmatrix}$$

We will predict K different probabilities: $y^{(i)}$ becomes $\mathbf{y}^{(i)} = [y_1^{(i)}, y_2^{(i)}, \ldots, y_K^{(i)}]^T$

Using 1-of-K encoding (one hot encoding).

$$y_j^{(i)} = \begin{cases} 1 & y^{(i)} = j \\ 0 & \textbf{otherwise} \end{cases}$$

If we have 3 classes:

class C_1 → $[1 \quad 0 \quad 0]^T$

class C_2 → $[0 \quad 1 \quad 0]^T$

class C_3 → $[0 \quad 0 \quad 1]^T$

Any ideas?

$$h(\mathbf{x}) = \begin{bmatrix} p(y = C_1 \mid \mathbf{x}) \\ p(y = C_2 \mid \mathbf{x}) \\ p(y = C_3 \mid \mathbf{x}) \\ \cdots \\ p(y = C_K \mid \mathbf{x}) \end{bmatrix}$$

Equivalent formula

$$p(y = 1 \mid \mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} \left(\frac{e^{\mathbf{w}^T\mathbf{x}}}{e^{\mathbf{w}^T\mathbf{x}}} \right) = \frac{e^{\mathbf{w}^T\mathbf{x}}}{e^{\mathbf{w}^T\mathbf{x}} + 1}$$

$$h(\mathbf{x}) = \begin{bmatrix} p(y = 0 \mid \mathbf{x}) \\ p(y = 1 \mid \mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{1}{e^{\mathbf{w}^T\mathbf{x}} + 1} \\ \frac{e^{\mathbf{w}^T\mathbf{x}}}{e^{\mathbf{w}^T\mathbf{x}} + 1} \end{bmatrix}$$

- Setosa - C_1
- Versicolor -C_2
- Veronica - C_3

If $\mathbf{w}_{C_1}^T\mathbf{x} = -18$, $\mathbf{w}_{C_2}^T\mathbf{x} = 12$, **and** $\mathbf{w}_{C_3}^T\mathbf{x} = 6$

We predict **Versicolor**

We could turn these numbers to probabilities using softmax (well-formed conditional probabilities)

$$p(y = C_1 \mid \mathbf{x}) = \frac{e^{-18}}{e^{-18} + e^{12} + e^{6}} < 1$$

$$p(y = C_2 \mid \mathbf{x}) = \frac{e^{12}}{e^{-18} + e^{12} + e^{6}} < 1$$

$$p(y = C_3 \mid \mathbf{x}) = \frac{e^{6}}{e^{-18} + e^{12} + e^{6}} < 1$$

Normalization term makes the probabilities sum to 1

- Multi-class: every example belongs to **one** class.
- Multi-label: every example can belong to **one or more** classes

SOFTMAX- A GENERALIZED FORM OF LOGISTIC REGRESSION (THE CLASSES ARE *MUTUALLY EXCLUSIVE -IE MULTI-CLASS*)

Called:
multiclass logistic regression, *softmax regression* and *multinomial regression*

How can we modify our existing model ?

Our existing model gave a probability of belonging to a class.

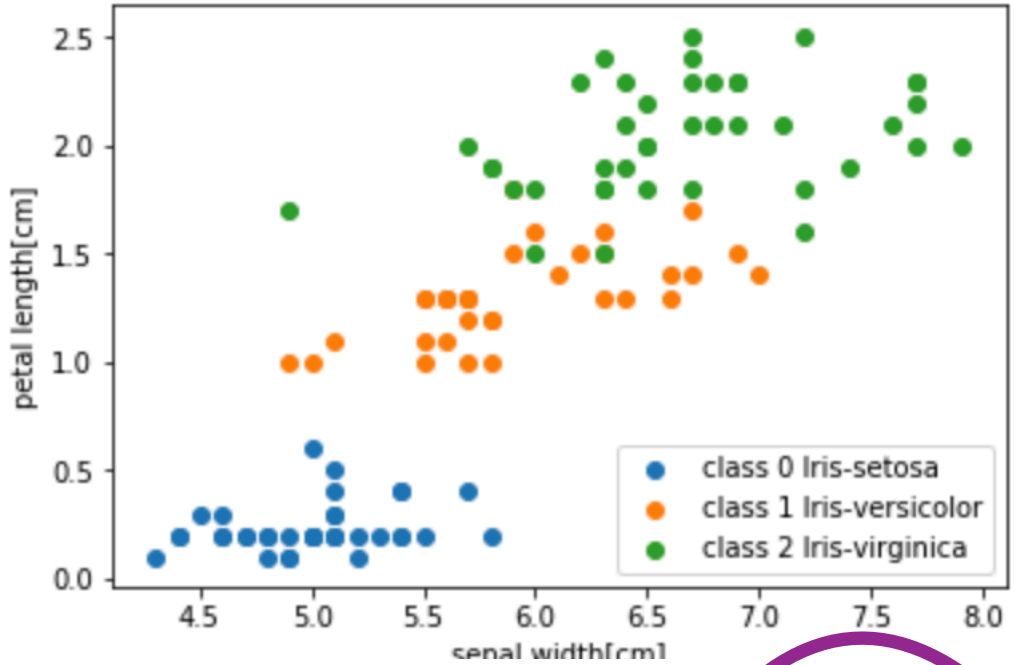
How can we provide an estimate of K classes? If K = 3 we can represent each class as follows:

$[1 \quad 0 \quad 0]^T$ $[0 \quad 1 \quad 0]^T$ $[0 \quad 0 \quad 1]^T$

class Setosa

class Versicolor

class Virginia



$$X = \begin{bmatrix} 1 & 5.8 & 2.4 \\ 1 & 6 & 1 \\ 1 & 5.5 & 0.2 \\ 1 & 7.3 & 1.8 \\ 1 & 5 & 0.2 \\ 1 & 6.3 & 2.5 \\ 1 & 5 & 0.3 \\ 1 & 6.7 & 1.5 \\ 1 & 6.8 & 1.4 \\ 1 & 6.1 & 1.3 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 0 \\ 2 \\ 0 \\ 0 \\ 1 \\ 2 \\ 2 \end{bmatrix} \quad \begin{bmatrix} [0,1,0] \\ [0,1,0] \\ [0,0,1] \\ [1,0,0] \\ [0,0,1] \\ [1,0,0] \\ [1,0,0] \\ [0,1,0] \\ [0,0,1] \\ [0,0,1] \end{bmatrix}$$

We will predict K different probabilities: $y^{(i)}$ becomes $\mathbf{y}^{(i)} = [y_1^{(i)}, y_2^{(i)}, \ldots, y_K^{(i)}]^T$

Using 1-of-K encoding (one hot encoding).

$$y_j^{(i)} = \begin{cases} 1 & y^{(i)} = j \\ 0 & \textbf{otherwise} \end{cases}$$

Approach is maximize Log likelihood

Only one of these is 1

Maximize $\log L = \sum_{i=1}^{N} \log p(y^{(i)} \mid \mathbf{x}^{(i)}) = \sum_{i=1}^{N} \log \prod_{k=1}^{K} p(C_k^{(i)} \mid \mathbf{x}^{(i)})^{y_k^{(i)}} = \sum_{i=1}^{N} \sum_{k=1}^{K} y_k^{(i)} \log p(C_k^{(i)} \mid \mathbf{x}^{(i)})$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} y_k^{(i)} \log \frac{e^{\mathbf{w}_k^T \mathbf{x}^{(i)}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T \mathbf{x}^{(i)}}}$$

Negative Log likelihood for softmax is the cross entropy error

Convex thus parameters can be found using gradient ascent

Logistic Regression

$$\{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\} \qquad y^{(i)} \in \{0,1\}$$

$$\mathbf{w}^T\mathbf{x} = z$$

$$p(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} = \frac{e^{\mathbf{w}^T\mathbf{x}}}{1 + e^{\mathbf{w}^T\mathbf{x}}}$$

W has dimension k x (d+1), we stacked the transpose of the coefficient vectors

Soft-max

$$\{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\} \qquad y^{(i)} \in \{1,2,\ldots,K\}$$

$$p(y = j \mid \mathbf{x}) \qquad j \in \{1,2,\ldots,K\}$$

$$W = \begin{bmatrix} -\mathbf{w}_1^T- \\ -\mathbf{w}_2^T- \\ -\mathbf{w}_3^T- \\ \cdots \\ -\mathbf{w}_K^T- \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdots \\ x_d \end{bmatrix}$$

$$\begin{bmatrix} -\mathbf{w}_1^T- \\ -\mathbf{w}_2^T- \\ -\mathbf{w}_3^T- \\ \cdots \\ -\mathbf{w}_K^T- \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdots \\ x_d \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T\mathbf{x} \\ \mathbf{w}_2^T\mathbf{x} \\ \mathbf{w}_3^T\mathbf{x} \\ \cdots \\ \mathbf{w}_K^T\mathbf{x} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \cdots \\ z_K \end{bmatrix}$$

score for each coefficient Vector

$$h(\mathbf{x}) = \begin{bmatrix} p(y = 1 \mid \mathbf{x}) \\ p(y = 2 \mid \mathbf{x}) \\ p(y = 3 \mid \mathbf{x}) \\ \cdots \\ p(y = K \mid \mathbf{x}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\mathbf{w}_j^T\mathbf{x}}} \begin{bmatrix} e^{\mathbf{w}_1^T\mathbf{x}} \\ e^{\mathbf{w}_2^T\mathbf{x}} \\ e^{\mathbf{w}_3^T\mathbf{x}} \\ \cdots \\ e^{\mathbf{w}_K^T\mathbf{x}} \end{bmatrix}$$

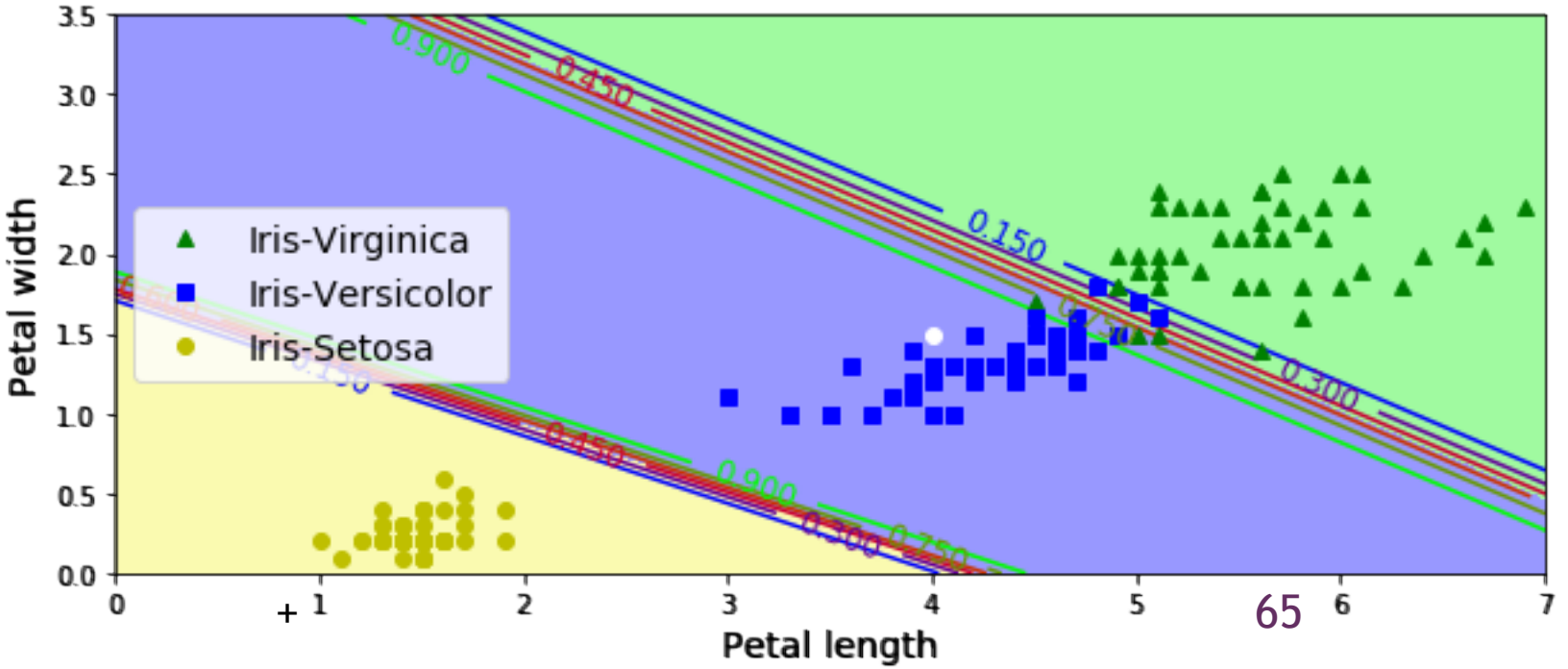
normalizing the score

Example

Suppose we are clarifying which iris out of 3 choices. If we have fit the model and thus have created W

$$\left(P(y = i \mid \mathbf{x})\right) = \left(\frac{e^{\mathbf{w}_i^T X_i}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T X_i}}\right)$$

W			x	Z score	e^{score}	normalize divide by 250750
41.09	-8.08	-18.02	1	-18.28	1.15E-08	4.60E-14
2.09	1.16	3.79	4	12.43	250404	0.9986
-43.18	6.92	14.23	1.5	5.84	346	0.00137



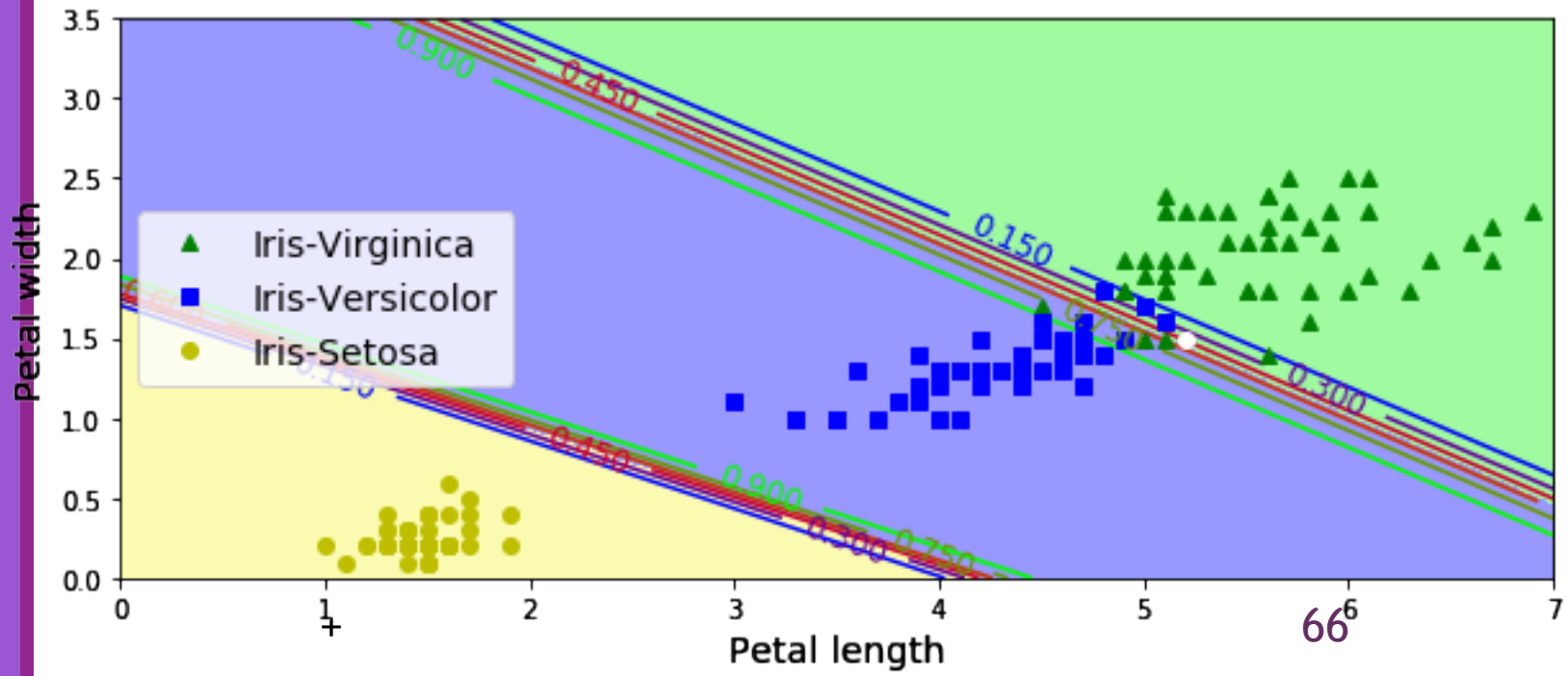
Code to create graphics adapted from Hands-On Machine Learning with Scikit-Learn & TensorFlow

Example

Suppose we are clarifying which iris out of 3 choices. If we have fit the model and thus have created **W**

$$\left(P(y = i \mid \mathbf{x})\right) = \left(\frac{e^{\mathbf{w}_i^T X_i}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T X_i}}\right)$$

W			x	Z score	e^{score}	normalize divide by 2410413.2
41.09	-8.08	-18.02	1	-27.98	7.06E-13	2.93E-19
2.09	1.16	3.79	5.2	13.83	1013275.8	0.420
-43.18	6.92	14.23	1.5	14.15	1397137.3	0.580

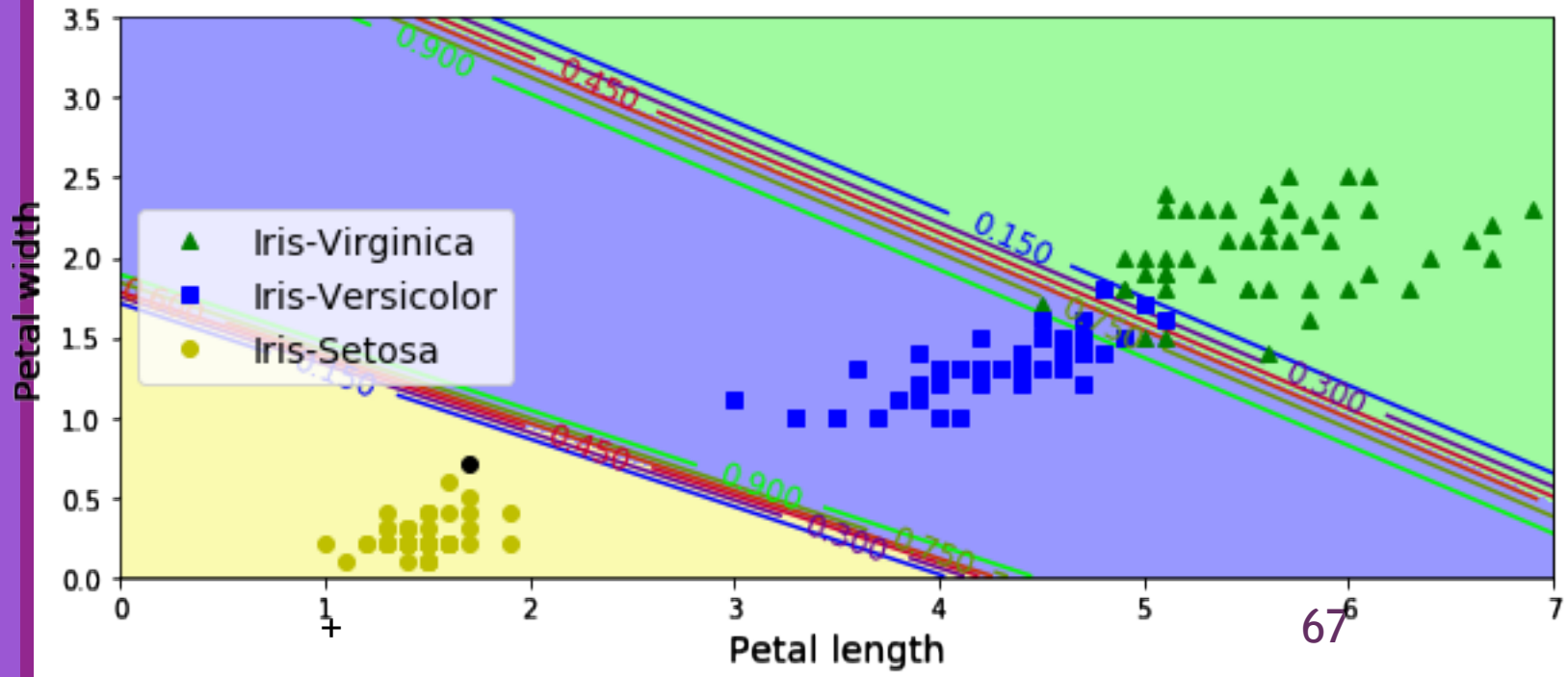


Example

Suppose we are clarifying which iris out of 3 choices. If we have fit the model and thus have created W

$$\left(P(y = i \mid \mathbf{x})\right) = \left(\frac{e^{\mathbf{w}_i^T X_i}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T X_i}}\right)$$

W			x		Z score	e^{score}	normalize divide by 2502575.5
41.09	-8.08	-18.02	1	=	14.73	2501744.5	0.999679
2.09	1.16	3.79	1.7		6.72	830.99	0.000332
-43.18	6.92	14.23	0.7		-21.46	4.81E-10	1.92E-16



If K=2 Softmax ➜ Logistic Regression

We can subtract any fixed K dimensional vector from each of the K coefficient vectors w_j

$$p(y_i = j \mid \mathbf{x}; \mathbf{w}) = \frac{e_i^{(\mathbf{w}_j - \theta)^T \mathbf{x}}}{\sum_{i=1}^{K} e^{(\mathbf{w}_i - \theta)^T \mathbf{x}}}$$

$$p(y_i = j \mid \mathbf{x}; \mathbf{w}) = \frac{e_i^{(\mathbf{w}_j)^T \mathbf{x}} e_i^{(-\theta)^T \mathbf{x}}}{\sum_{i=1}^{K} e_i^{(\mathbf{w}_i)^T \mathbf{x}} e_i^{(-\theta)^T \mathbf{x}}}$$

$$p(y_i = j \mid \mathbf{x}; \mathbf{w}) = \frac{e_i^{(\mathbf{w}_j)^T \mathbf{x}}}{\sum_{i=1}^{K} e^{(\mathbf{w}_i)^T \mathbf{x}}}$$

For $y \in \{1,2\}$

$$h(\mathbf{x}) = \begin{bmatrix} p(y=1 \mid \mathbf{x}) \\ p(y=2 \mid \mathbf{x}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{2} e^{\mathbf{w}_j^T \mathbf{x}}} \begin{bmatrix} e^{\mathbf{w}_1^T \mathbf{x}} \\ e^{\mathbf{w}_2^T \mathbf{x}} \end{bmatrix}$$

$$= \frac{1}{e^{\vec{0}^T \mathbf{x}} + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \begin{bmatrix} e^{\vec{0}^T \mathbf{x}} \\ e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}} \end{bmatrix}$$

$$= \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \begin{bmatrix} 1 \\ e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \\ \frac{e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \end{bmatrix} = \begin{bmatrix} \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \\ 1 - \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \end{bmatrix} = \begin{bmatrix} \frac{1}{1 + e^{(\mathbf{w}')^T \mathbf{x}}} \\ 1 - \frac{1}{1 + e^{(\mathbf{w}')^T \mathbf{x}}} \end{bmatrix}$$

If K=2 Softmax ➜ Logistic Regression

We can subtract any fixed K dimensional vector from each of the K coefficient vectors w_j

$$p(y_i = j \mid \mathbf{x}; \mathbf{w}) = \frac{e_i^{(\mathbf{w}_j - \theta)^T \mathbf{x}}}{\sum_{i=1}^{K} e^{(\mathbf{w}_i - \theta)^T \mathbf{x}}}$$

$$p(y_i = j \mid \mathbf{x}; \mathbf{w}) = \frac{e_i^{(\mathbf{w}_j)^T \mathbf{x}} e_i^{(-\theta)^T \mathbf{x}}}{\sum_{i=1}^{K} e_i^{(\mathbf{w}_i)^T \mathbf{x}} e_i^{(-\theta)^T \mathbf{x}}}$$

$$p(y_i = j \mid \mathbf{x}; \mathbf{w}) = \frac{e_i^{(\mathbf{w}_j)^T \mathbf{x}}}{\sum_{i=1}^{K} e^{(\mathbf{w}_i)^T \mathbf{x}}}$$

over-parameterized there are multiple parameters that give the same answer

For $y \in \{1,2\}$

$$h(\mathbf{x}) = \begin{bmatrix} p(y=1 \mid \mathbf{x}) \\ p(y=2 \mid \mathbf{x}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{2} e^{\mathbf{w}_j^T \mathbf{x}}} \begin{bmatrix} e^{\mathbf{w}_1^T \mathbf{x}} \\ e^{\mathbf{w}_2^T \mathbf{x}} \end{bmatrix}$$

$$\frac{1}{+ e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \begin{bmatrix} e^{\vec{0}^T \mathbf{x}} \\ e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}} \end{bmatrix}$$

$$\frac{1}{+ e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \begin{bmatrix} 1 \\ e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \\ \frac{e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \end{bmatrix} = \begin{bmatrix} \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \\ 1 - \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}}} \end{bmatrix} = \begin{bmatrix} \frac{1}{1 + e^{(\mathbf{w}')^T \mathbf{x}}} \\ 1 - \frac{1}{1 + e^{(\mathbf{w}')^T \mathbf{x}}} \end{bmatrix}$$

Using Sklearn's Logistic Regression

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(solver='lbfgs',multi_class='multinomial')

clf.fit(X_train, y_train)

yhat_test = clf.predict(X_test)

score = clf.score(X_test, y_test)
```

make an instance of the model
Parameters set for this instance:
multi_class = 'multinomial'

Use the coefficient vector to predict

compute the accuracy

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Using Sklearn's Logistic Regression

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(solver='lbfgs',multi_class='multinomial')

clf.fit(X_train, y_train)

yhat_test = clf.predict(X_test)

score = clf.score(X_test, y_test)
```

import the model in Sklearn this will implemented as a class

make an instance of the model
Parameters set for this instance:
multi_class = 'multinomial'

Train the model. (i.e. run the algorithm using X_train and y_train to create the coefficient vector

Use the coefficient vector to predict

compute the accuracy

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)#default is 1/4 test

scaler = StandardScaler()
X_train= scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_train = [[ 7.  6. 10.  5.  3. 10.  9. 10.  2.]
           [ 8.  3.  8.  3.  4.  9.  8.  9.  8.]
           [ 8. 10. 10. 10.  6. 10. 10. 10.  1.]
           [ 1.  1.  1.  1.  2.  1.  1.  1.  1.]]
```

```
X_test = [[ 1.  1.  1.  1.  2.  5.  1.  1.  1.]
          [ 3.  1.  1.  1.  2.  1.  2.  1.  1.]
          [ 5.  5.  5.  2.  5. 10.  4.  3.  1.]
          [ 4.  7.  8.  3.  4. 10.  9.  1.  1.]]
```

y_train = [1 1 1 0]

```
y_test = [0 0 1 1]
```

```
[[ 0.92  0.94  2.31  0.77 -0.1   1.81  2.23  2.27  0.25]
 [ 1.28 -0.04  1.63  0.07  0.34  1.53  1.82  1.95  3.75]
 [ 1.28  2.25  2.31  2.53  1.23  1.81  2.63  2.27 -0.34]
 [-1.22 -0.7  -0.74 -0.64 -0.55 -0.68 -0.99 -0.62 -0.34]]
```

```
[[-1.22 -0.7  -0.74 -0.64 -0.55  0.43 -0.99 -0.62 -0.34]
 [-0.51 -0.7  -0.74 -0.64 -0.55 -0.68 -0.59 -0.62 -0.34]
 [ 0.21  0.61  0.62 -0.28  0.78  1.81  0.22  0.02 -0.34]
 [-0.15  1.27  1.63  0.07  0.34  1.81  2.23 -0.62 -0.34]]
```

Next, we create a logistic regression object. By default, the logistic regresssion object will use $L2$ regularization. Here we have selected not to have regularization by using *penalty='none'*. If regularization is used, the parameter `C` states the level of regularization. Higher values of `C` have `less` regularization. We can chose to set explicity the regularization, or have the optimal value determined for us.

Note that $\lambda = 1/C$.

```
logreg = linear_model.LogisticRegression(penalty='none')
```

Penalty choices: L1,L2, L1 and L2. Default is L2

Then, we fit the model.

```
logreg.fit(X_train, y_train)
```

```
LogisticRegression(penalty='none')
```

```
print(logreg.intercept_)
print(logreg.coef_)
```

	feature	slope
0	thick	1.194634
1	size_unif	0.260756
2	shape_unif	0.839409
3	marg	0.615559
4	cell_size	0.592045
5	bare	1.601667
6	chrom	0.843135
7	normal	0.740983
8	mit	0.340539

```
yhat = logreg.predict(X_test)# the predict method will return 0 or 1
```

```
[0 0 1 1]
```

```
yhat_proba = logreg.predict_proba(X_test)
```

```
[[0.99 0.01]
 [1.   0.  ]
 [0.06 0.94]
 [0.01 0.99]]
```

score = clf.score(X_test, y_test)

"Return the mean accuracy"

```
0.947368
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(yhat, y_test)
```

"Confusion matrix whose i-th row and j-th column entry indicates the number of samples with true label being i-th class and predicted label being j-th class."

```
logreg = linear_model.LogisticRegression(solver='liblinear', penalty='l1', C=0.01)
```

There is a better λ

```
logreg = linear_model.LogisticRegression( penalty='l2', C=0.08)
```

	feature	slope
0	thick	1.194634
1	size_unif	0.260756
2	shape_unif	0.839409
3	marg	0.615559
4	cell_size	0.592045
5	bare	1.601667
6	chrom	0.843135
7	normal	0.740983
8	mit	0.340539

```
[0 0 1 1]

[[0.99 0.01]
 [1.   0.  ]
 [0.06 0.94]
 [0.01 0.99]]
```

	feature	slope
0	thick	0.000000
1	size_unif	0.436027
2	shape_unif	0.191333
3	marg	0.000000
4	cell_size	0.000000
5	bare	0.484421
6	chrom	0.000000
7	normal	0.000000
8	mit	0.000000

```
[0 0 1 1]

[[0.56 0.44]
 [0.68 0.32]
 [0.22 0.78]
 [0.15 0.85]]
```

	feature	slope
0	thick	0.622906
1	size_unif	0.500051
2	shape_unif	0.560409
3	marg	0.431736
4	cell_size	0.453271
5	bare	0.892948
6	chrom	0.550934
7	normal	0.503239
8	mit	0.226096

```
[0 0 1 1]

[[0.97 0.03]
 [0.98 0.02]
 [0.17 0.83]
 [0.05 0.95]]
```