CS-UY 4563: Lecture 9
Linear Classifiers, Logistic Regression

NYU Tandon School of Engineering, Prof. Christopher Musco

Two ways to multiply every entry in a matrix by 2.

In [1]:
```
import numpy as np
import time
```

In [2]:
```
n = 10000
d = 10000
X = np.random.randn(n,d)
Y = np.zeros(X.shape)
```

In [3]:
```
t = time.time()
for i in range(1,n):
    for j in range(1,d):
        Y[i,j] = 2*X[i,j]
elapsed = time.time() - t
print(str(elapsed) + " seconds")
```

```
75.23580312728882 seconds
```

In [4]:
```
t = time.time()
Y = 2*X
elapsed = time.time() - t
print(str(elapsed) + " seconds")
```

```
0.20938587188720703 seconds
```

Second one is way faster...

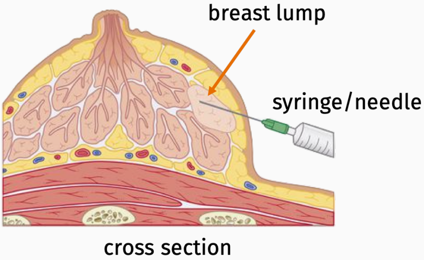
QUICK COMMENT ON PYTHON

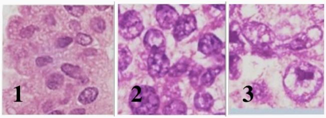
- Try to use matrix operations as much as possible!
- Slice indexing instead of loops. Broadcasting instead of entrywise multiplication.
- Review `demo2_more_numpy.ipynb`.
- This will make a huge difference in labs, and in your projects. Knowing how to work with matrices in an efficient way is one of the most important coding skills for machine learning and data science.

CLASSIFICIATION

Breast Cancer Biopsy: Determine if a breast lump in a patient is malignant (cancerous) or benign (safe).

- Collect cells from lump using fine needle biopsy.
- Stain and examine cells under microscope.
- Based on certain characteristics (shape, size, cohesion) determine if likely malignant or not).





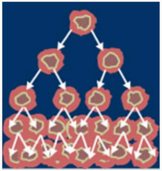
Demo: `demo_breast_cancer.ipynb`

Data: UCI machine learning repository

Breast Cancer Wisconsin (Original) Data Set

Download: Data Folder, Data Set Description

Abstract: Original Wisconsin Breast Cancer Database



Data Set Characteristics:	Multivariate	**Number of Instances:**	699	**Area:**	Life
Attribute Characteristics:	Integer	**Number of Attributes:**	10	**Date Donated**	1992-07-15
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	564320

`https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)`

Features: 10 numerical scores about cell characteristics (Clump Thickness, Uniformity, Marginal Adhesion, etc.)

Data: $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$.

$\vec{x}_i = [1, 5, 4 \ldots, 2]$ contains score values.

Label $y_i \in \{0, 1\}$ is 0 if benign cells, 1 if malignant cells.

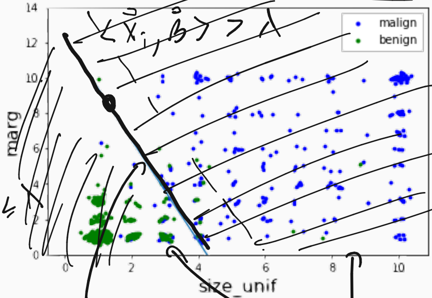
Goal: Based on scores (which would be collected manually, or even learned on their own using an ML algorithm) predict if a sample of cells is malignant or benign.

LINEAR CLASSIFIER

Given vector of predictors $\vec{x}_i \in \mathbb{R}^d$ (here $d = 2$) find a parameter vector $\vec{\beta} \in \mathbb{R}^d$ and threshold λ.

- Predict $y_i = 0$ if $\langle \vec{x}_i, \vec{\beta} \rangle \leq \lambda$.
- Predict $y_i = 1$ if $\langle \vec{x}_i, \vec{\beta} \rangle > \lambda$

Line has equation $\langle \vec{x}, \vec{\beta} \rangle = \lambda$



Hyperplane **Half-space**

$0 - 1$ LOSS

Question: How do we find a good linear classifier automatically?

$x_1, x_2, 1$ $\quad \beta_1, \beta_2, \beta_3$ $\quad \beta_3 = -\lambda$

Loss minimization approach (first attempt):

$x_1\beta_1 + x_2\beta_2 = \lambda$

$x_1\beta_1 + x_2\beta_2 + x_3\beta_3 = 0$

- **Model**[1]:

$$f_{\vec{\beta}}(\vec{x}) = \mathbb{1}\left[\langle \vec{x}, \vec{\beta} \rangle > 0\right]$$

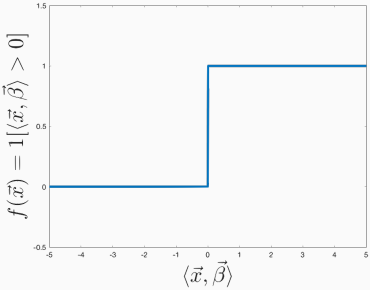
- **Loss function**: "$0 - 1$ Loss" = # of errors

$$L(\vec{\beta}) = \sum_{i=1}^{n} |f_{\vec{\beta}}(\vec{x}_i) - y_i|$$

[1] $\mathbb{1}$[event] is the indicator function: it evaluates to 1 if the argument inside is true, 0 if false.

$0-1$ loss

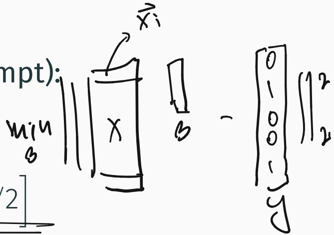
Problem with $0-1$ loss:



- The loss function $L(\vec{\beta})$ is not differentiable because $f_{\vec{\beta}}(\vec{x})$ is discontinuous.
- Impossible to take the gradient, very hard to minimize loss to find optimal $\vec{\beta}$.

Question: How do we find a good linear classifier automatically?

Loss minimization approach (second attempt):



- **Model**:

$$f_{\vec{\beta}}(\vec{x}) = \mathbb{1}\left[\langle \vec{x}, \vec{\beta} \rangle > 1/2\right]$$

- **Loss function**: "Square Loss"

$\beta^* = \arg\min L(\beta)$

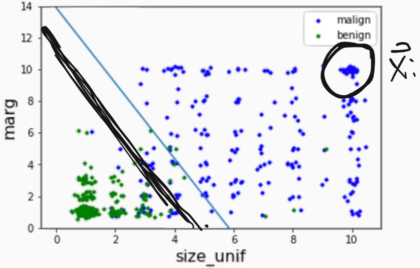
$$L(\vec{\beta}) = \sum_{i=1}^{n} (\langle \vec{x}_i, \vec{\beta} \rangle - y_i)^2$$

$= \|X\beta - y\|_2^2$

Intuitively tries to make $\langle \vec{x}, \vec{\beta} \rangle$ close to 0 for examples in class 0, close too 1 for examples in class 1.

LINEAR CLASSIFIER VIA SQUARE LOSS

We can solve for $\vec{\beta}$ my just solving a least squares multiple linear regression problem.



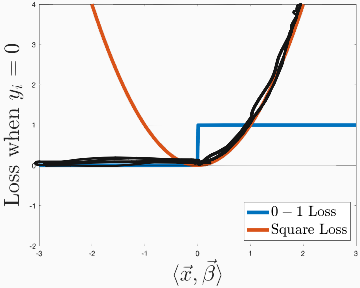
Do you see any issues here?

$\langle \vec{x}_i, \beta \rangle = 5$

$(\langle \vec{x}_i, \beta \rangle - 1)^2$

$y^2 = 16$

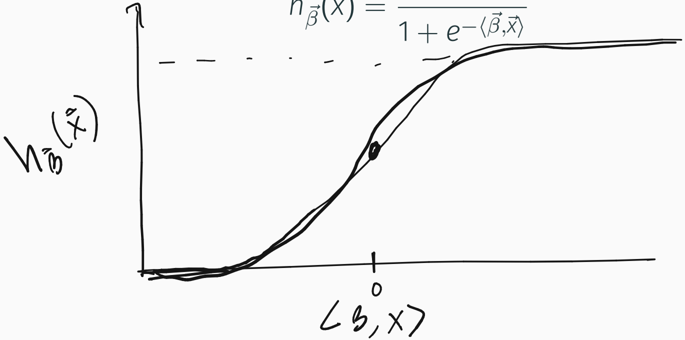
Problem with square loss:



- Loss increases if $\langle \vec{x}, \vec{\beta} \rangle < 0$ even if correct label is 0. Or if $\langle \vec{x}, \vec{\beta} \rangle > 1$ even if correct label is 1. Or
- Intuitively we don't want to "punish" these cases.

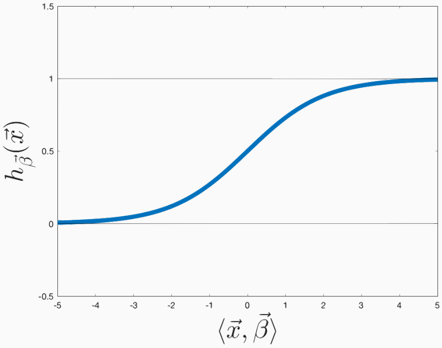
Let $h_{\vec{\beta}}(\vec{x})$ be the logistic function:

$$h_{\vec{\beta}}(\vec{x}) = \frac{1}{1 + e^{-\langle \vec{\beta}, \vec{x} \rangle}}$$



Let $h_{\vec{\beta}}(\vec{x})$ be the logistic function:

$$h_{\vec{\beta}}(\vec{x}) = \frac{1}{1 + e^{-\langle \vec{\beta}, \vec{x} \rangle}}$$



p of y_i being 1

We can think of $h_{\vec{\beta}}(\vec{x})$ as mapping $\langle \vec{\beta}, \vec{x} \rangle$ to a probability.

Loss minimization approach (what works!):

- **Model**: Let $h_{\vec{\beta}}(\vec{x}) = \frac{1}{1+e^{-\langle \vec{\beta}, \vec{x} \rangle}}$

$$\frac{1}{1+e^{-\langle \vec{x}, \vec{\beta} \rangle}} > 1/2 \qquad \langle \vec{x}, \vec{\beta} \rangle > 0$$

$$f_{\vec{\beta}}(\vec{x}) = \mathbb{1}\left[h_{\vec{\beta}}(\vec{x}) > 1/2\right] = \mathbb{1}\left[\langle \vec{x}, \vec{\beta} \rangle > 0\right]$$

- **Loss function**: "Logistic loss" aka "Cross-entropy loss"

$$L(\vec{\beta}) = -\sum_{i=1}^{n} y_i \log(h_{\vec{\beta}}(\vec{x})) + (1 - y_i) \log(1 - h_{\vec{\beta}}(\vec{x}))$$

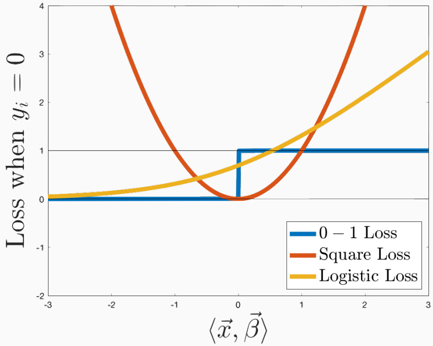
$y_i = 0 \longrightarrow -\log(1 - h_{\vec{\beta}}(\vec{x}))$

$y_i = 1$

$\vec{\beta}^* = \arg\min L(\vec{\beta})$

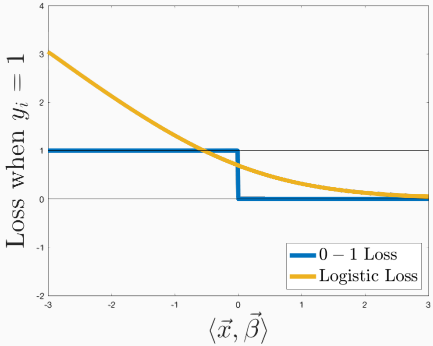
Logistic Loss:

$L(\vec{\beta}) = -\sum_{i=1}^{n} y_i \log(h_{\vec{\beta}}(\vec{x})) + (1 - y_i)\log(1 - h_{\vec{\beta}}(\vec{x}))$

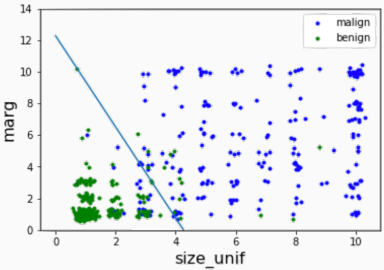


Logistic Loss:

$L(\vec{\beta}) = -\sum_{i=1}^{n} y_i \log(h_{\vec{\beta}}(\vec{x})) + (1 - y_i)\log(1 - h_{\vec{\beta}}(\vec{x}))$

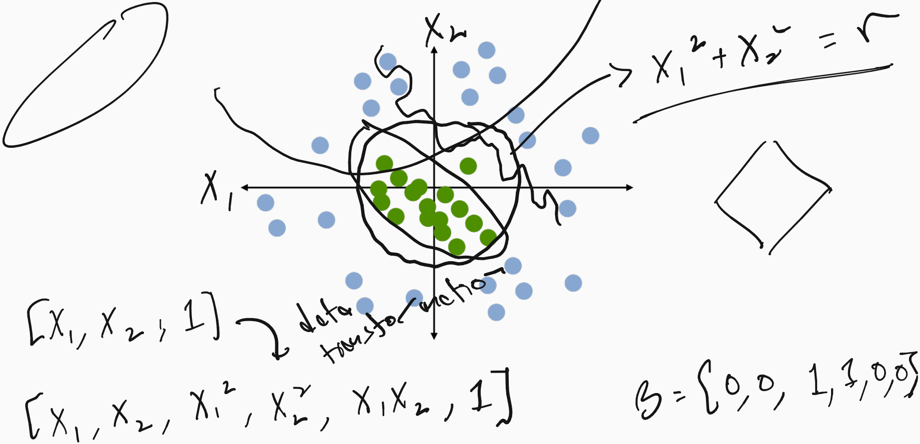


- Convex function, can be minimized using gradient descent (next lecture).
- Works well in practice.
- Good Bayesian motivation: see posted lecture notes if you are interested.

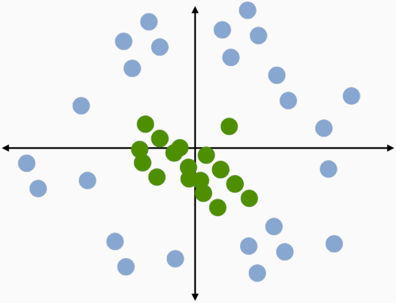


Fit using logistic regression/log loss.

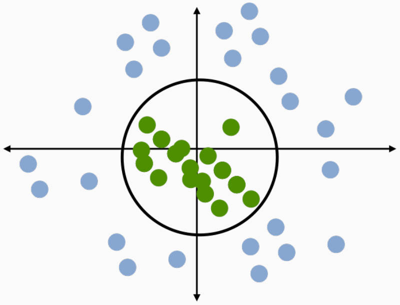
How would we learn a classifier for this data?



How would we learn a classifier for this data using logisitic regression?



Transform each $\vec{x} = [x_1, x_2]$ to $\vec{x} = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]$



- Predict class 1 if $x_1^2 + x_2^2 < \lambda$.
- Predict class 0 if $x_1^2 + x_2^2 \geq \lambda$.

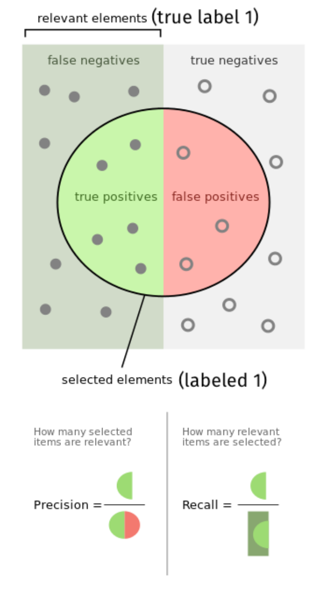
This is a linear classifier on our transformed data set. Logisitic regression would learn $\vec{\beta} = [0, 0, 0, 1, 1, 0]$.

Once we have a classification algorithm, how do we judge its performance?

- **Simplest answer:** Error rate = fraction of data examples misclassified in test set.
- What are some issues with this approach?

- **Precision:** Fraction of positively labeled examples (label 1) which are correct.
- **Recall:** Fraction of true positives that we labeled correctly with label 1.

Question: Which should we optimize for medical diagnosis?



Possible logistic regression workflow:

- Learn $\vec{\beta}$ and compute $h_{\vec{\beta}}(\vec{x}_i) = \frac{1}{1+e^{-\langle \vec{x}_i, \vec{\beta} \rangle}}$ for all $\vec{x}_i$.
- Predict $y_i = 0$ if $h_{\vec{\beta}}(\vec{x}_i) \leq \lambda$, $y_i = 1$ if $h_{\vec{\beta}}(\vec{x}_i) > \lambda$.
- Default value of λ is 1/2. Increasing λ improves precision. Decreasing λ improves recall.

This is very heuristic. There are other methods for handling "class imbalance" which can often lead to good overall error, but poor precision or recall. Techniques include weighting the loss function to care more about false negatives, or subsampling the larger class.

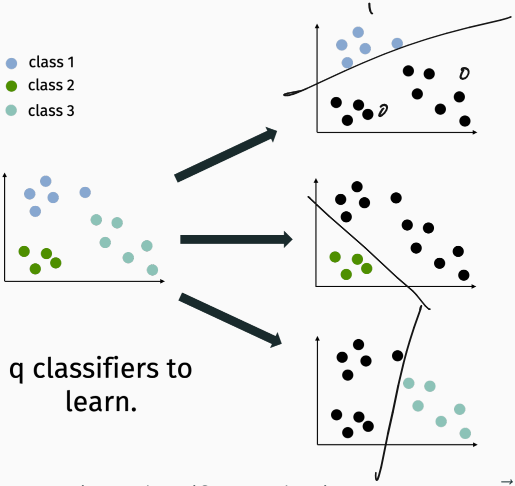
What about when $y \in \{1, \ldots, q\}$ instead of $y \in \{0, 1\}$

Two options for multiclass data:

- One-vs.-all (most common, also called one-vs.-rest)
- One-vs.-one (slower, but can be more effective)

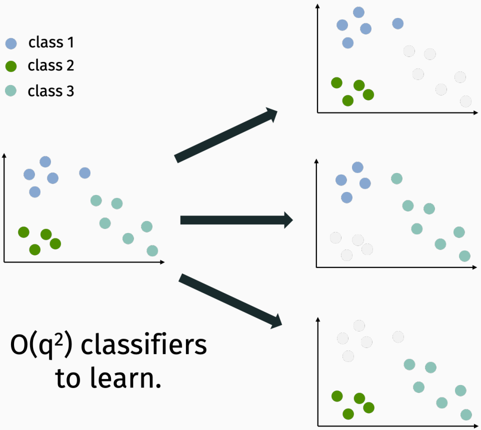
In both cases, we convert to multiple binary classification problem.

ONE VS. REST



- For q classes train q classifiers. Obtain parameters $\vec{\beta}_1, \ldots, \vec{\beta}_q$.
- Assign y to class i with maximum $\langle \vec{\beta}_i, \vec{x} \rangle$.

ONE VS. REST



- For q classes train $\frac{q(q-1)}{2}$ classifiers.
- Assign y to class which i which wins in the most number of head-to-head comparisons.

Hard case for one-vs.-all.

- One-vs.-one would be a better choice here.
- Also tends to work better when there is class in balance.

Confusion matrix for k classes:

Pred--> Real↓	**1**	**2**	**…**	**K**
1				
2				
…				
K				

- Entry i, j is the fraction of class i items classified as class j.
- Overall accuracy is the average of the diagonals.
- Useful to see whole matrix to visualize where errors occur.