CS-GY 6923: Lecture 5
Linear Classification, Logistic Regression, Gradient Descent

NYU Tandon School of Engineering, Prof. Christopher Musco

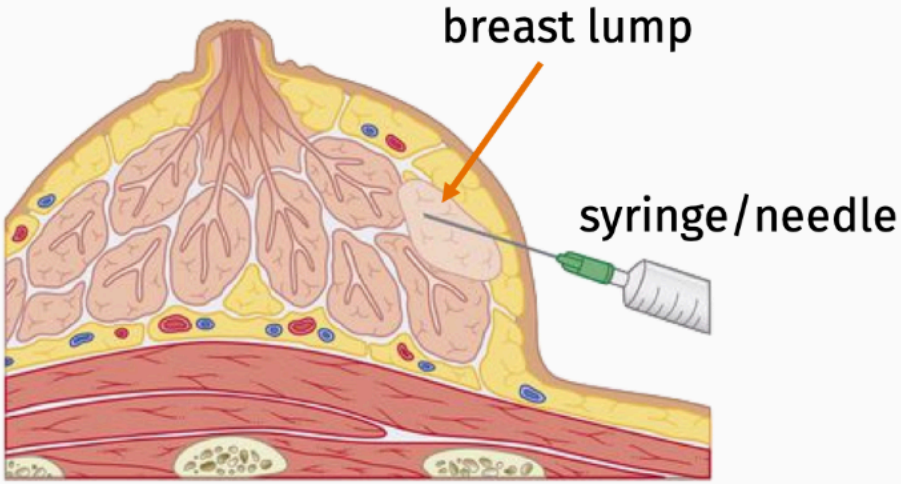
COURSE ADMIN

- We will release solutions to HW1 shortly and I will go over solutions next week in office hours.
- Lab 3 due next Tuesday.
- Written Homework 2 due the day after that. No slip days for this one.
- Midterm exam on Friday 10/18. We will do an optional lecture on a miscellaneous topic after the midterm.

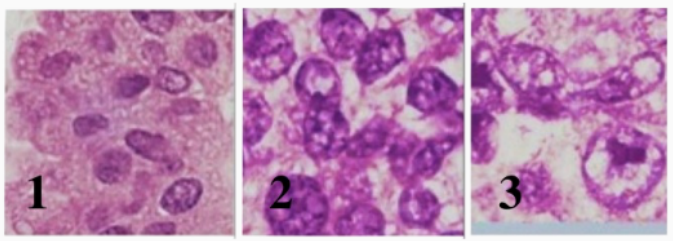
MOTIVATING PROBLEM

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).



cross section



MOTIVATING PROBLEM

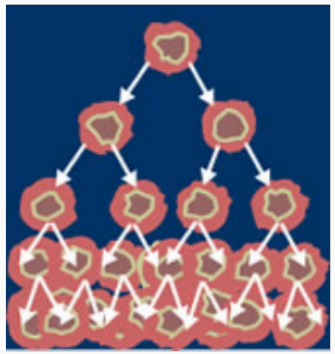
Demo: `demo_breast_cancer.ipynb`

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.)

MOTIVATING PROBLEM

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

$\mathbf{x}_i = [1, 5, 4 \ldots, 2]$ 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.

Approach:

- Naive Bayes Classifier can be extended to $\mathbf{x}$ with numerical values (instead of binary values as seen before). Will see on Homework 2.

What are other classification algorithms people have heard of?

k-NEAREST NEIGHBOR METHOD

k-NN algorithm: a simple but powerful baseline for classification.

Training data: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where $y_1, \ldots, y_n \in \{1, \ldots, q\}$.

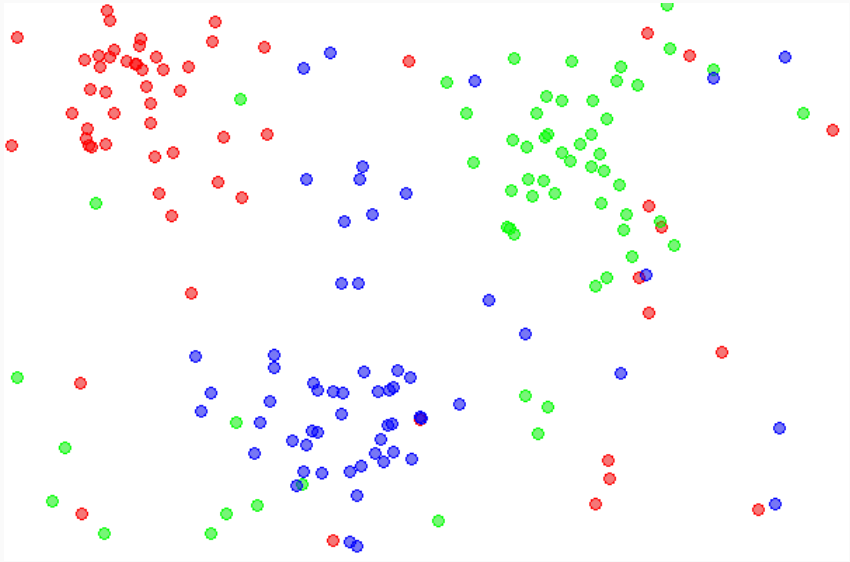
Classification algorithm:

Given new input $\mathbf{x}_{new}$,

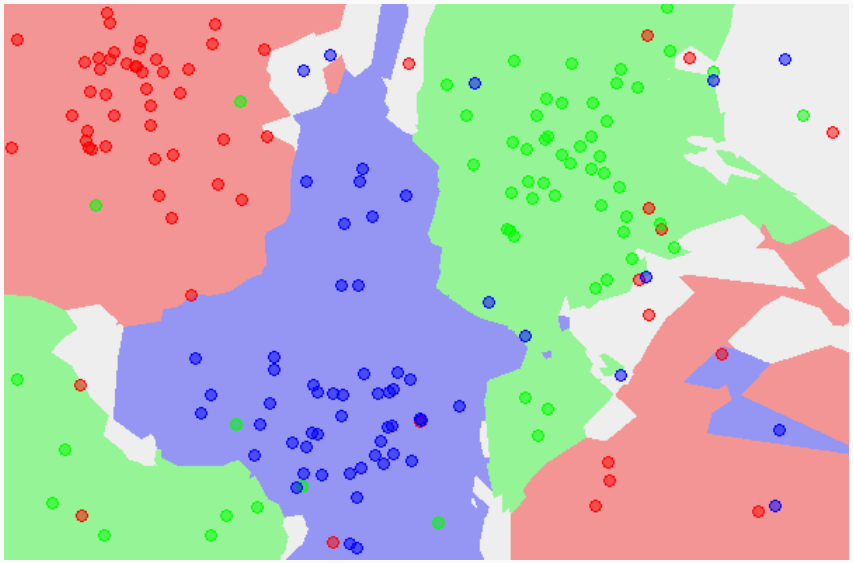
- Compute $sim(\mathbf{x}_{new}, \mathbf{x}_1), \ldots, sim(\mathbf{x}_{new}, \mathbf{x}_n)$.[1]
- Let $\mathbf{x}_{j_1}, \ldots, \mathbf{x}_{j_k}$ be the training data vectors with highest similarity to $\mathbf{x}_{new}$.
- Predict y_{new} as $majority(y_{j_1}, \ldots, y_{j_k})$. Break ties any way you want.

[1] $sim(\mathbf{x}_{new}, \mathbf{x}_i)$ is any chosen similarity function, like $\langle \mathbf{x}_{new}, \mathbf{x}_i \rangle$ or $-\|\mathbf{x}_{new} - \mathbf{x}_i\|_2$.

k-NEAREST NEIGHBOR METHOD



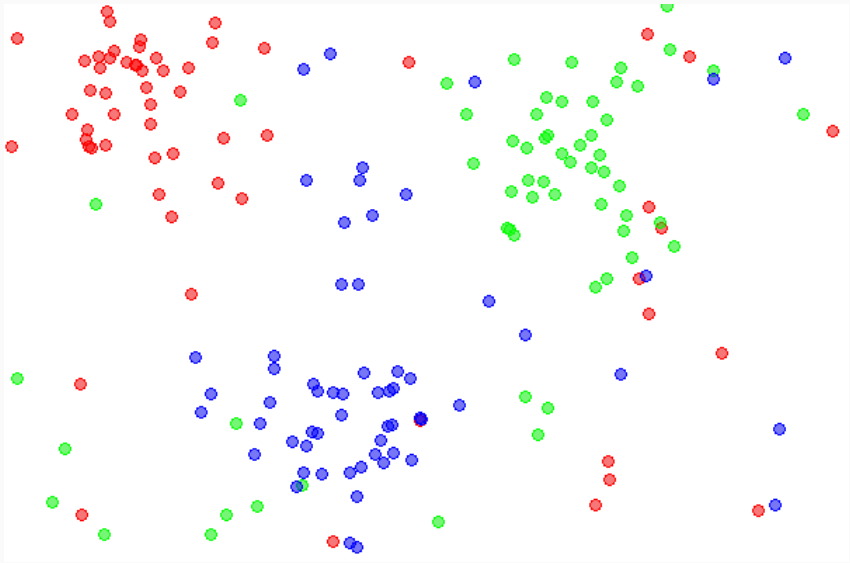
Data



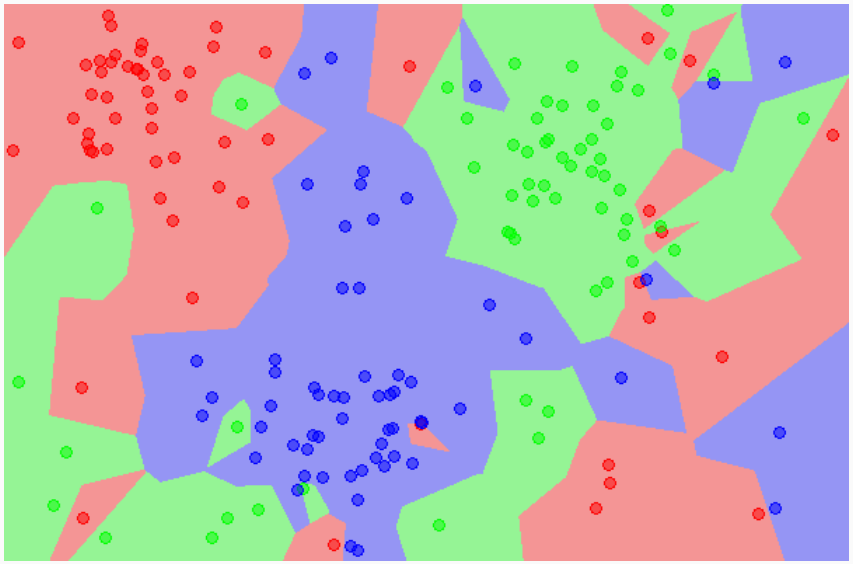
5-NN classifier

- Smaller k, more complex classification function.
- Larger k, more robust to noisy labels/class overlap.

k-NEAREST NEIGHBOR METHOD



Data

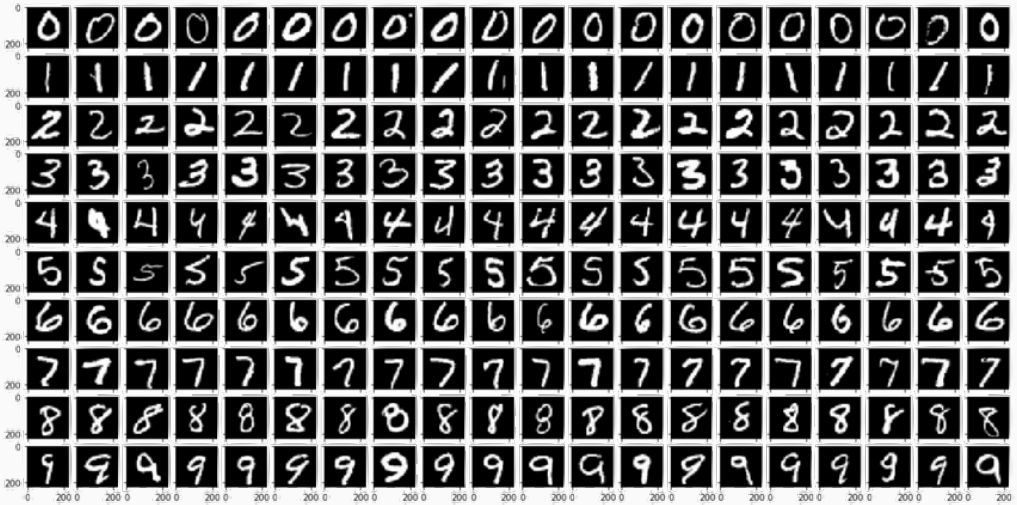


1-NN classifier

- Smaller k, more complex classification function.
- Larger k, more robust to noisy labels/class overlap.

MNIST IMAGE DATA

Especially good for large datasets with lots of repetition. Works well on MNIST for example:



≈ 95% Accuracy out-of-the-box.

Can be improved to 99.5% with a fancy similarity function![2]

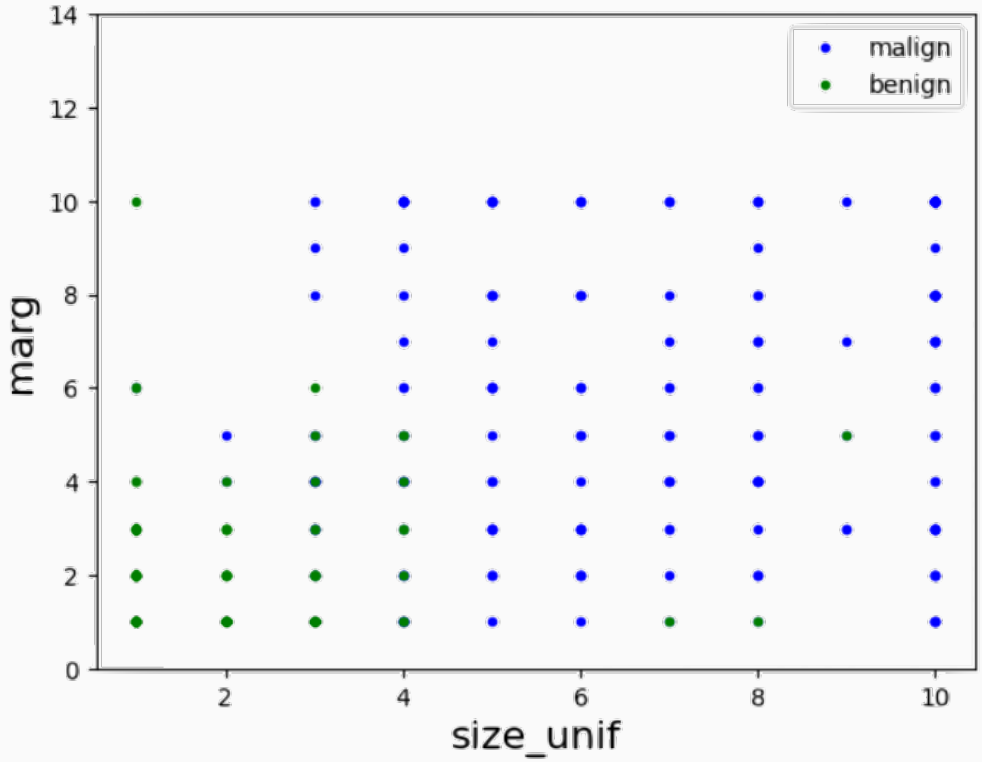
One issue is that prediction can be computationally intensive...

[2] We will revisit this when we talk about kernel methods.

LINEAR CLASSIFICATION

BEGIN BY PLOTTING DATA

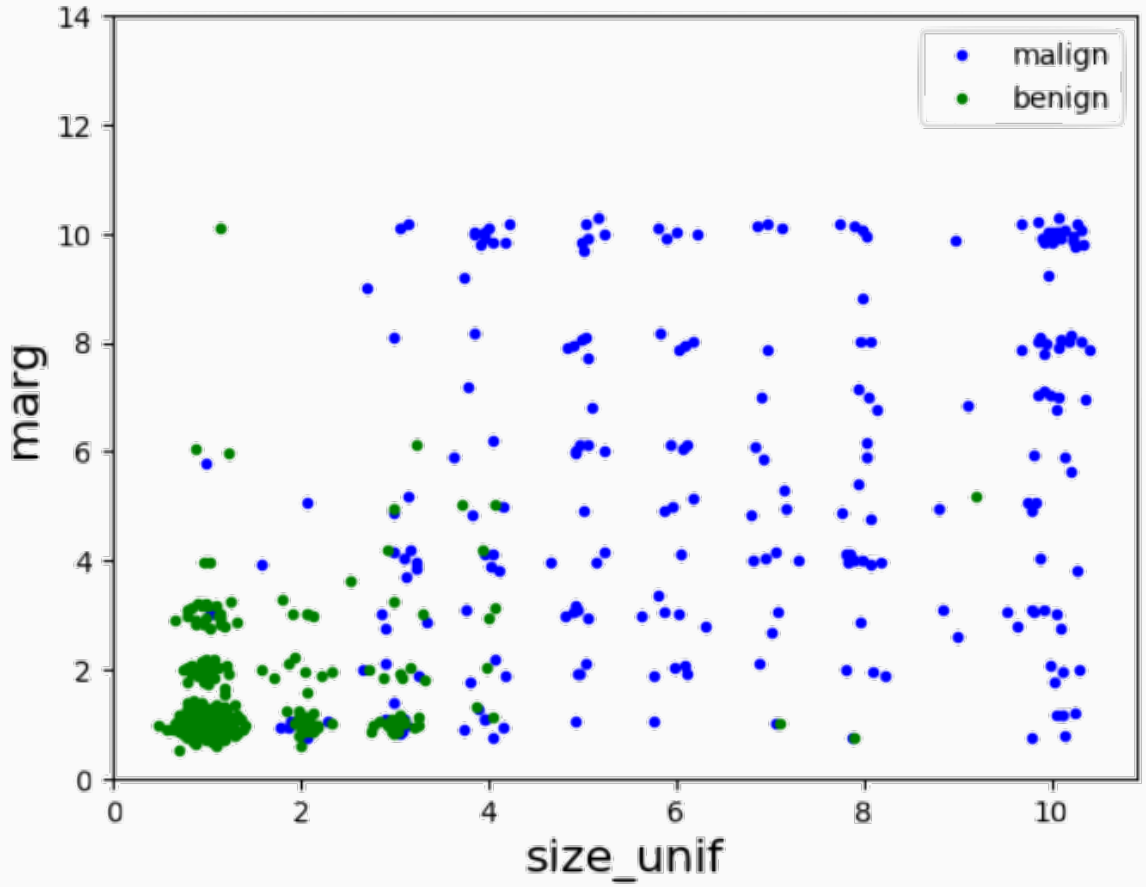
We pick two variables, Marginal Adhesion and Size Uniformity and plot a scatter plot. Points with label 1 (malignant) are plotted in blue, those with label 2 (benign) are plotted in green.



Lots of overlapping points! Hard to get a sense of the data.

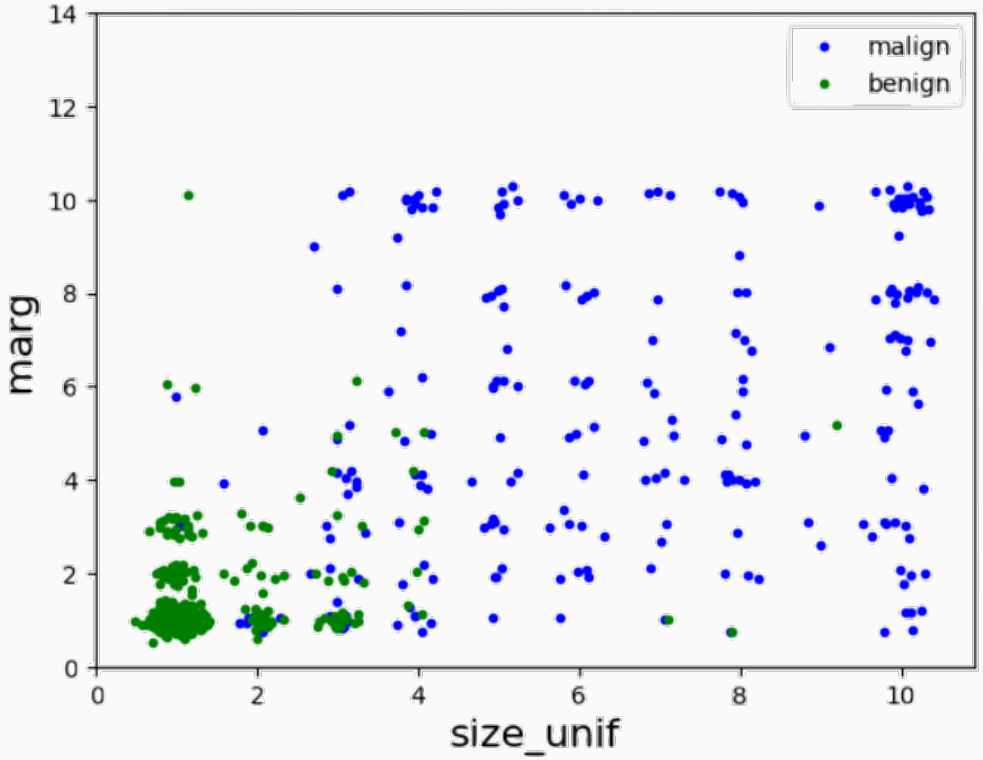
PLOTTING WITH JITTER

Simple + Useful Trick: data jittering. Add tiny random noise (using e.g. `np.random.randn`) to data to prevent overlap.



Noise is only for plotting. It is not added to the data for training, testing, etc.

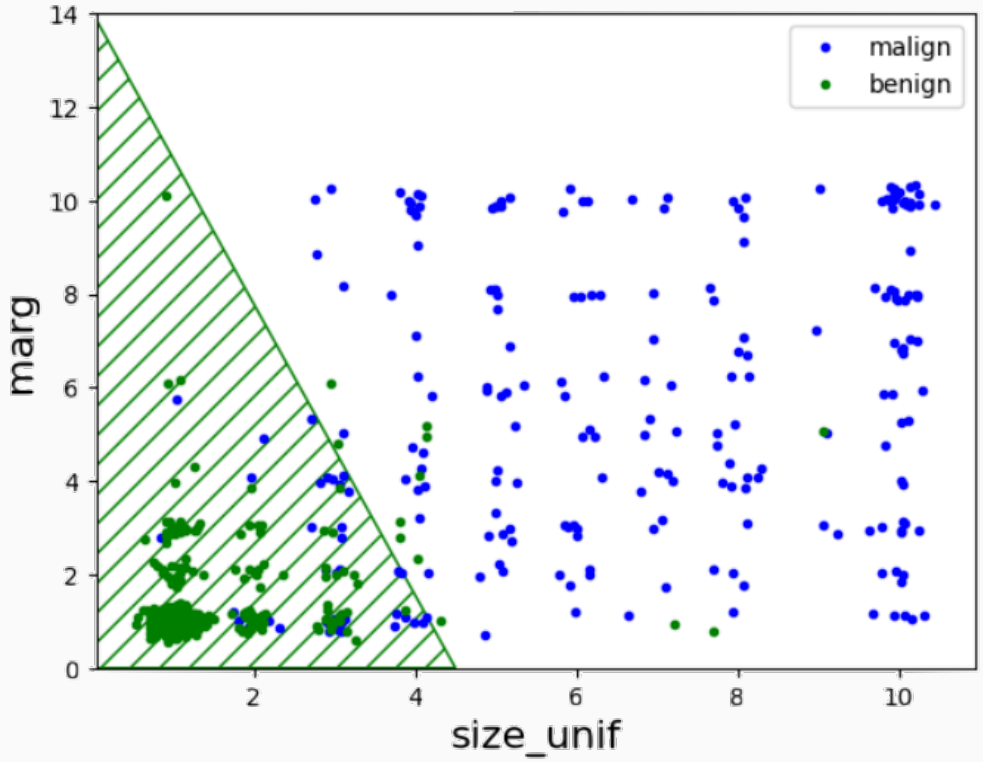
Any ideas for possible classification rules for this data?



LINEAR CLASSIFIER

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

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

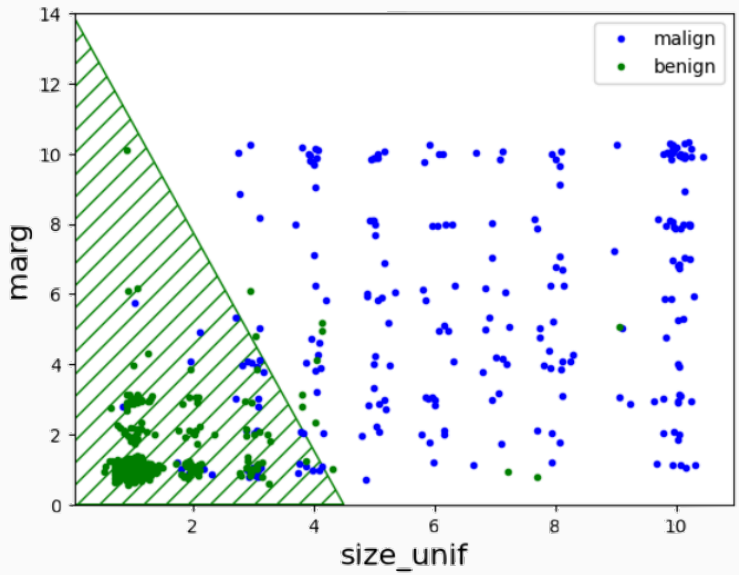


Line has equation $\langle \mathbf{x}, \boldsymbol{\beta} \rangle = \lambda$.

LINEAR CLASSIFIER

As long as we append a 1 onto each data vector $\mathbf{x}_i$ (i.e. a column of ones onto the data matrix $\mathbf{X}$) like we did for linear regression, an equivalent function is:

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



Line has equation $\langle \mathbf{x}, \boldsymbol{\beta} \rangle = 0$.

$0-1$ loss

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

Loss minimization approach (first attempt):

- **Model**[3]:

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

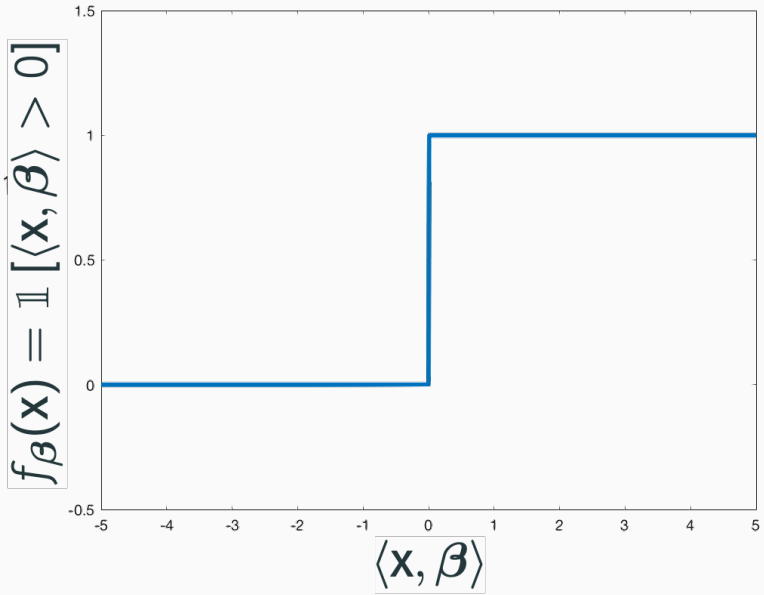
- **Loss function**: "$0-1$ Loss"

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

[3] $\mathbb{1}$[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 $L(\beta)$ is not differentiable because $f_\beta(\mathbf{x})$ is discontinuous.
- Impossible to take the gradient, very hard to minimize loss to find optimal β.
- Non-convex function (will make more sense next lecture).

LINEAR CLASSIFIER VIA SQUARE LOSS

Loss minimization approach (second attempt):

- **Model**:

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

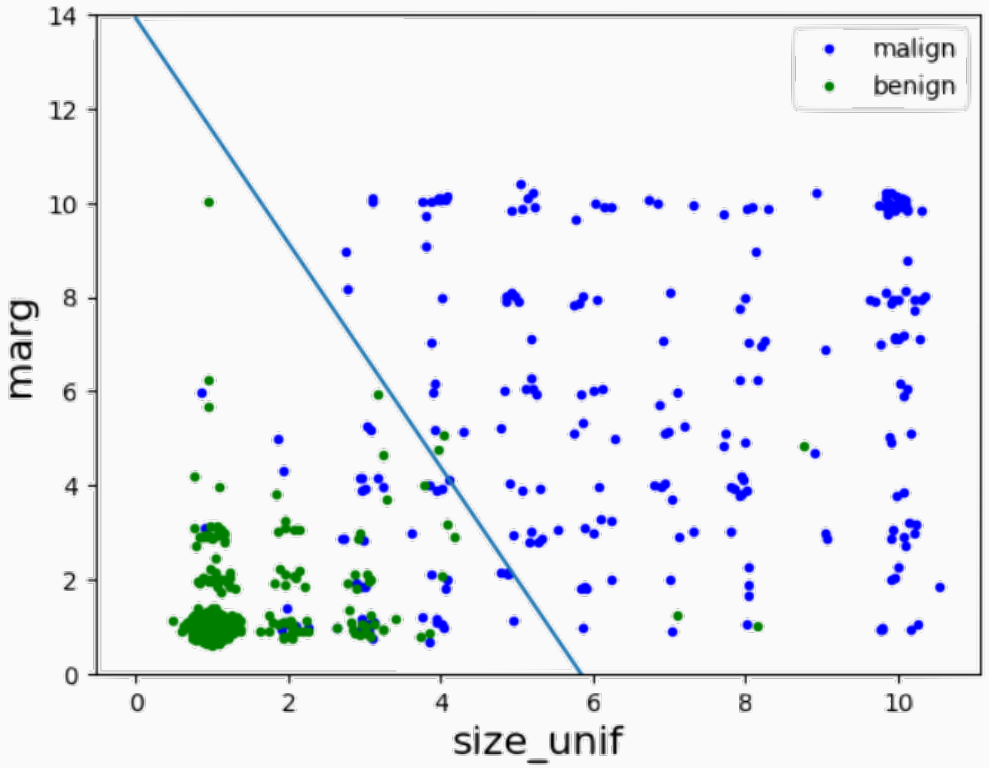
- **Loss function**: "Square Loss"

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

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

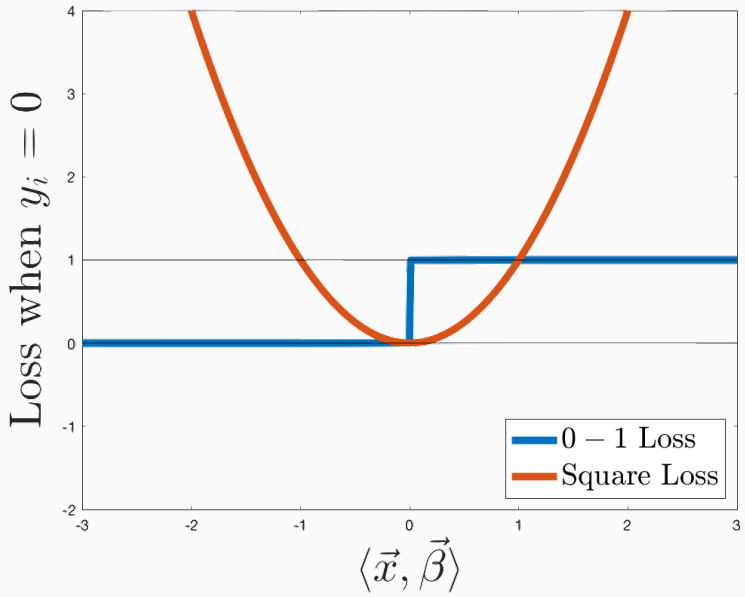
LINEAR CLASSIFIER VIA SQUARE LOSS

We can solve for β by just solving a least squares multiple linear regression problem.



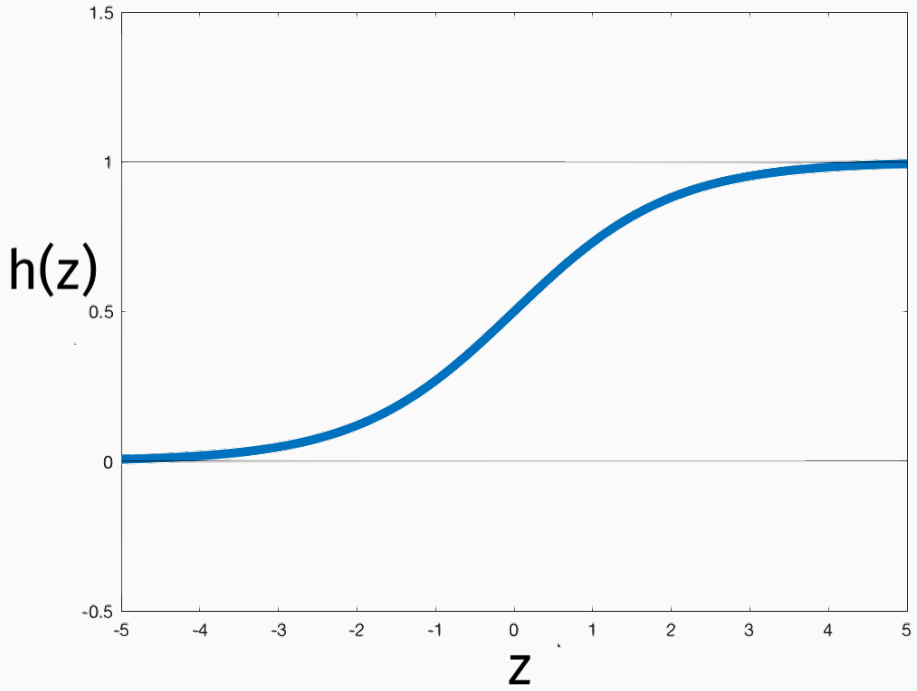
Do you see any issues here?

Problem with square loss:



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

Let $h(z)$ be the logistic/sigmoid function: $h(z) = \frac{1}{1+e^{-z}}$



As discussed in previous lecture, can think of this function as mapping $\mathbf{x}^T\boldsymbol{\beta}$ to a probability that the true label is 1. If $\mathbf{x}^T\boldsymbol{\beta} \gg 0$ then the probability is close to 1, if $\mathbf{x}^T\boldsymbol{\beta} \ll 0$ then the probability is close to 0.

Loss minimization approach (this works!):

- **Model**: $h(\langle \boldsymbol{\beta}, \mathbf{x} \rangle) = \frac{1}{1+e^{-\langle \boldsymbol{\beta}, \mathbf{x} \rangle}}$.

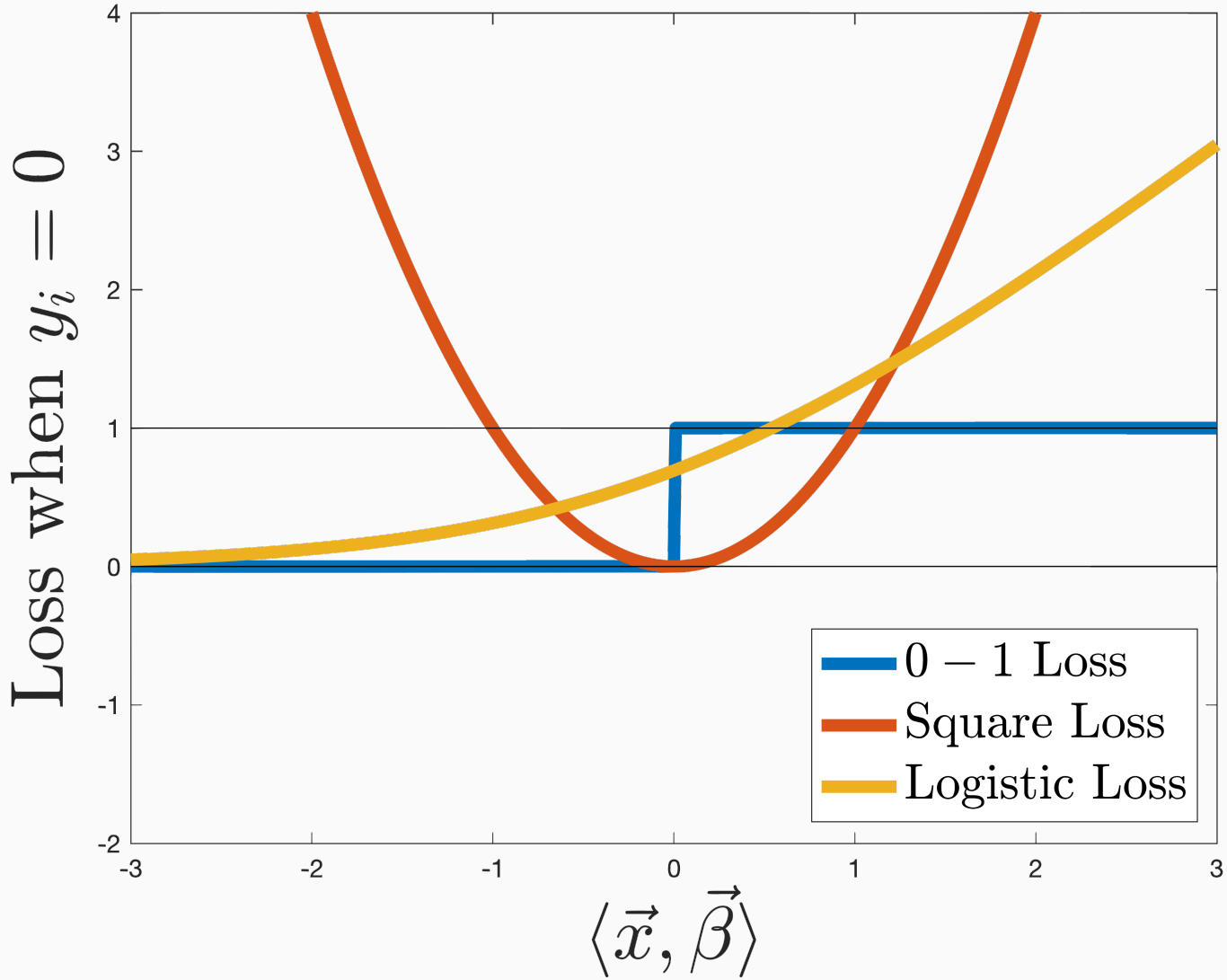
$$\begin{aligned} f_{\boldsymbol{\beta}}(\mathbf{x}) &= \mathbb{1}\left[h(\langle \boldsymbol{\beta}, \mathbf{x} \rangle) > 1/2\right] \\ &= \mathbb{1}\left[\langle \mathbf{x}, \boldsymbol{\beta} \rangle > 0\right] \end{aligned}$$

- **Loss function**: "Logistic loss" aka "binary cross-entropy loss"

$$L(\boldsymbol{\beta}) = -\sum_{i=1}^{n} y_i \log(h(\langle \boldsymbol{\beta}, \mathbf{x} \rangle)) + (1 - y_i) \log(1 - h(\langle \boldsymbol{\beta}, \mathbf{x} \rangle))$$

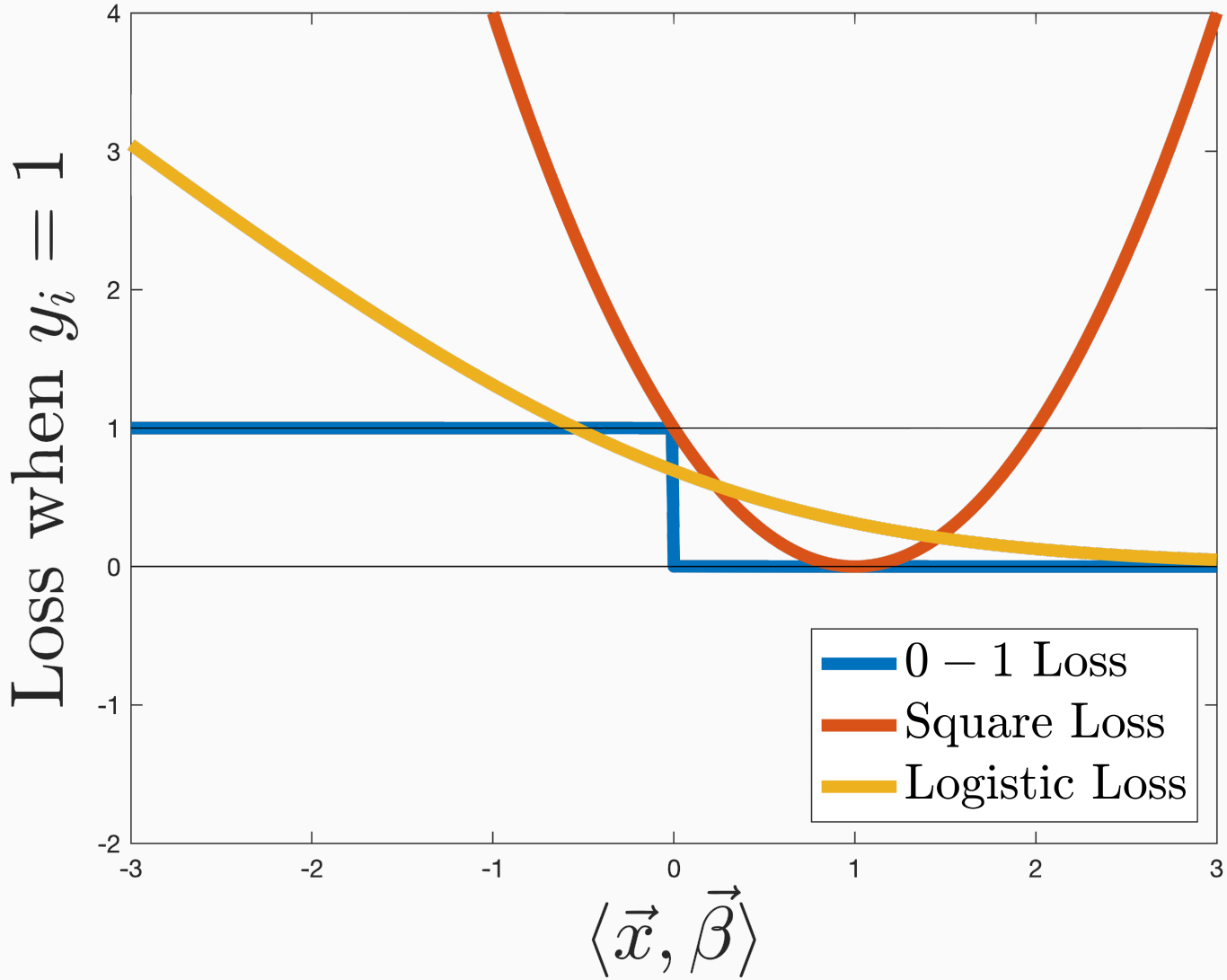
Logistic Loss:

$L(\beta) = -\sum_{i=1}^{n} y_i \log(h(\langle \beta, x \rangle)) + (1 - y_i) \log(1 - h(\langle \beta, x \rangle))$



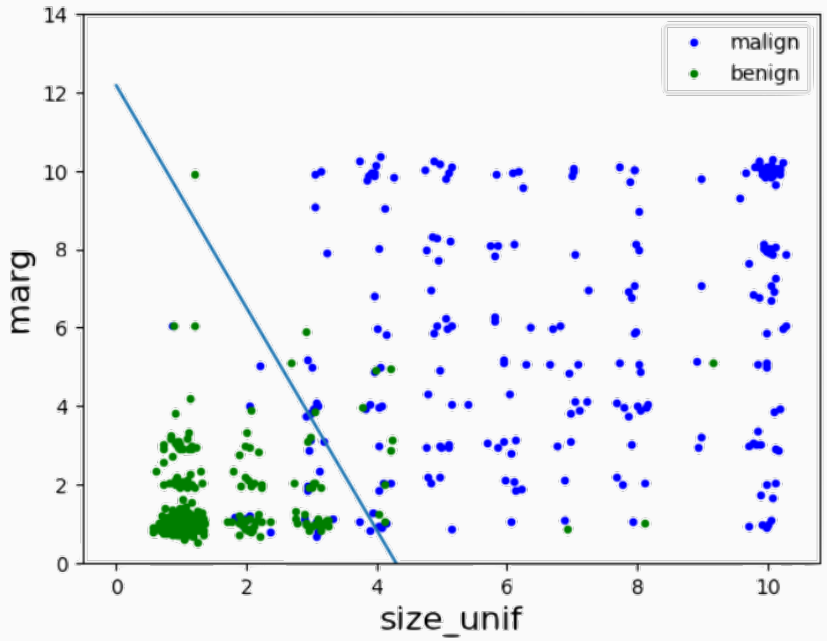
Logistic Loss:

$$L(\boldsymbol{\beta}) = -\sum_{i=1}^{n} y_i \log(h(\langle \boldsymbol{\beta}, \mathbf{x} \rangle)) + (1 - y_i) \log(1 - h(\langle \boldsymbol{\beta}, \mathbf{x} \rangle))$$



LOGISTIC LOSS

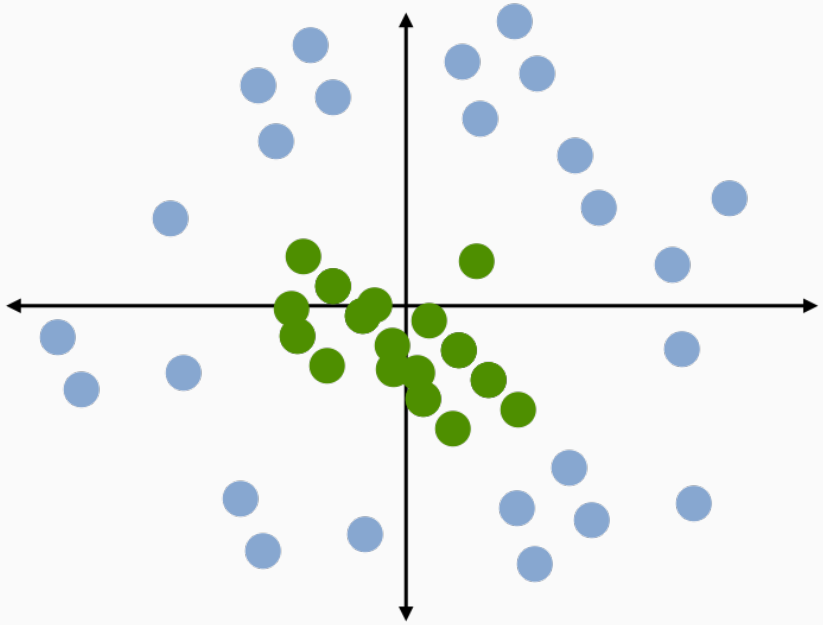
- Convex function in β, can be minimized using gradient descent.
- Works well in practice.
- Good Bayesian motivation (discussed last class).



Fit using logistic regression/log loss.

NON-LINEAR TRANSFORMATIONS

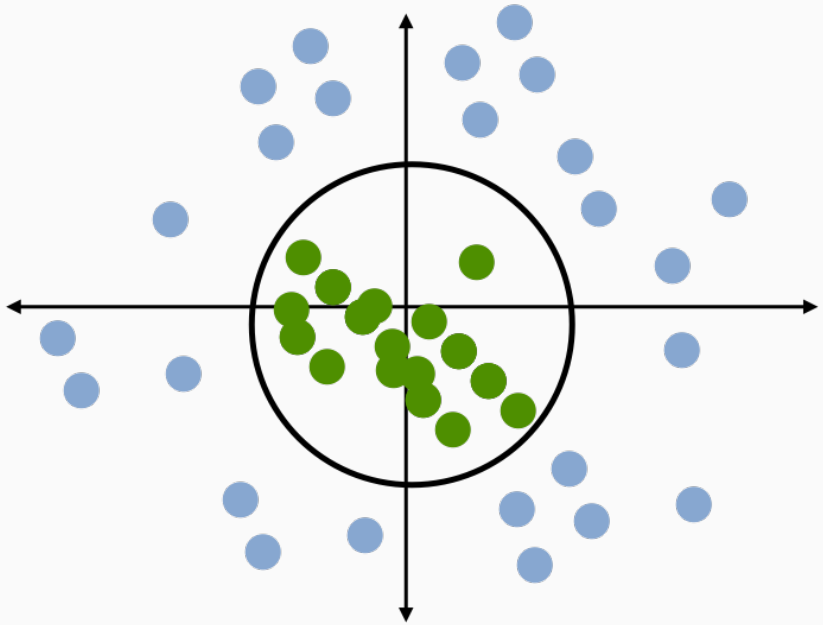
How would we learn a classifier for this data using logisitic regression?



This data is not linearly separable or even approximately linearly separable.

NON-LINEAR TRANSFORMATIONS

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

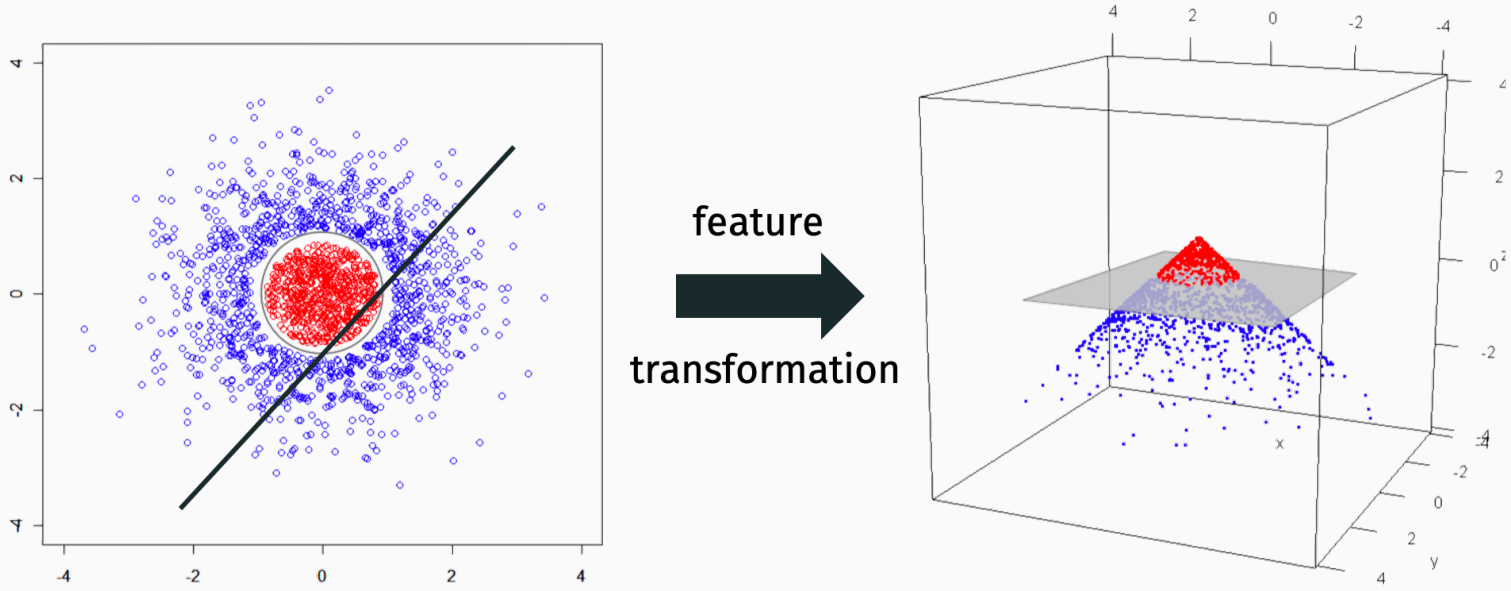


- Predict class 1 if $x_1^2 + x_2^2 \geq \lambda$.
- Predict class 0 if $x_1^2 + x_2^2 < \lambda$.

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

NON-LINEAR TRANSFORMATIONS

View as mapping data to a higher dimensional space, where it is linearly separable.



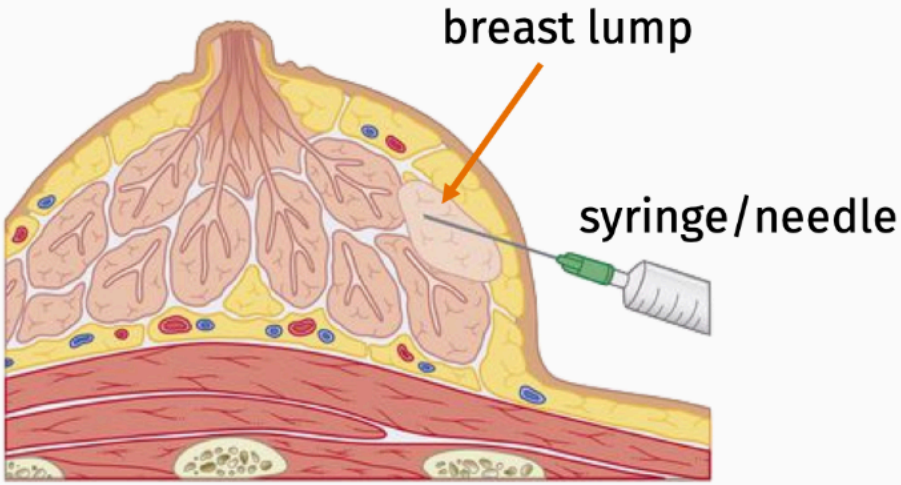
Lots more on this in future lecture!

ERROR IN CLASSIFICATION

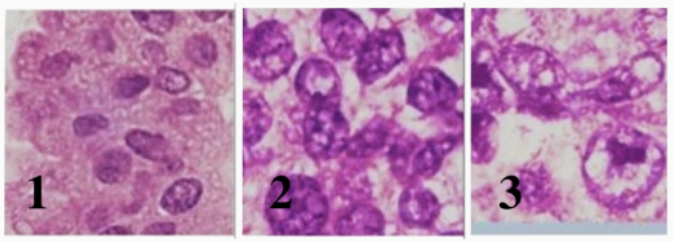
Once we have a classification algorithm, how do we judge its performance?

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

Think back to motivating problem of breast cancer detection.



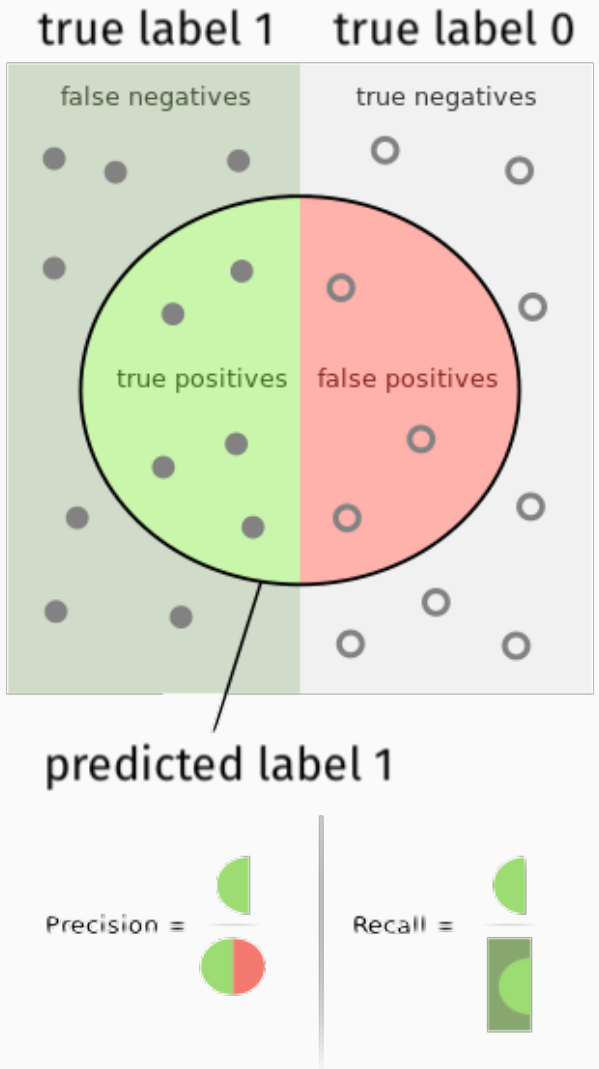
cross section



ERROR IN CLASSIFICATION

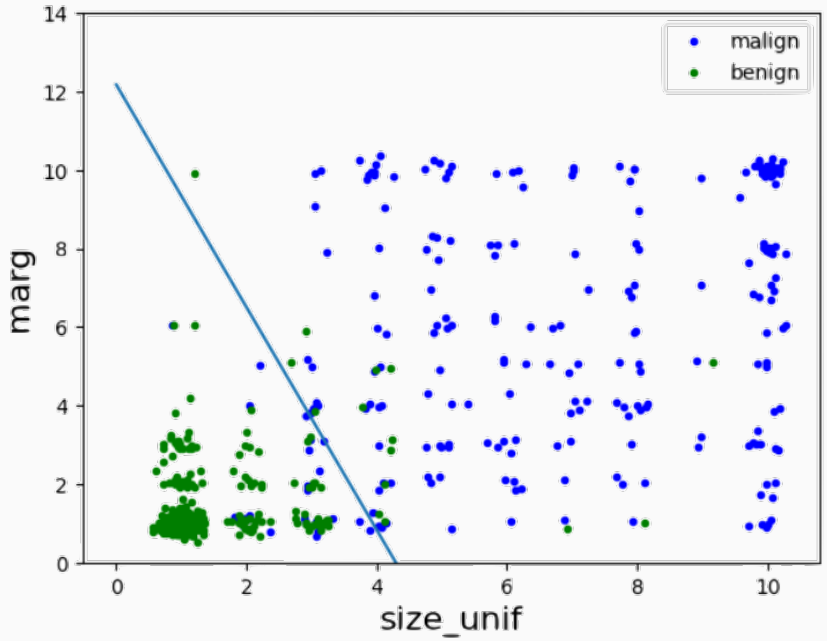
- **Precision:** Fraction of positively labeled examples (label 1) which are correct.
- **Recall:** Fraction of true positives that we labeled correctly with label 1.

Question: Which should we optimize for medical diagnosis? (Here "positive" label means the patient has the disease.)

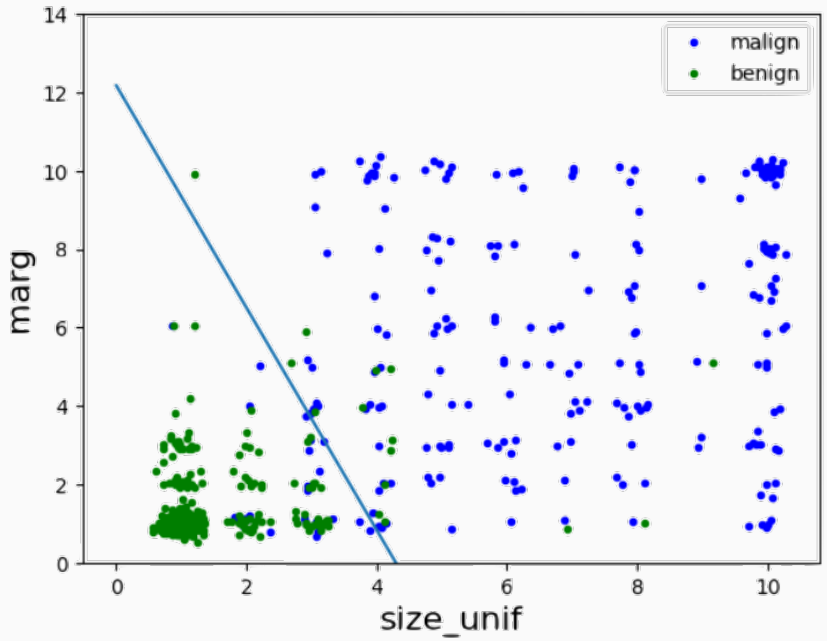


Possible logistic regression workflow:

- Learn β using logistic loss.
- Predict $y_i = 0$ if $\langle \beta, \mathbf{x} \rangle < \lambda$, $y_i = 1$ if $\langle \beta, \mathbf{x} \rangle \geq \lambda$ where $\lambda = 0$ to start.
- Increase λ to improve precision. Decrease λ to improve recall.



CLASS IMBALANCE



One very common cause of poor precision or recall is class imbalance. A common way of dealing with this is to subsample down the larger class.

This is actually what was done with the breast cancer dataset.

MULTI-CLASS

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

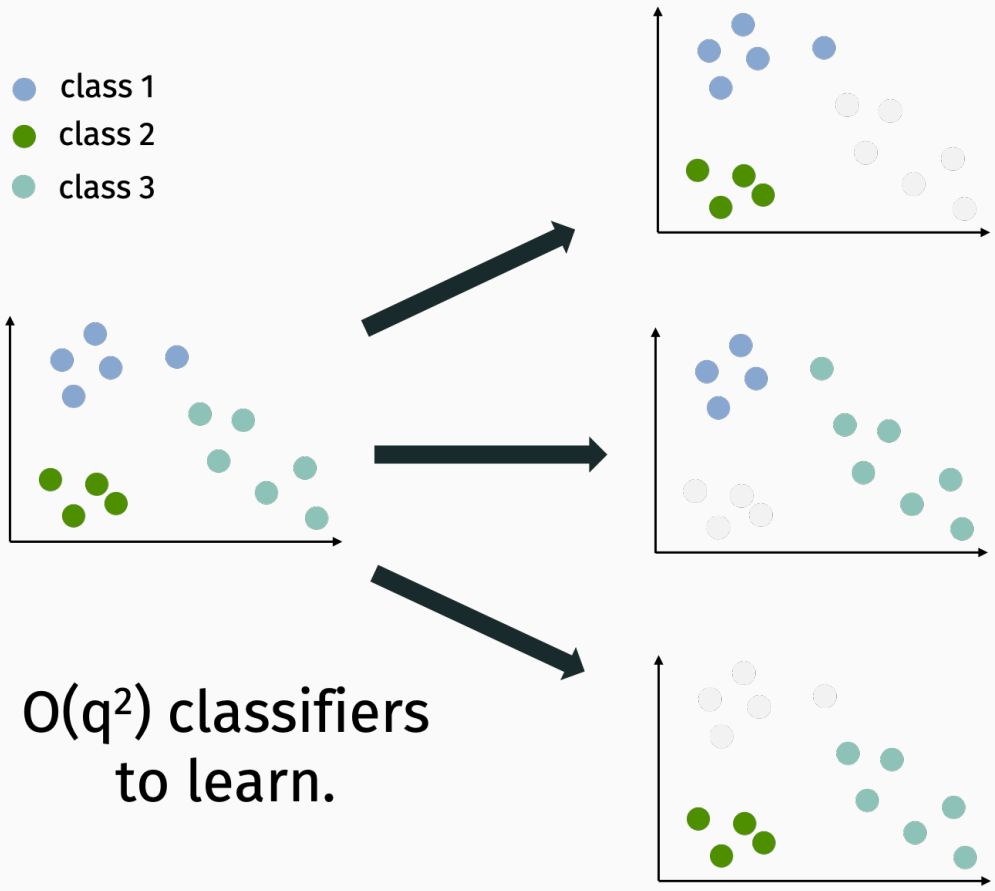
Two common options for reducing multi-class problems to binary problems:

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

ONE VS. REST

- For q classes train q classifiers. Obtain parameters $\beta^{(1)}, \ldots, \beta^{(q)}$.
- Assign y to class i if $\langle \beta^{(i)}, \mathbf{x} \rangle \geq 0$. Could be ambiguous!
- **Better:** Assign y to class i with maximum value of $\langle \beta^{(i)}, \mathbf{x} \rangle$.

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.

ONE VS. ONE

Hard case for one-vs.-all.

- One-vs.-one would be a better choice here.

But one-vs.-one can be super expensive! E.g when $q = 100$ or $q = 1000$.

More common modern alternative: If we have q classes, train a single model with q parameter vectors $\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(q)}$, and predict class $i = \arg\max_i \langle \boldsymbol{\beta}^{(i)}, \mathbf{x} \rangle$.

Same idea as one-vs.-rest, but we treat $[\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(q)}]$ as a single length qd parameter vector which we optimize to minimize a single joint loss function. We do not train the parameter vectors separately.

What's a good loss function?

Softmax function:

$$\begin{bmatrix} \langle \boldsymbol{\beta}^{(1)}, \mathbf{x} \rangle \\ \vdots \\ \langle \boldsymbol{\beta}^{(q)}, \mathbf{x} \rangle \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} e^{\langle \boldsymbol{\beta}^{(1)}, \mathbf{x} \rangle} / \sum_{i=1}^{q} e^{\langle \boldsymbol{\beta}^{(i)}, \mathbf{x} \rangle} \\ \vdots \\ e^{\langle \boldsymbol{\beta}^{(q)}, \mathbf{x} \rangle} / \sum_{i=1}^{q} e^{\langle \boldsymbol{\beta}^{(i)}, \mathbf{x} \rangle} \end{bmatrix}$$

Softmax takes in a vector of numbers and converts it to a vector of probabilities:

$$\begin{bmatrix} -10 & 4 & 1 & 0 & -5 \end{bmatrix} \to \begin{bmatrix} .00 & .93 & .04 & .02 & .01 \end{bmatrix}$$

Multi-class cross-entropy:

$$L(\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(q)}) = -\sum_{i:y_i=1} \log \frac{e^{\langle \boldsymbol{\beta}^{(1)}, \mathbf{x}_i \rangle}}{\sum_{j=1}^{q} e^{\langle \boldsymbol{\beta}^{(j)}, \mathbf{x}_i \rangle}} - \ldots - \sum_{i:y_i=q} \log \frac{e^{\langle \boldsymbol{\beta}^{(q}, \mathbf{x}_i \rangle}}{\sum_{j=1}^{q} e^{\langle \boldsymbol{\beta}^{(j)}, \mathbf{x}_i \rangle}}$$

$$= -\sum_{i=1}^{n} \sum_{\ell=1}^{q} \mathbb{1}[y_i = \ell] \cdot \log \frac{e^{\langle \boldsymbol{\beta}^{(\ell)}, \mathbf{x}_i \rangle}}{\sum_{j=1}^{q} e^{\langle \boldsymbol{\beta}^{(j)}, \mathbf{x}_i \rangle}}$$

Binary cross-entropy:

$$L(\boldsymbol{\beta}) = -\sum_{i=1}^{n} y_i \log(h(\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle)) + (1 - y_i) \log(1 - h(\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle))$$

$$= -\sum_{i:y_i=1} \log(h(\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle)) - \sum_{i:y_i=0} \log(1 - h(\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle))$$

Not exactly the same... but can show equivalent if you set $\boldsymbol{\beta}^{(0)} = \boldsymbol{\beta}$ and $\boldsymbol{\beta}^{(1)} = -\boldsymbol{\beta}$.

Multi-class cross-entropy:

$$L(\boldsymbol{\beta}) = -\sum_{i=1}^{n} y_i \log(h(\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle)) + (1 - y_i) \log(1 - h(\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle))$$

$$= -\sum_{i:y_i=1} \log(h(\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle)) - \sum_{i:y_i=0} \log(1 - h(\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle))$$

Error in (multiclass) classification

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.
- Useful to see whole matrix to visualize where errors occur.

OPTIMIZATION

Goal: Minimize the logistic loss:

$$L(\boldsymbol{\beta}) = -\sum_{i=1}^{n} y_i \log(h(\boldsymbol{\beta}^T \mathbf{x}_i)) + (1 - y_i) \log(1 - h(\boldsymbol{\beta}^T \mathbf{x}_i))$$

I.e. find $\boldsymbol{\beta}^* = \arg\min L(\boldsymbol{\beta})$. How should we do this?

LOGISTIC REGRESSION GRADIENT

$$L(\beta) = -\sum_{i=1}^{n} y_i \log(h(\beta^T \mathbf{x}_i)) + (1 - y_i) \log(1 - h(\beta^T \mathbf{x}_i))$$

Let $\mathbf{X} \in \mathbb{R}^{d \times n}$ be our data matrix with $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ as rows. Let $\mathbf{y} = [y_1, \ldots, y_n]$. A calculation gives (see notes on webpage):

$$\nabla L(\beta) = \mathbf{X}^T (h(\mathbf{X}\beta) - \mathbf{y})$$

where $h(\mathbf{X}\beta) = \frac{1}{1+e^{-\mathbf{X}\beta}}$. Here all operations are entrywise. I.e in Python you would compute:

```
h = 1/(1 + np.exp(-X@beta))
grad = np.transpose(X)@(h - y)
```

LOGISTIC REGRESSION GRADIENT

To find $\boldsymbol{\beta}$ minimizing $L(\boldsymbol{\beta})$ we typically start by finding a $\boldsymbol{\beta}$ where:

$$\nabla L(\boldsymbol{\beta}) = \mathbf{X}^T \left(h(\mathbf{X}\boldsymbol{\beta}) - \mathbf{y}\right) = \mathbf{0}$$

- In contrast to what we saw when minimizing the squared loss for linear regression, there's no simple closed form expression for such a $\boldsymbol{\beta}$!
- This is the typical situation when minimizing loss in machine learning: linear regression was a lucky exception.
- **Main question:** How do we minimize a loss function $L(\boldsymbol{\beta})$ when we can't explicitly compute where it's gradient is $\mathbf{0}$?

Always an option: Brute-force search. Test our many possible values for $\boldsymbol{\beta}$ and just see which gives the smallest value of $L(\boldsymbol{\beta})$.

- As we saw on Lab 1, this actually works okay for low-dimensional problems (e.g. when $\boldsymbol{\beta}$ has 1 or 2 entries).
- **Problem:** Super computationally expensive in high-dimension. For $\boldsymbol{\beta} \in \mathbb{R}^d$, run time grows as:

Much Better idea. Some sort of guided search for a good of $\boldsymbol{\beta}$.

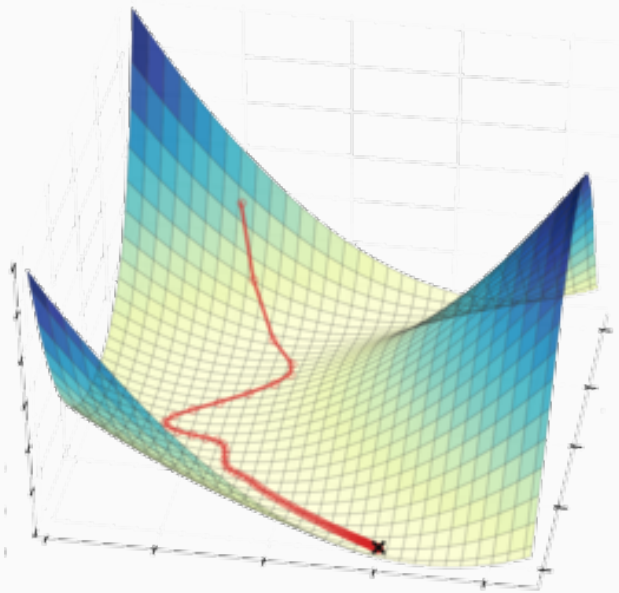
- Start with some $\boldsymbol{\beta}^{(0)}$, and at each step try to change $\boldsymbol{\beta}$ slightly to reduce $L(\boldsymbol{\beta})$.
- Hopefully find an approximate minimizer for $L(\boldsymbol{\beta})$ much more quickly than brute-force search.
- **Concrete goal:** Find $\boldsymbol{\beta}$ with

$$L(\boldsymbol{\beta}) < \min_{\boldsymbol{\beta}} L(\boldsymbol{\beta}) + \epsilon$$

  for some small error term ϵ.

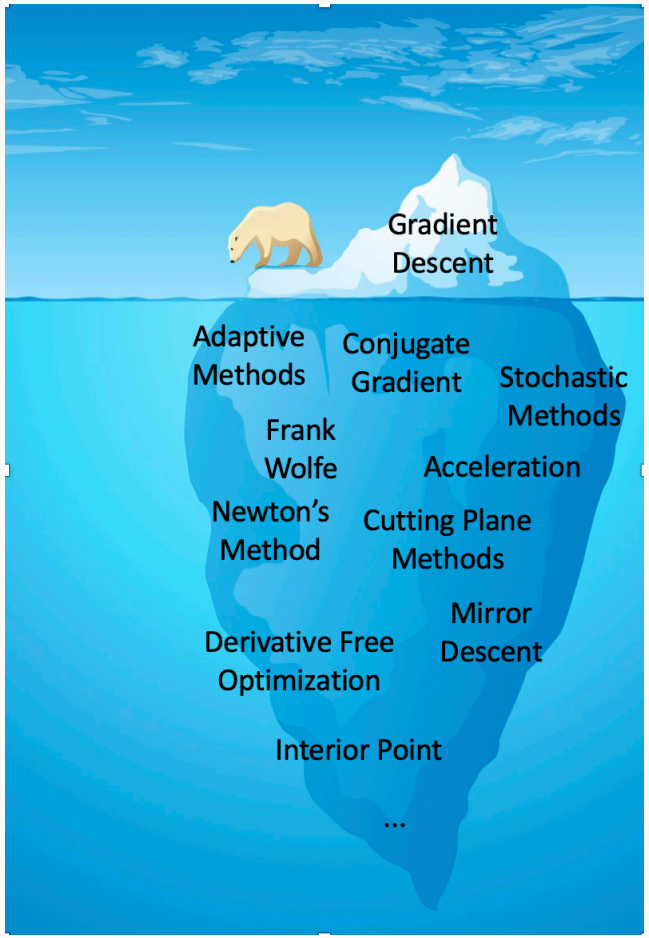
GRADIENT DESCENT

Gradient descent: A greedy search algorithm for minimizing functions of multiple variables (including loss functions) that often works amazingly well.



The single most important computational tool in machine learning. And it's remarkable simple + easy to implement.

OPTIMIZATION ALGORITHMS



Just one method in a huge class of algorithms for numerical optimization. All of these methods are important in ML.

First order oracle model: Given a function L to minimize, assume we can:

- **Function oracle**: Evaluate $L(\beta)$ for any β.
- **Gradient oracle**: Evaluate $\nabla L(\beta)$ for any β.

These are very general assumptions. Gradient descent will not use any other information about the loss function L when trying to find a β which minimizes L.

Basic Gradient descent algorithm:

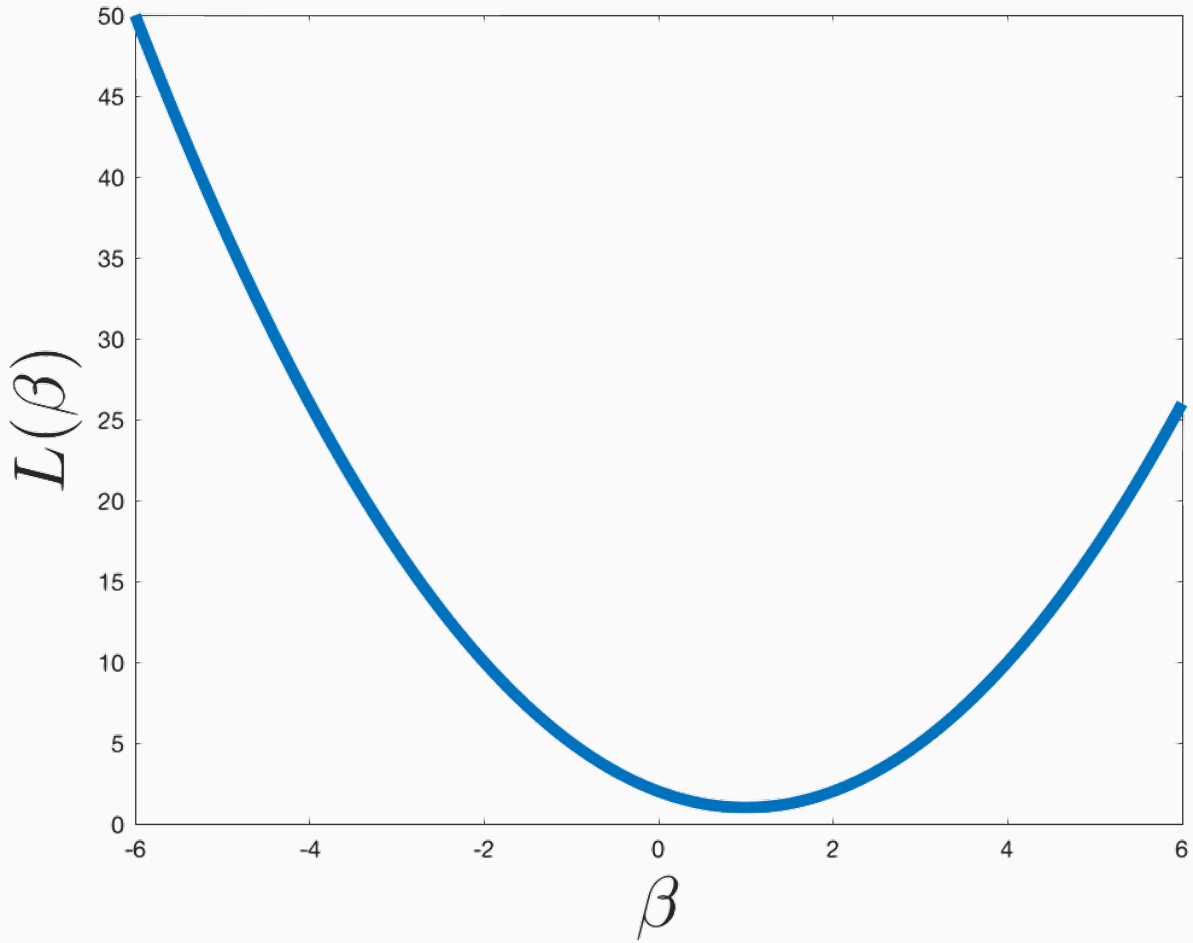
- Choose starting point $\boldsymbol{\beta}^{(0)}$.
- For $i = 0, \ldots, T$:
  - $\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} - \eta \nabla L(\boldsymbol{\beta}^{(i)})$
- Return $\boldsymbol{\beta}^{(T)}$.

$\eta > 0$ is a step-size parameter. Also called the learning rate.

Why does this method work?

First observation: if we actually reach the minimizer $\boldsymbol{\beta}^*$ then we stop.

Consider a 1-dimensional loss function. I.e. where β is just a single value. Our update step is $\beta^{(i+1)} = \beta^{(i)} - \eta L'(\beta^{(i)})$



Mathematical way of thinking about it:

By definition, $L'(\beta) = \lim_{\Delta \to 0} \frac{L(\beta+\Delta)-L(\beta)}{\Delta}$. So for small values of Δ, we expect that:

$$L(\beta + \Delta) - L(\beta) \approx \Delta \cdot L'(\beta).$$

We want $L(\beta + \Delta)$ to be smaller than $L(\beta)$, so we want $\Delta \cdot L'(\beta)$ to be negative.

This can be achieved by choosing $\Delta = -L'(\beta)$, or really $\Delta = -\eta \cdot L'(\beta)$ for positive step size η.

$$\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} - \eta L'(\boldsymbol{\beta}^{(i)})$$

DIRECTIONAL DERIVATIVES

For high dimensional functions ($\boldsymbol{\beta} \in \mathbb{R}^d$), our update involves a vector $\mathbf{v} \in \mathbb{R}^d$. At each step:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \mathbf{v}.$$

Question: When $\mathbf{v}$ is small, what's an approximation for $L(\boldsymbol{\beta} + \mathbf{v}) - L(\boldsymbol{\beta})$?

$$L(\boldsymbol{\beta} + \mathbf{v}) - L(\boldsymbol{\beta}) \approx$$

We have

$$L(\boldsymbol{\beta} + \mathbf{v}) - L(\boldsymbol{\beta}) \approx \frac{\partial L}{\partial \beta_1} v_1 + \frac{\partial L}{\partial \beta_2} v_2 + \ldots + \frac{\partial L}{\partial \beta_d} v_d$$
$$= \langle \nabla L(\boldsymbol{\beta}), \mathbf{v} \rangle.$$

How should we choose $\mathbf{v}$ so that $L(\boldsymbol{\beta} + \mathbf{v}) < L(\boldsymbol{\beta})$?

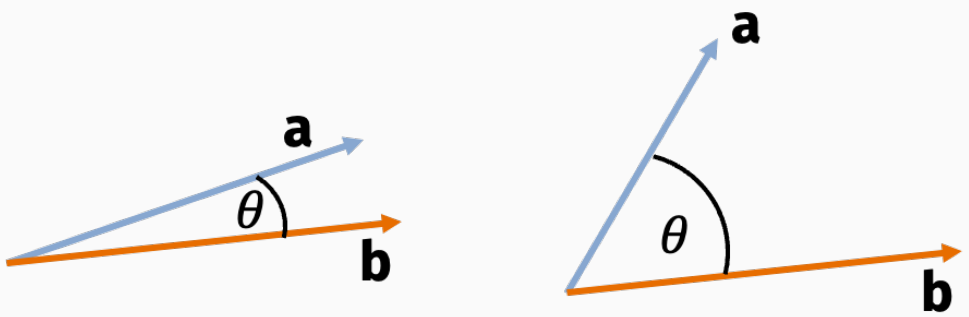
[3] Formally, you might remember that we can define the **directional derivative** of a multivariate function: $D_{\mathbf{v}} L(\boldsymbol{\beta}) = \lim_{\Delta \to 0} \frac{L(\boldsymbol{\beta} + \Delta \mathbf{v}) - L(\boldsymbol{\beta})}{\Delta}$.

Claim (Gradient descent = Steepest descent[4])

$$\frac{-\nabla L(\beta)}{\|\nabla L(\beta)\|_2} = \arg\min_{\mathbf{v}, \|\mathbf{v}\|_2 = 1} \langle \nabla L(\beta), \mathbf{v} \rangle$$

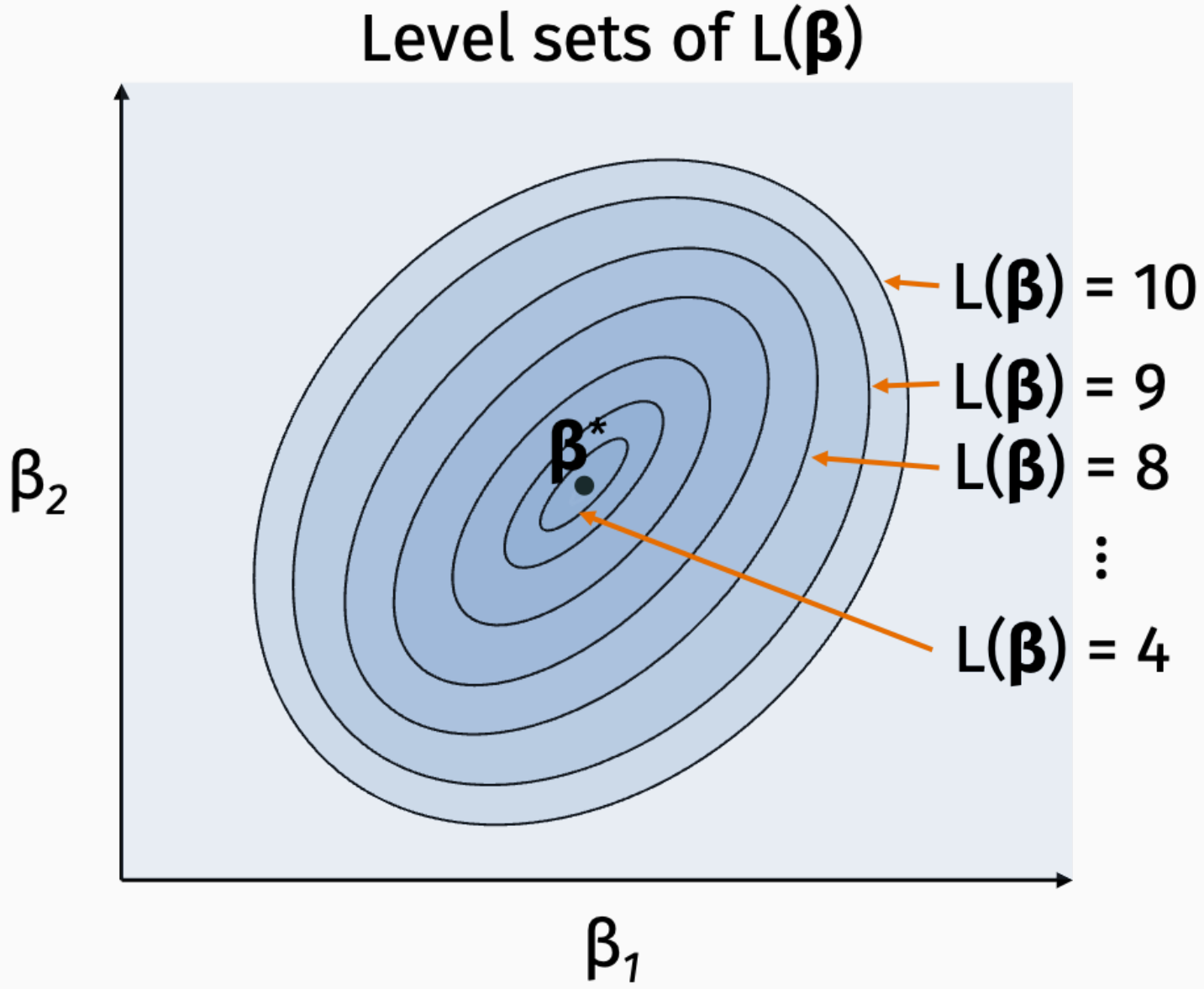
Recall: For two vectors $\mathbf{a}, \mathbf{b}$,

$$\langle \mathbf{a}, \mathbf{b} \rangle = \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 \cdot \cos(\theta)$$



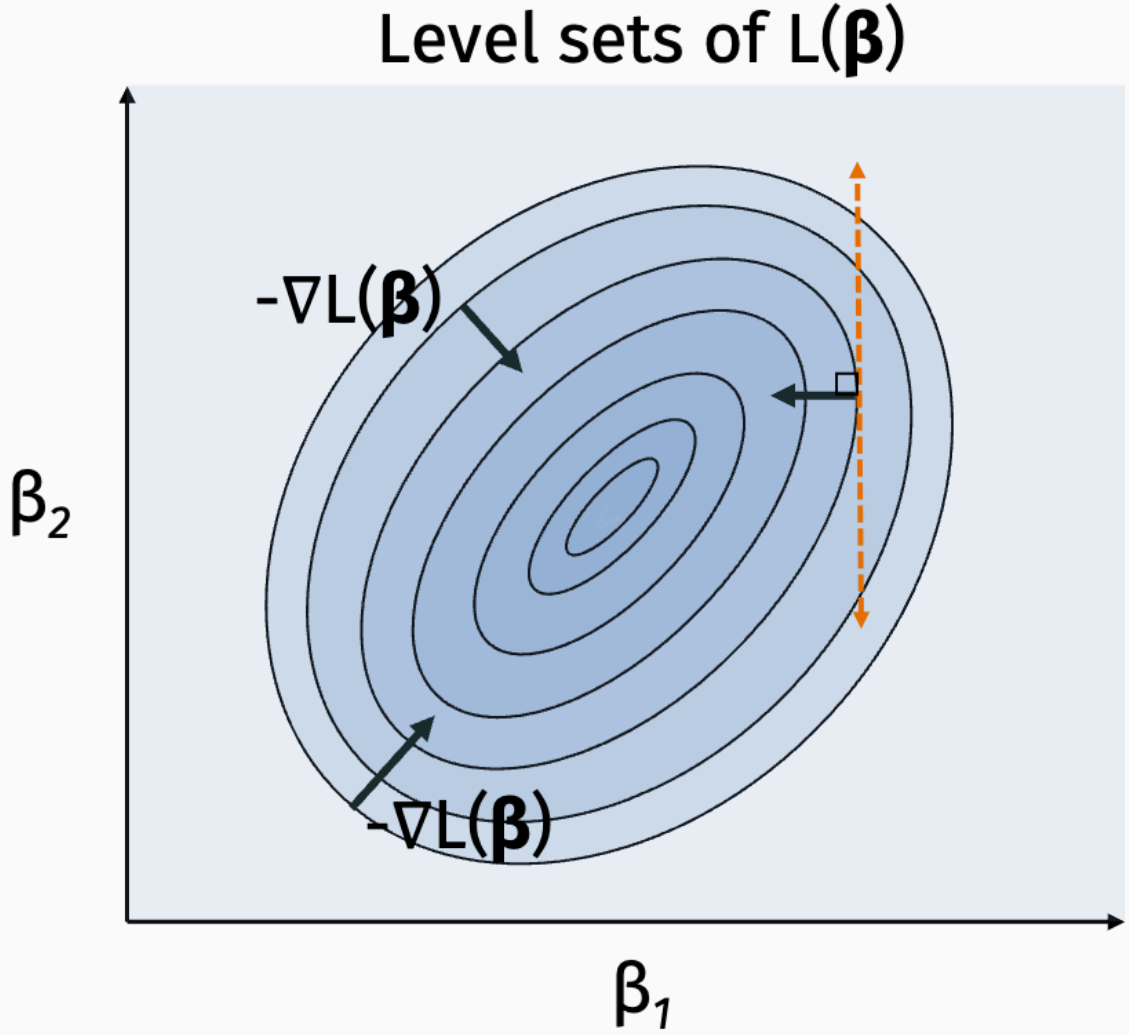
[4] We could have restricted $\mathbf{v}$ using a different norm. E.g. $\|\mathbf{v}\|_1 \leq 1$ or $\|\mathbf{v}\|_\infty = 1$. These choices lead to variants of generalized steepest descent..

VISUALIZING IN 2D



Claim (Gradient descent = Steepest descent)

$$\frac{-\nabla L(\boldsymbol{\beta})}{\|\nabla L(\boldsymbol{\beta})\|_2} = \arg\min_{\mathbf{v}, \|\mathbf{v}\|_2 = 1} \langle \nabla L(\boldsymbol{\beta}), \mathbf{v} \rangle$$



Basic Gradient descent (GD) algorithm:

- Choose starting point $\boldsymbol{\beta}^{(0)}$.
- For $i = 0, \ldots, T$:
  - $\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} - \eta \nabla L(\boldsymbol{\beta}^{(i)})$
- Return $\boldsymbol{\beta}^{(T)}$.

- **Theoretical questions:** Does gradient descent always converge to the minimum of the loss function L? Can you prove how quickly?
- **Practical questions:** How to choose η? Any other modifications needed for good practical performance?