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

NYU Tandon School of Engineering, Akbar Rafiey

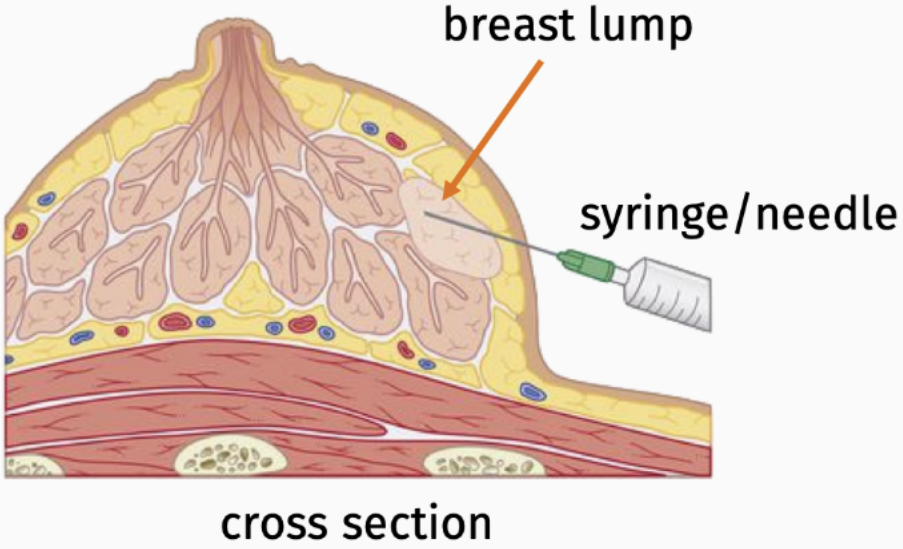
Course news

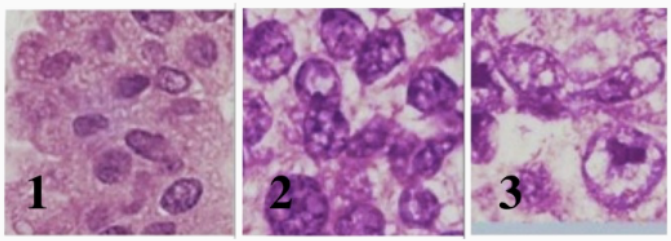
- HW02 is posted. Due on March 05.
- Lab 03 due next week February 26.

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).





Motivating problem

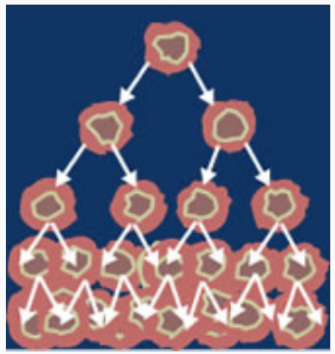
Demo: `demo_breast_cancer.ipynb`

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: 9 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 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).

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 $y_1, \ldots, y_n \in \{1, \ldots, q\}$.

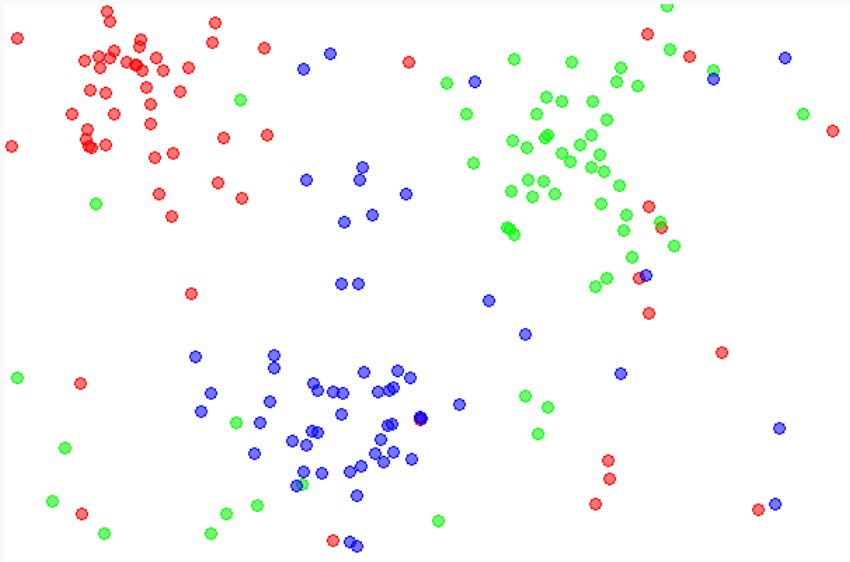
Classification algorithm:

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

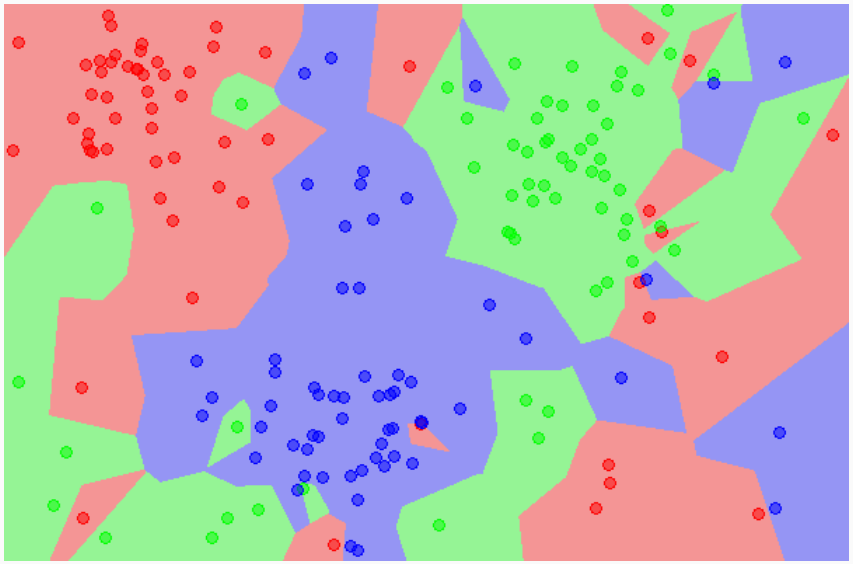
- Compute $sim(\mathbf{x}_{new}, \mathbf{x}_1), \ldots, sim(\mathbf{x}_{new}, \mathbf{x}_n)$.[1]
- Let $\mathbf{x}_{j_1}, \ldots, \mathbf{x}_{j_k}$ be the training data vectors with highest similarity to $\mathbf{x}_{new}$.
- Predict y_{new} as $majority(y_{j_1}, \ldots, y_{j_k})$.

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

k-nearest neighbor method



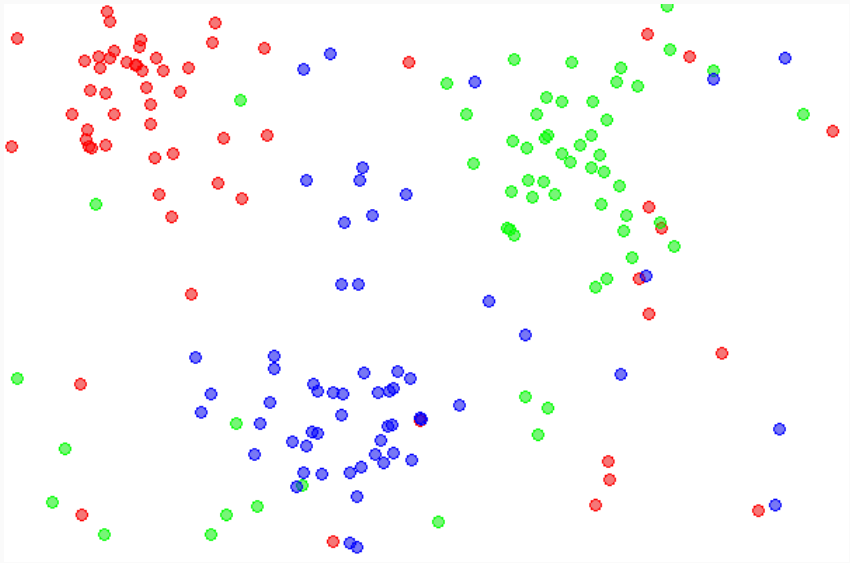
Data



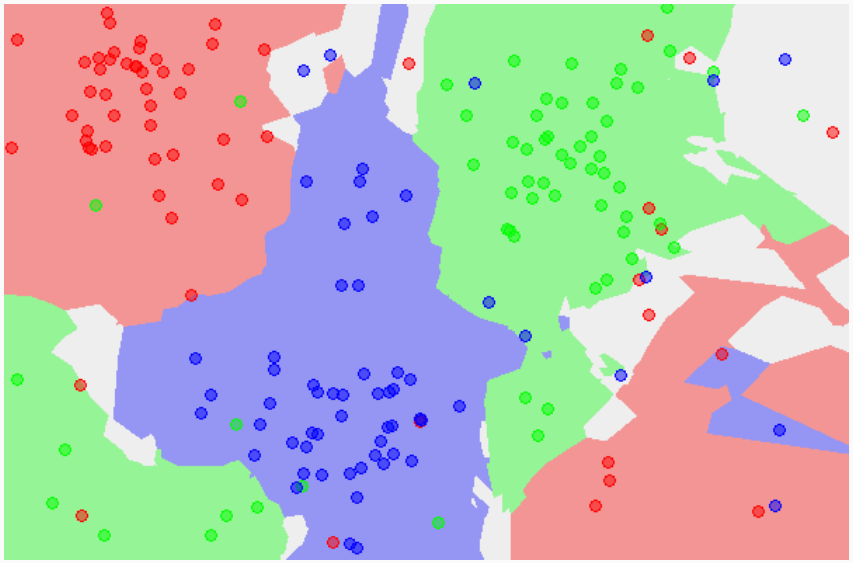
1-NN classifier

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

k-nearest neighbor method



Data



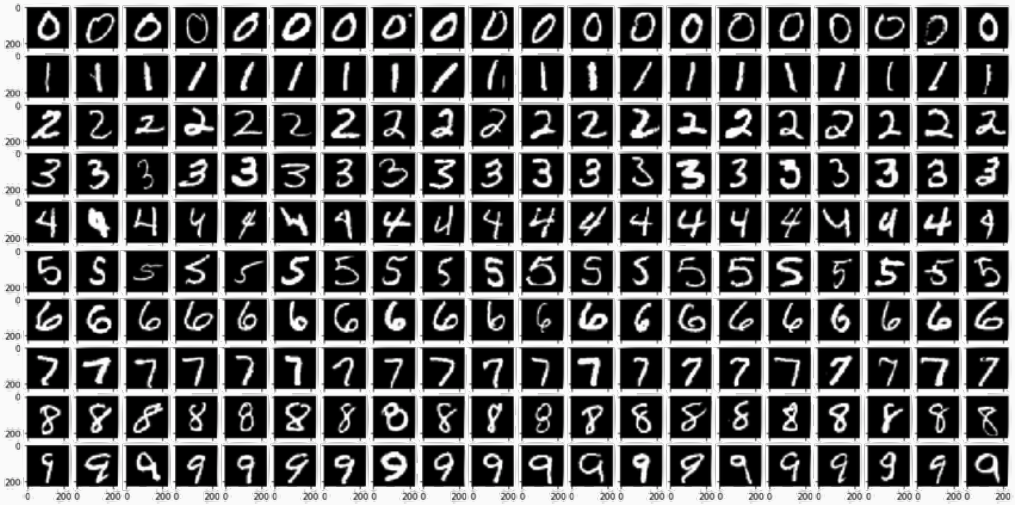
5-NN classifier

- Smaller k, more complex classification function.
- Larger k, more robust to noisy labels.
- Works remarkably well for many datasets.

MNIST image data

Especially good for large datasets with lots of repetition.

Example: works well on MNIST ($\approx 95\%$ accuracy out-of-the-box.):



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

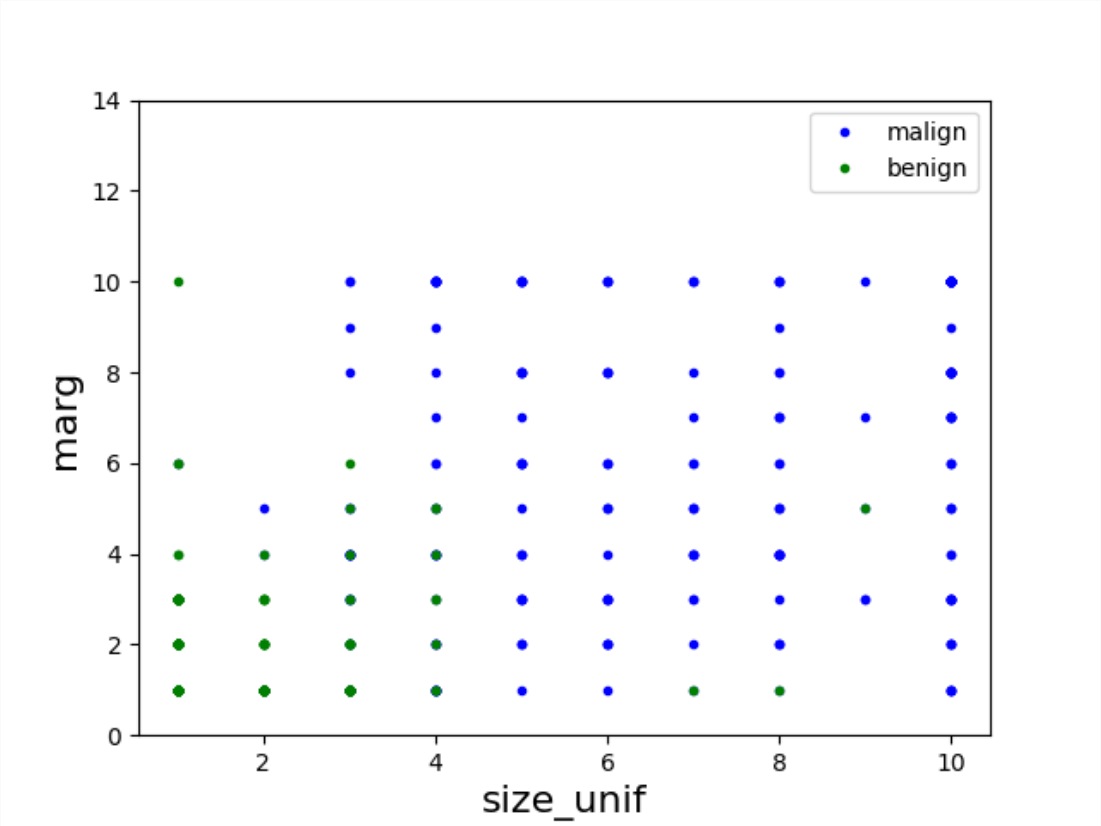
One issue is that prediction can be computationally intensive...

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

Linear classification

Begin by plotting data

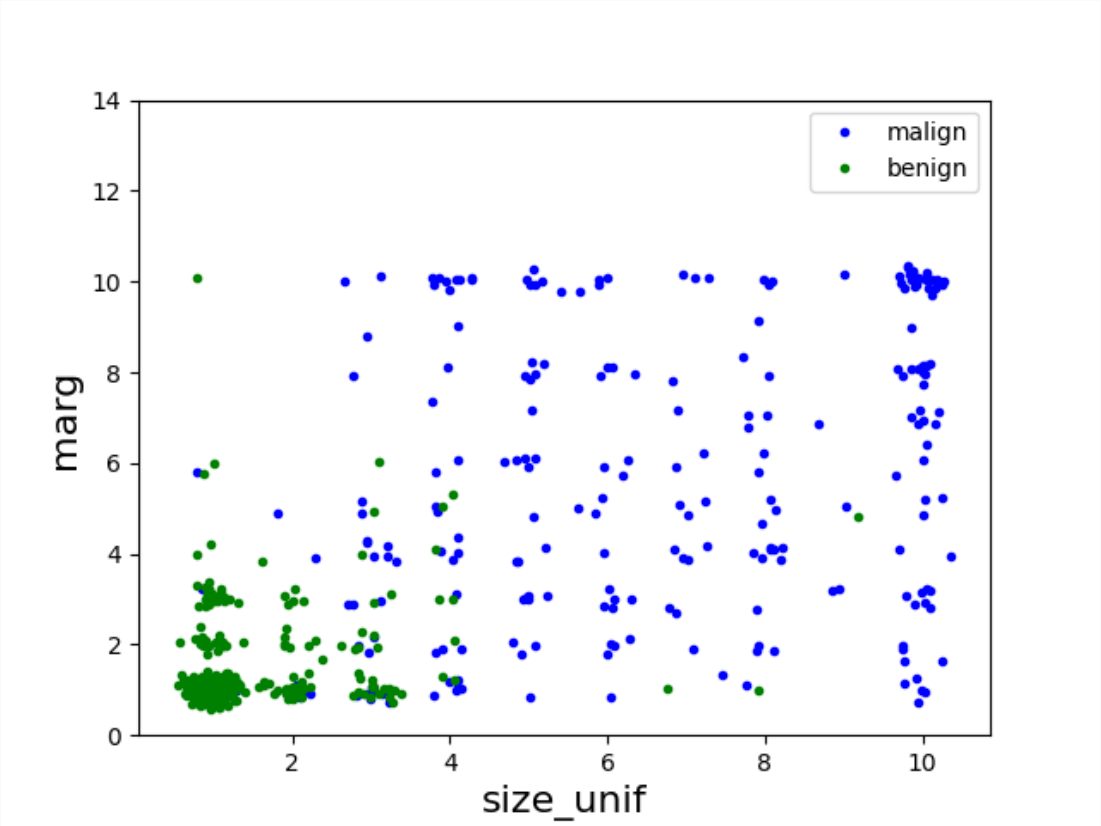
We pick two variables, Margin Adhesion and Size Uniformity and plot a scatter plot. Points with label 1 (malignant) are plotted in blue, those with label 2 (benign) are plotted in green.



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

Plotting with jitter

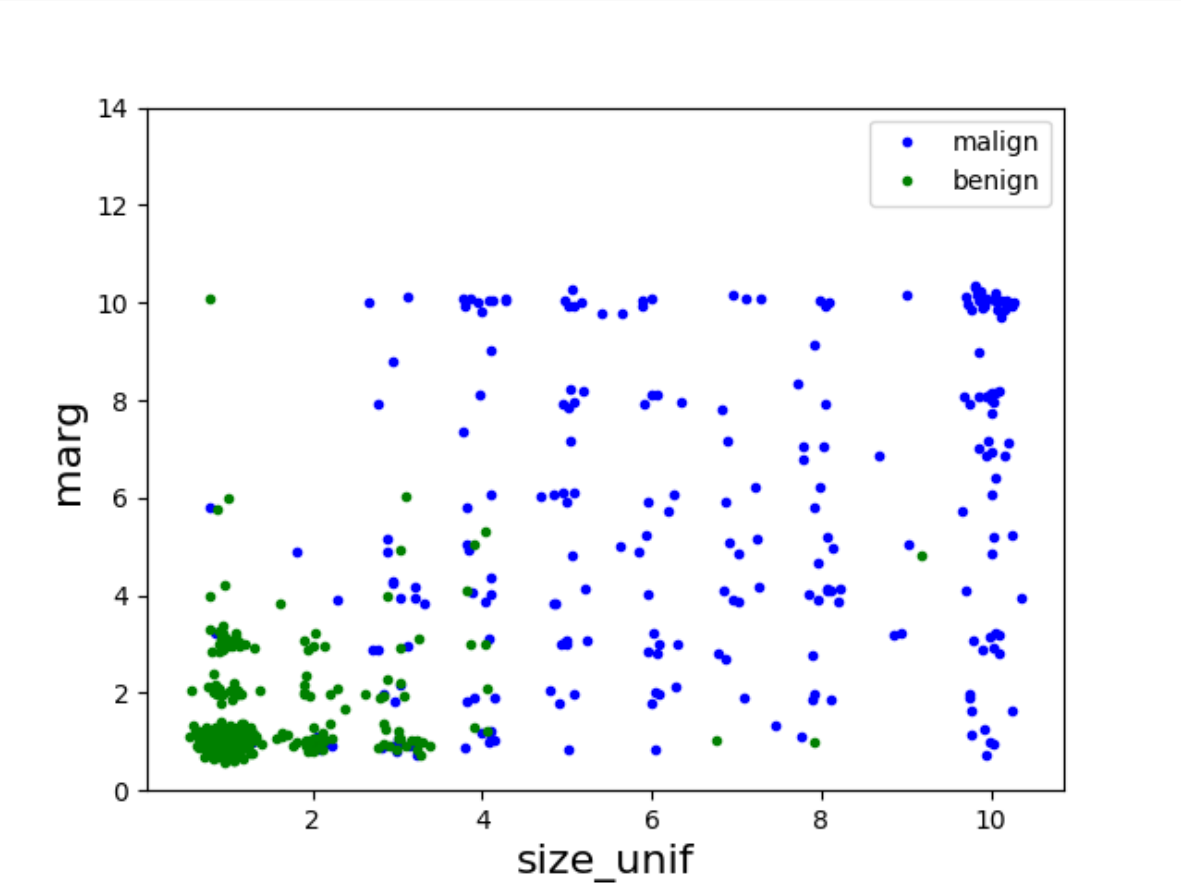
Simple + Useful Trick: data jittering. Add tiny random noise (using e.g. `np.random.randn`) to data to prevent overlap.



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

Brainstorming

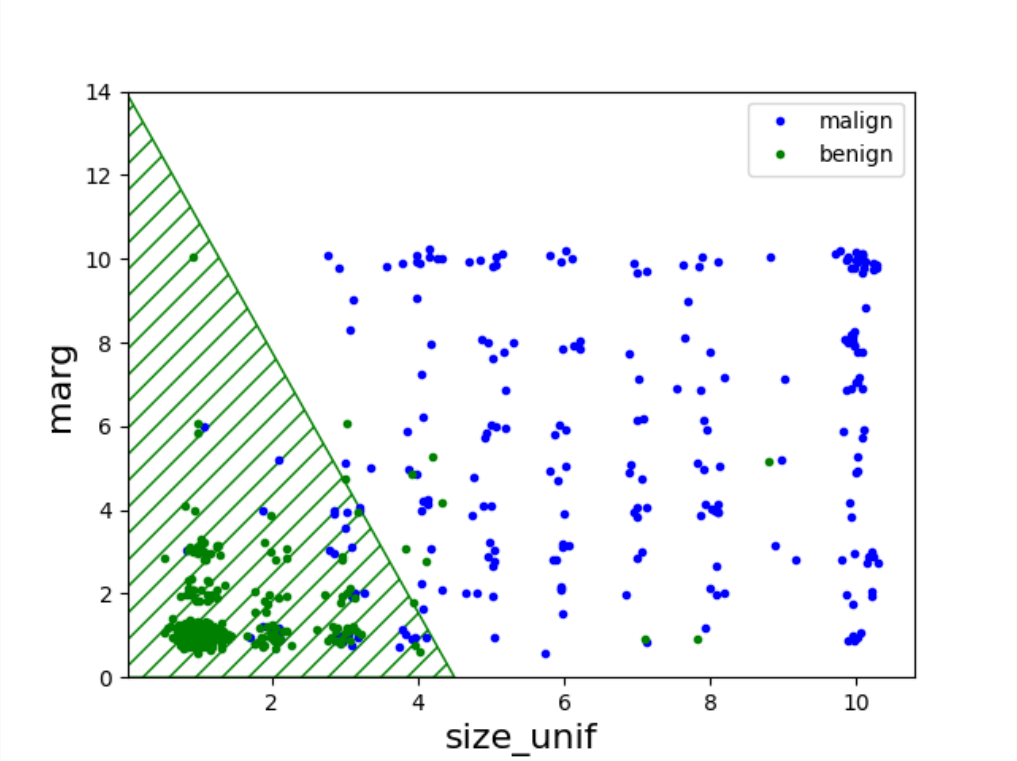
Any ideas for possible classification rules for this data?



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 $\langle \mathbf{x}_i, \boldsymbol{\beta} \rangle > \lambda$

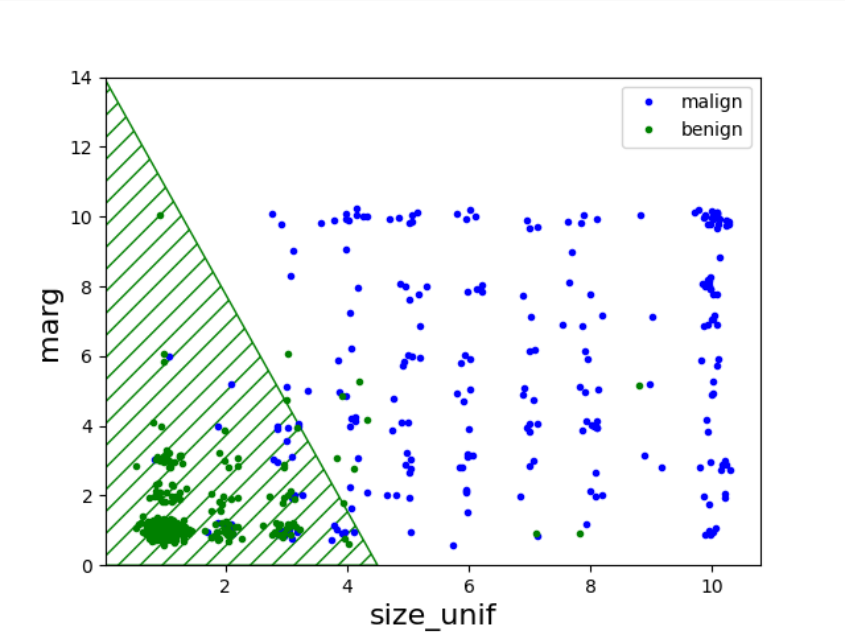


Line has equation $\langle \mathbf{x}, \boldsymbol{\beta} \rangle = \lambda$.

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 $\langle \mathbf{x}_i, \boldsymbol{\beta} \rangle > 0$

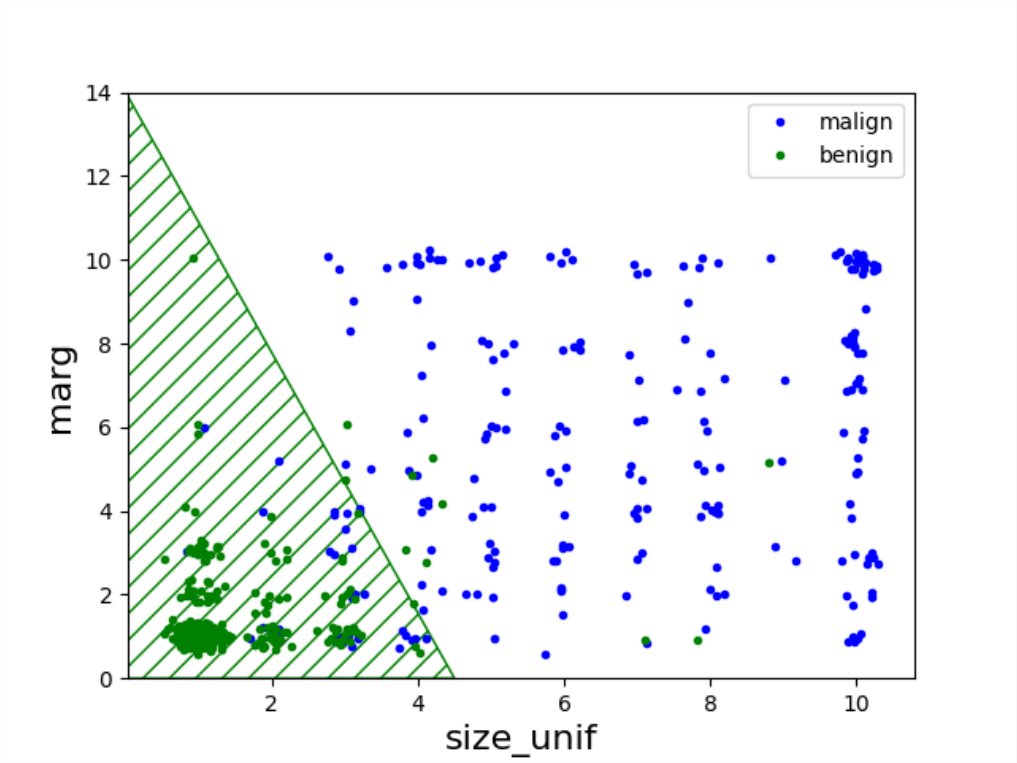


Line has equation $\langle \mathbf{x}, \boldsymbol{\beta} \rangle = 0$.

Linear classification

Standard approach for binary classification of real-valued data:

- Find parameter vector $\boldsymbol{\beta}$.
- For input data vector $\mathbf{x}$, predict 0 if $\boldsymbol{\beta}^T\mathbf{x} \leq \lambda$ and 1 if $\boldsymbol{\beta}^T\mathbf{x} > \lambda$ for some threshold λ.[3]



[3] Can always assume $\lambda = 0$ if $\mathbf{x}$ has an intercept term.

0 - 1 loss

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

Loss minimization approach (first attempt):

- **Model**[4]:

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

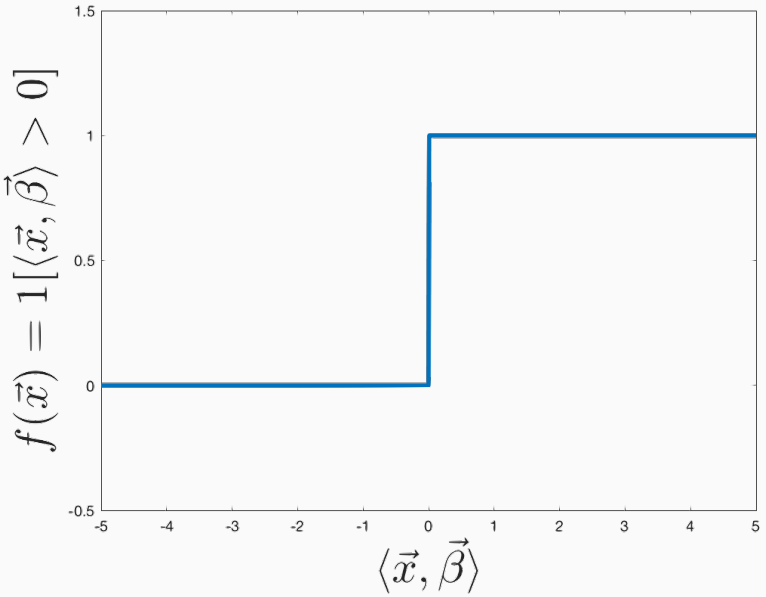
- **Loss function**: "$0-1$ Loss"

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

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

Linear classifier via square loss

Loss minimization approach (second attempt):

- **Model**:

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

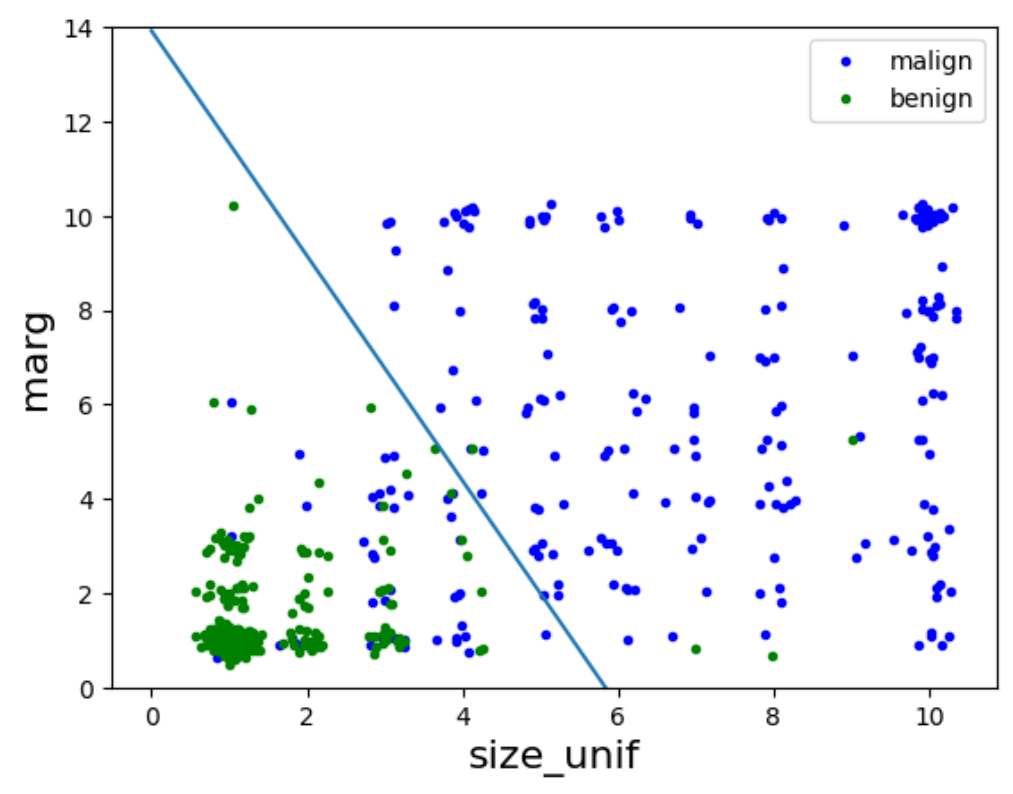
- **Loss function**: "Square Loss"

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

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

Linear classifier via square loss

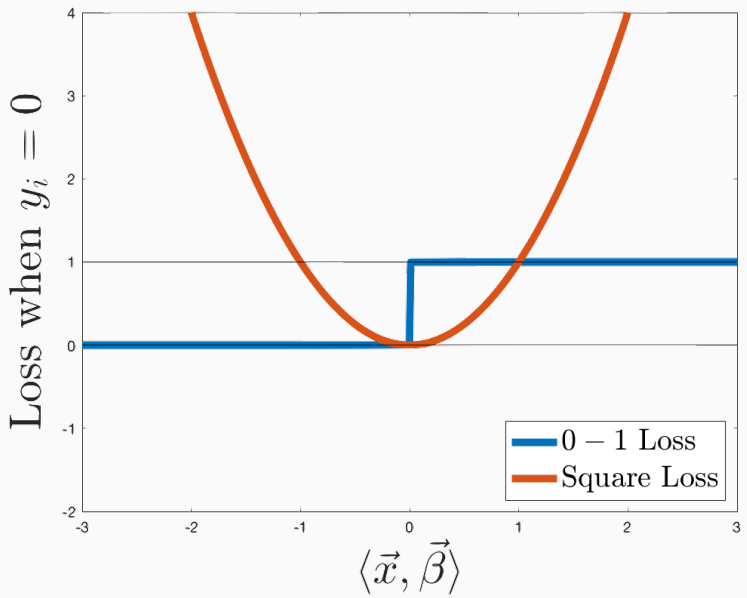
We can solve for β by just solving a least squares multiple linear regression problem.



Do you see any issues here?

Linear classifier via square loss

Problem with square loss:

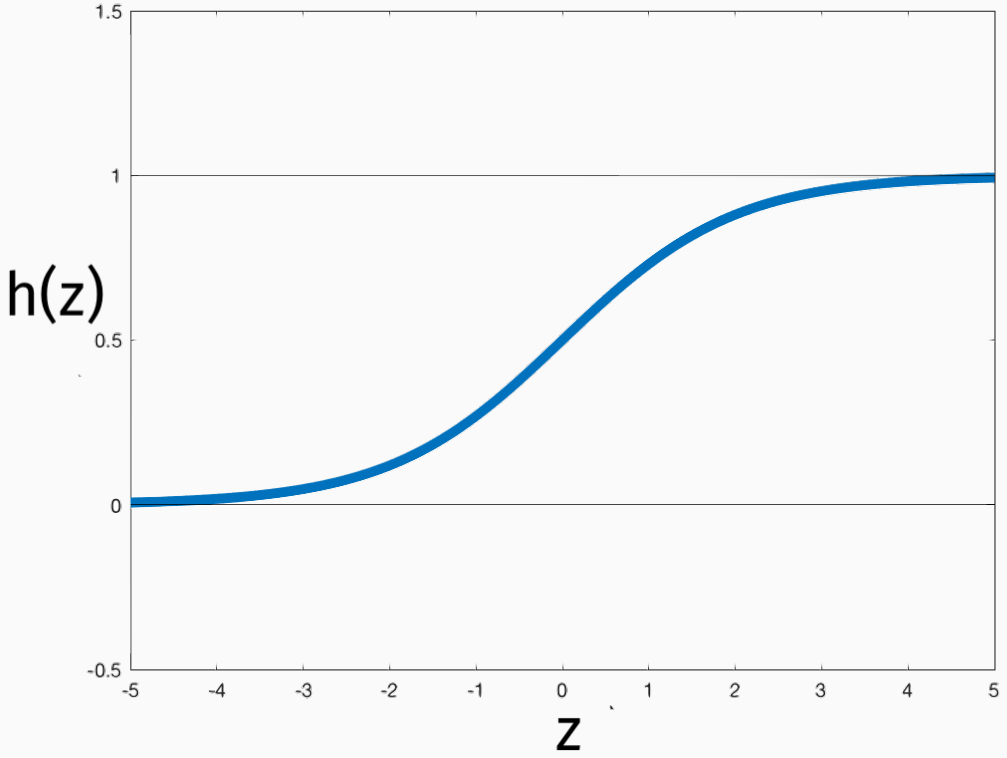


- Loss increases if $\langle \mathbf{x}, \boldsymbol{\beta} \rangle > 1$ even if correct label is 1. Or if $\langle \mathbf{x}, \boldsymbol{\beta} \rangle < 0$ even if correct label is 0.
- Intuitively we don't want to "punish" these cases.

Logistic regression

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

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



Logistic regression

Loss minimization approach (this works!):

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

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

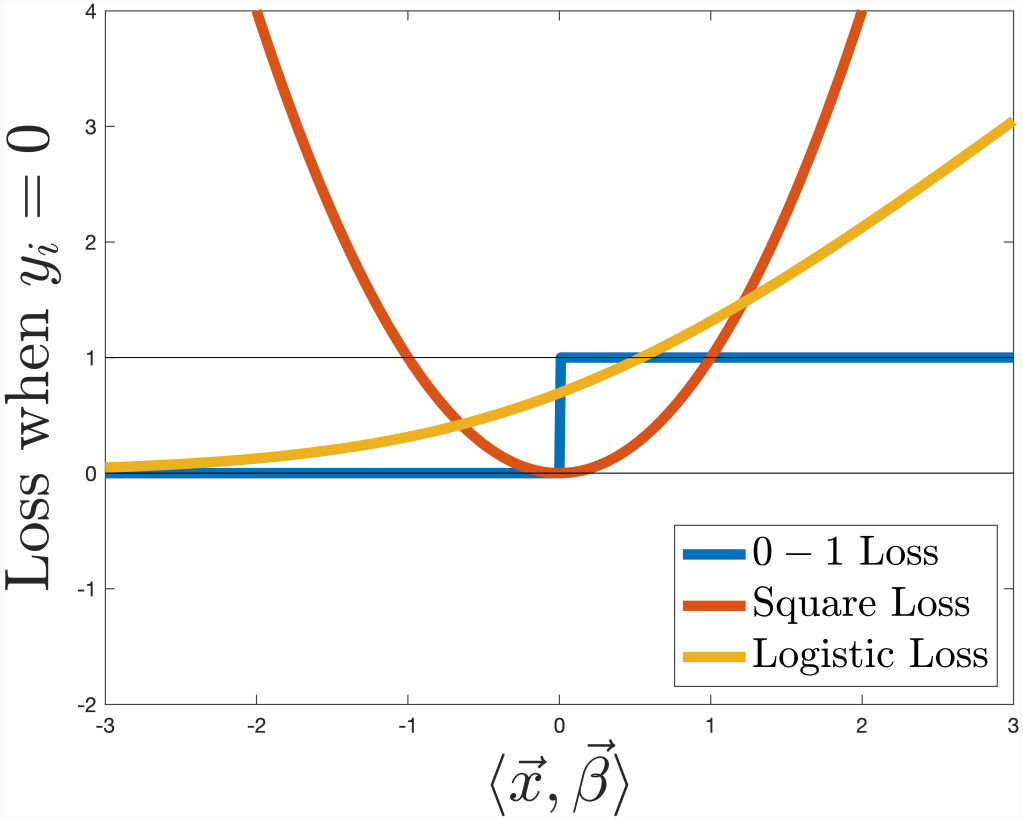
- **Loss function**: "Logistic loss" aka "binary cross-entropy loss"

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

Logistic loss

Logistic Loss:

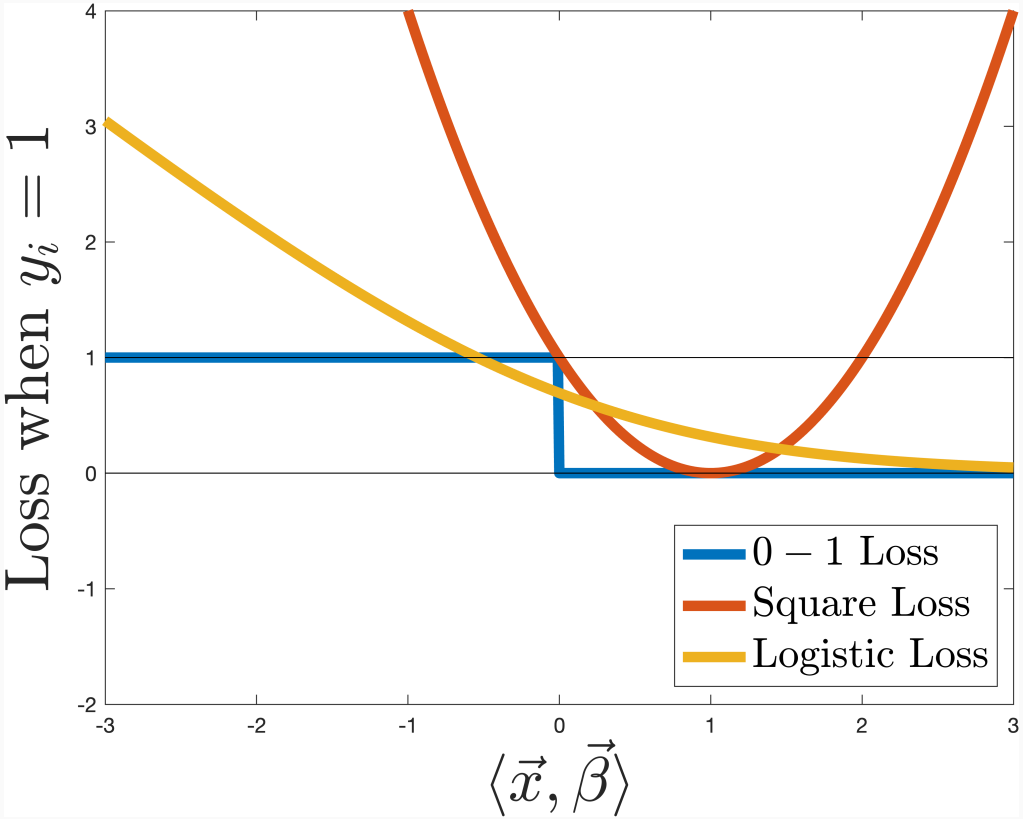
$L(\beta) = -\sum_{i=1}^{n} y_i \log(h_\beta(\mathbf{x})) + (1 - y_i) \log(1 - h_\beta(\mathbf{x}))$



Logistic loss

Logistic Loss:

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



Logistic regression

Loss minimization approach:

- Given training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$.
- Minimize "Logistic loss" aka "binary cross-entropy loss"

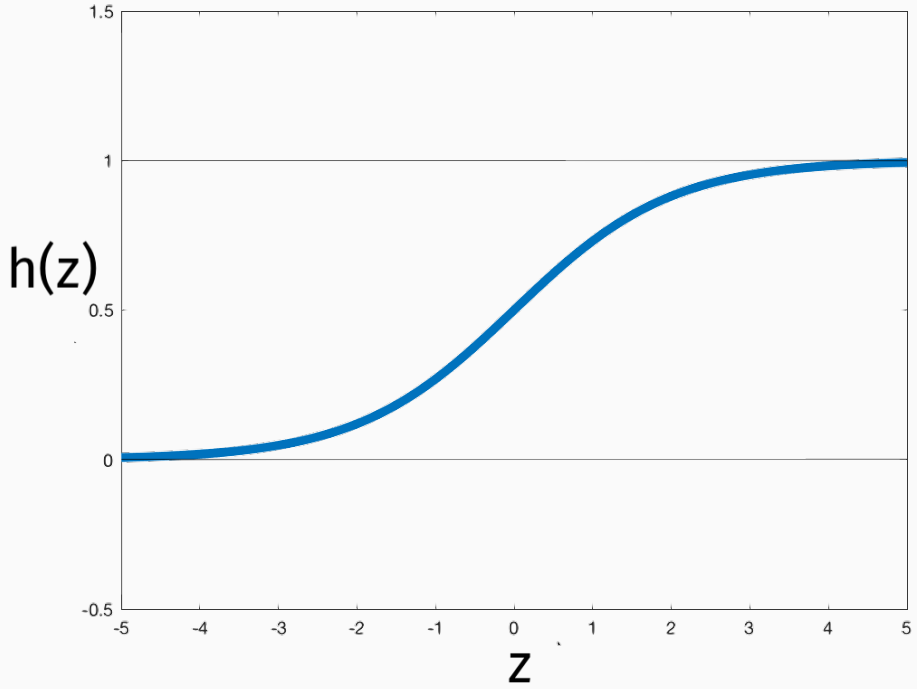
$$L(\beta) = -\sum_{i=1}^{n} y_i \log(h(\beta^T \mathbf{x}_i)) + (1 - y_i) \log(1 - h(\beta^T \mathbf{x}_i))$$

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

Prediction: Predict $y_i = 1$ if $\beta^T \mathbf{x}_i \geq 0$, predict 0 otherwise.

Logistic regression

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



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

Exercise

Why not minimize:

$$L(\beta) = \sum_{i=1}^{n} \Big(y_i - h(\mathbf{x}^T \beta) \Big)^2 ?$$

Exercise

Why not minimize:

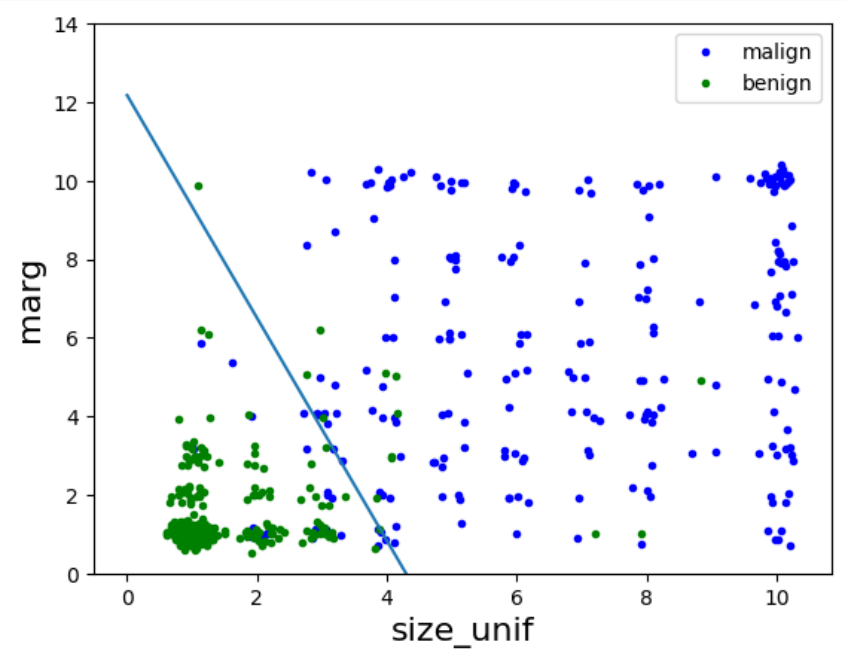
$$L(\beta) = \sum_{i=1}^{n} \Big(y_i - h(\mathbf{x}^T \beta) \Big)^2 ?$$

Answer: This is actually a pretty reasonable thing to do. An important issue however is that the loss here is not convex, which makes it hard to find the β that minimizes the loss.

Log-loss on the other hand is convex. More on this later.

Logistic loss

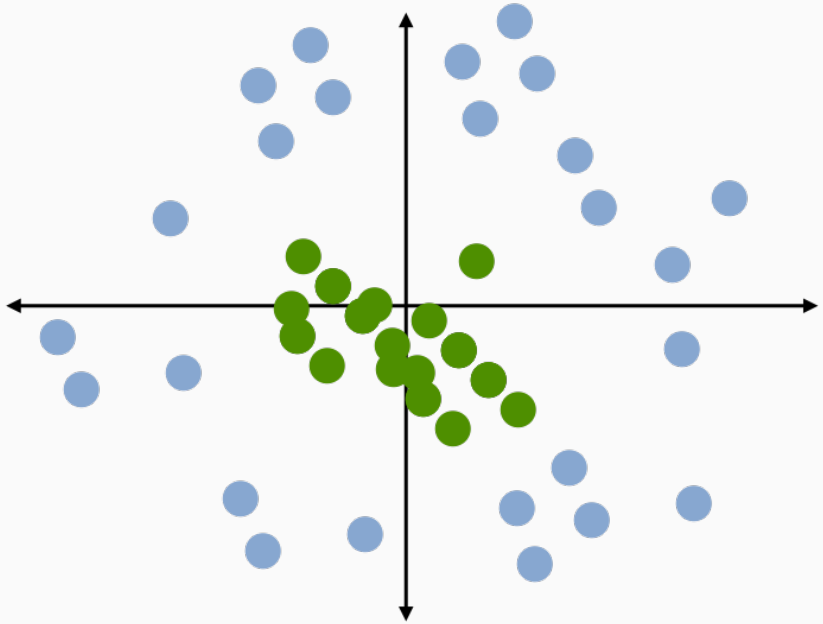
- Convex function in β, can be minimized using gradient descent.
- Works well in practice.
- Good Bayesian motivation.
- Easily combined with non-linear data transformations.



Fit using logistic regression/log loss.

Non-linear transformations

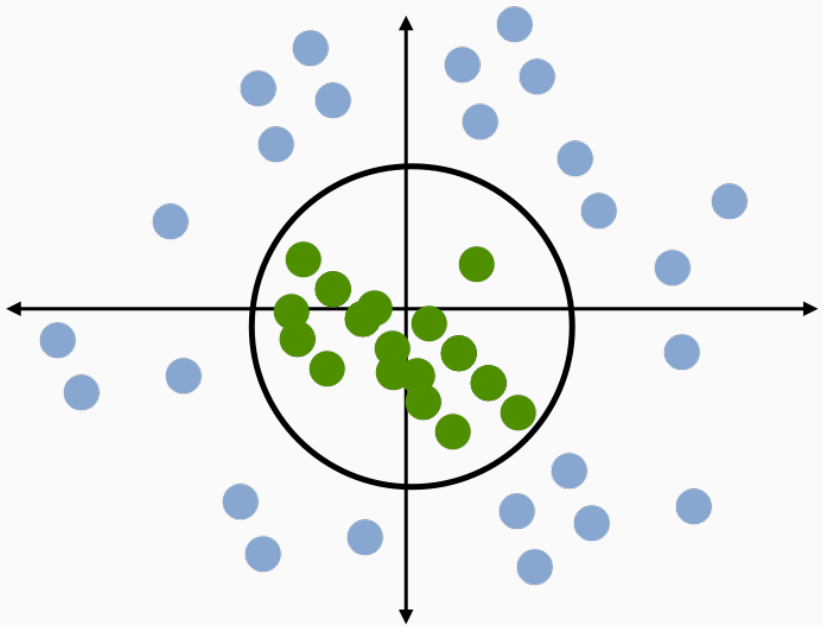
How would we learn a classifier for this data using logisitic regression?



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

Non-linear transformations

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

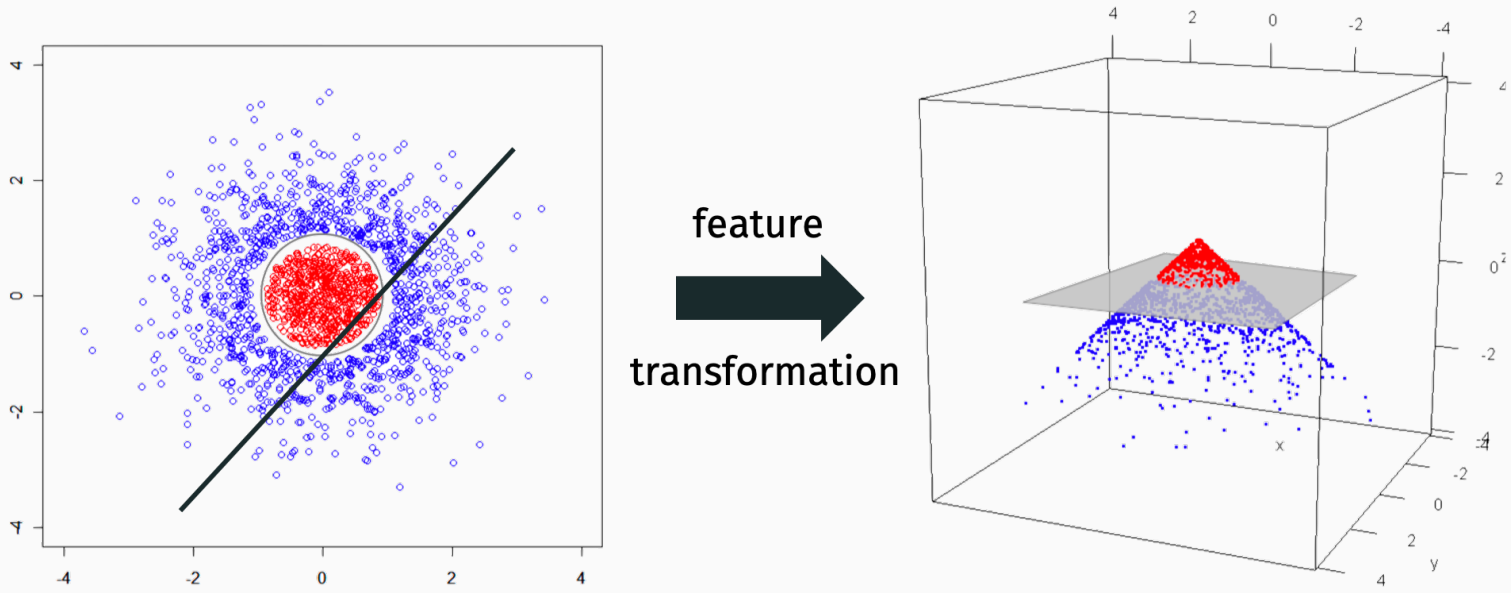


- Predict class 1 if $x_1^2 + x_2^2 < \lambda$.
- Predict class 0 if $x_1^2 + x_2^2 \geq \lambda$.

This is a linear classifier on our transformed data set. Logisitic regression might learn $\boldsymbol{\beta} = [r^2, 0, 0, 1, 1, 0]$.

Non-linear transformations

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



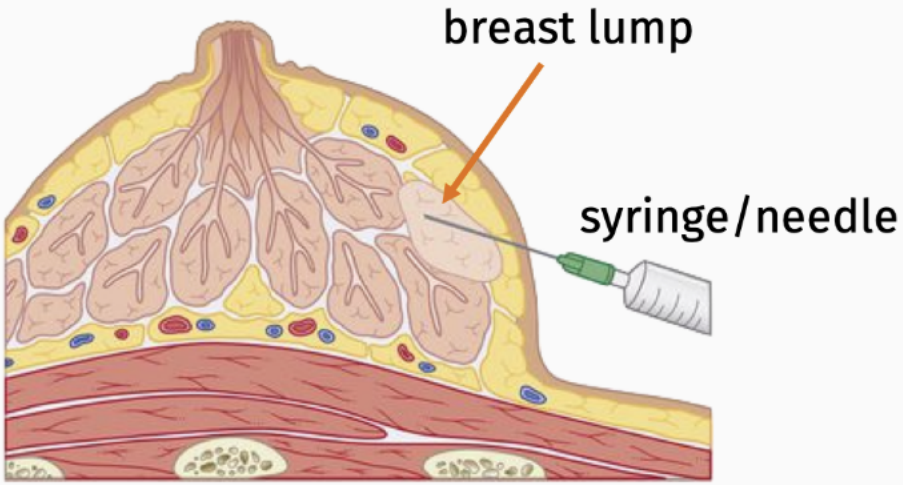
Lots more on this in future lecture!

Error in classification

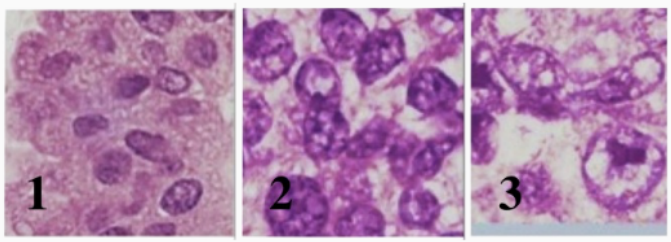
Once we have a classification algorithm, how do we judge its performance?

- **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.



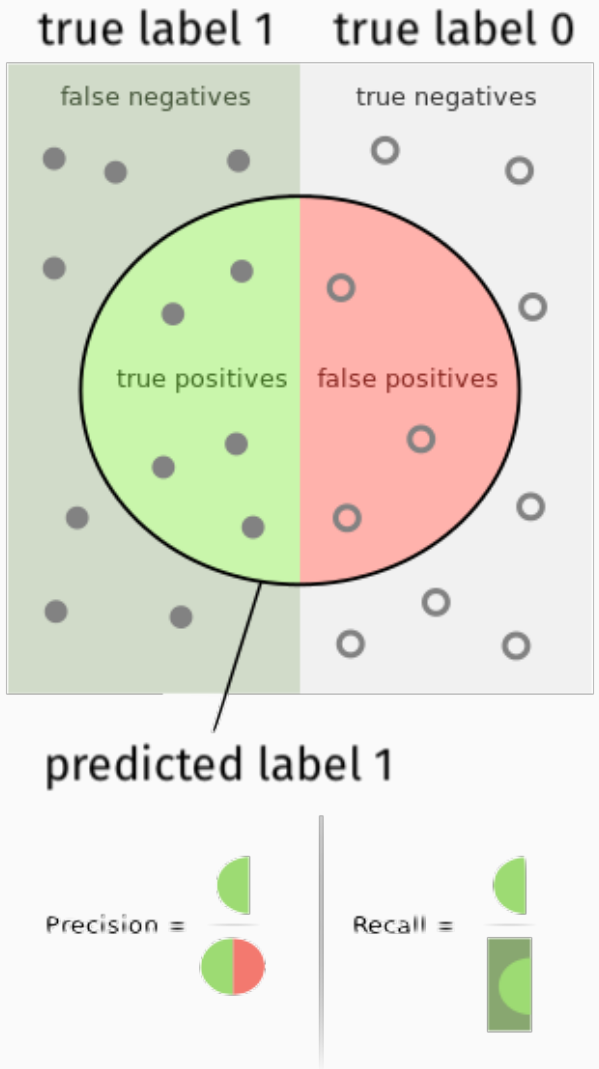
cross section



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.

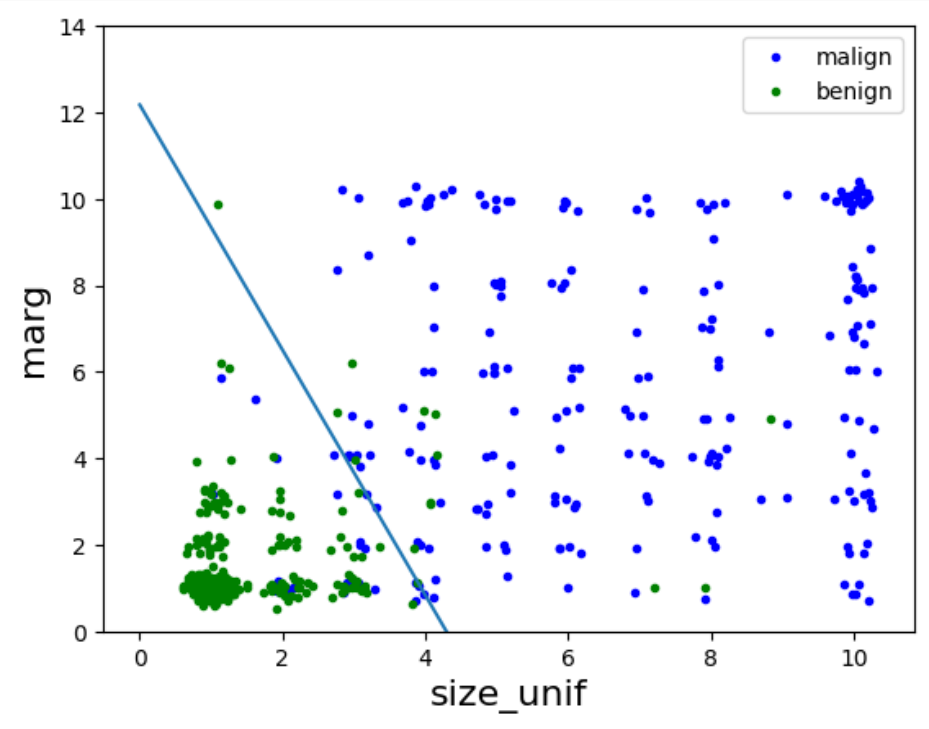
Question: Which should we optimize for medical diagnosis?



Error in classification

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 $1/2$. How does changing λ affect precision and recall ?



Error in classification

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 $1/2$. How does changing λ affect precision and 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.

Multi-class

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

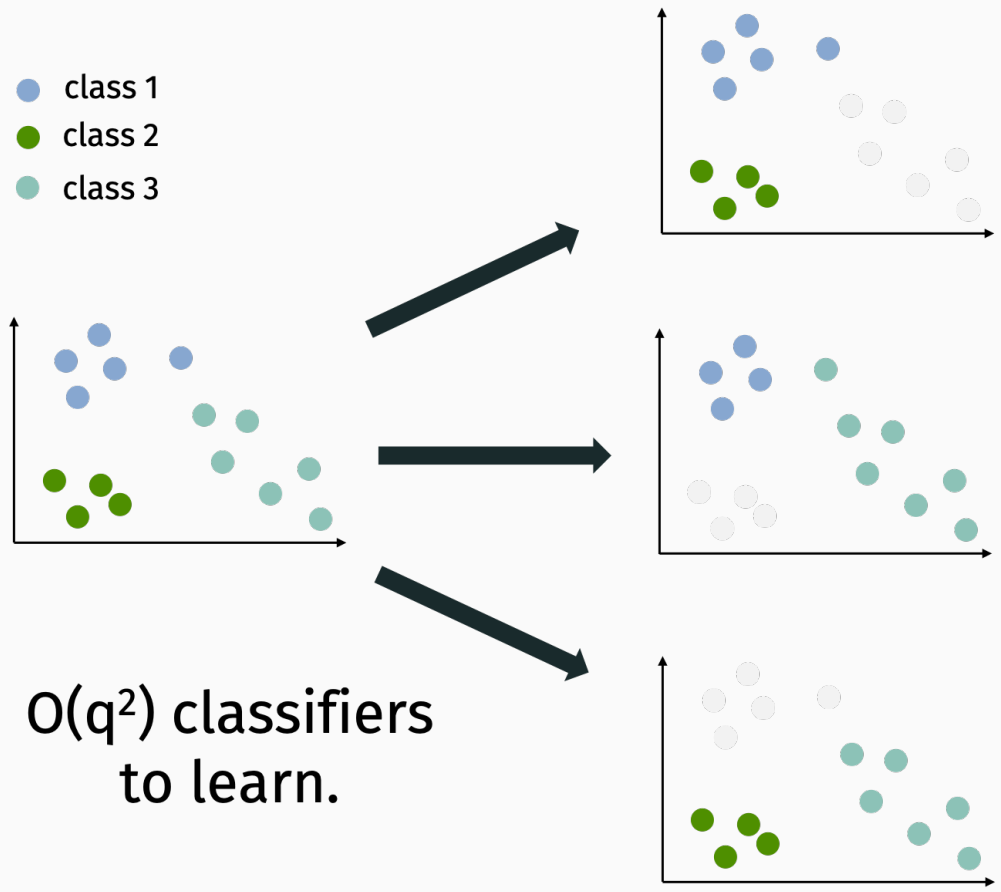
Two common options for reducing multi-class problems to binary problems:

- 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 $\langle \beta^{(i)}, \mathbf{x}_{new} \rangle \geq 0$. Could be ambiguous!
- **Better:** Assign y to class i with maximum value of $h(\langle \beta^{(i)}, \mathbf{x}_{new} \rangle)$.

One vs. one



- For q classes train $\frac{q(q-1)}{2}$ classifiers.
- Assign y to class 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.
- Also tends to work better when there is class imbalance.
- But one-vs.-one can be super expensive! E.g when $q = 100$ or $q = 1000$.

Multiclass logistic regression

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?

Multiclass logistic regression

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:

$$\begin{bmatrix} -10 & 4 & 1 & 0 & -5 \end{bmatrix} \to \begin{bmatrix} .00 & .94 & .04 & .02 & .00 \end{bmatrix}$$

Multiclass logistic regression

Multi-class cross-entropy:

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

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

Binary cross-entropy:

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

$$= -\sum_{i:y_i=1} \log(h(\beta^T \mathbf{x}_i)) - \sum_{i:y_i=0} \log(1 - h(\beta^T \mathbf{x}_i))$$

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

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

Logistic regression

Goal: Minimize the logistic loss:

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

I.e. find $\beta^* = \arg\min L(\beta)$. 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 (verify!):

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

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 β minimizing $L(\beta)$ we typically start by finding a β where:

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

- In contrast to what we saw when minimizing the squared loss for linear regression, there's no simple closed form expression for such a β!
- 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 $L(\beta)$ when we can't explicitly compute where it's gradient is $\mathbf{0}$?

Minimizing loss functions

Always an option: Brute-force search. Test as 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:

Minimizing loss functions

Much Better idea. Some sort of guided search for a good of β.

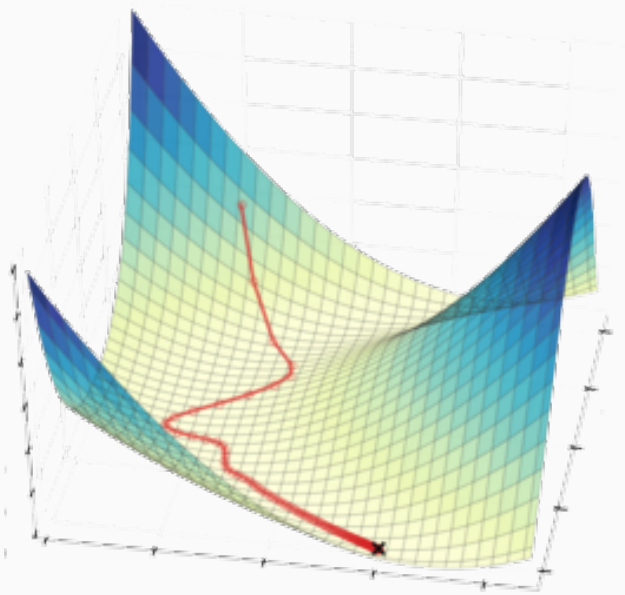
- Start with some $\beta^{(0)}$, and at each step try to change β slightly to reduce $L(\beta)$.
- Hopefully find an approximate minimizer for $L(\beta)$ much more quickly than brute-force search.
- **Concrete goal:** Find β with

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

for some small error term ϵ.

Gradient descent

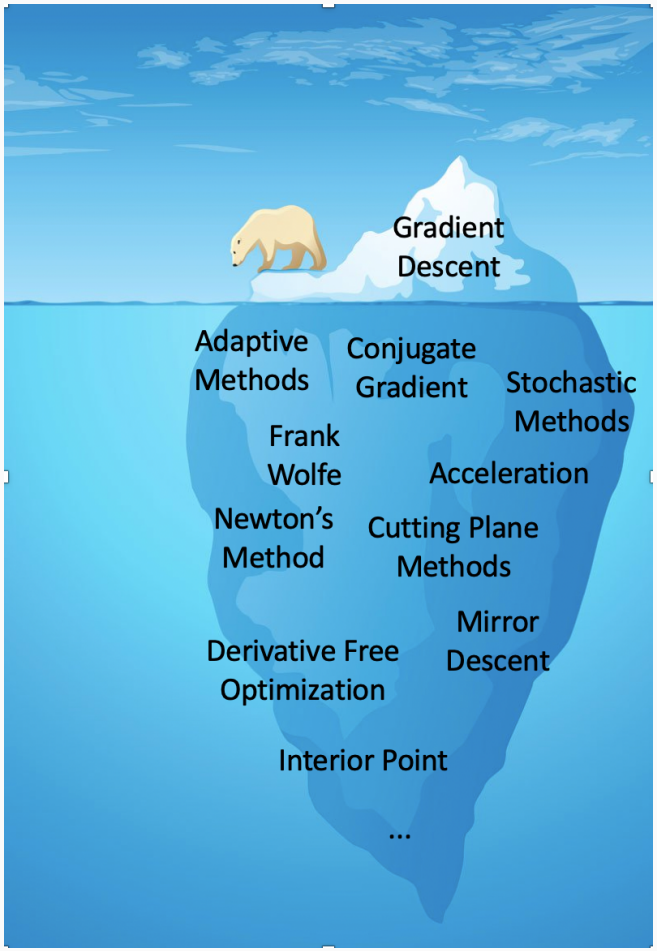
Gradient descent: A greedy search algorithm for minimizing functions of multiple variables (including loss functions) that often works amazingly well.



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 optimization

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.

Gradient descent

Basic Gradient descent algorithm:

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

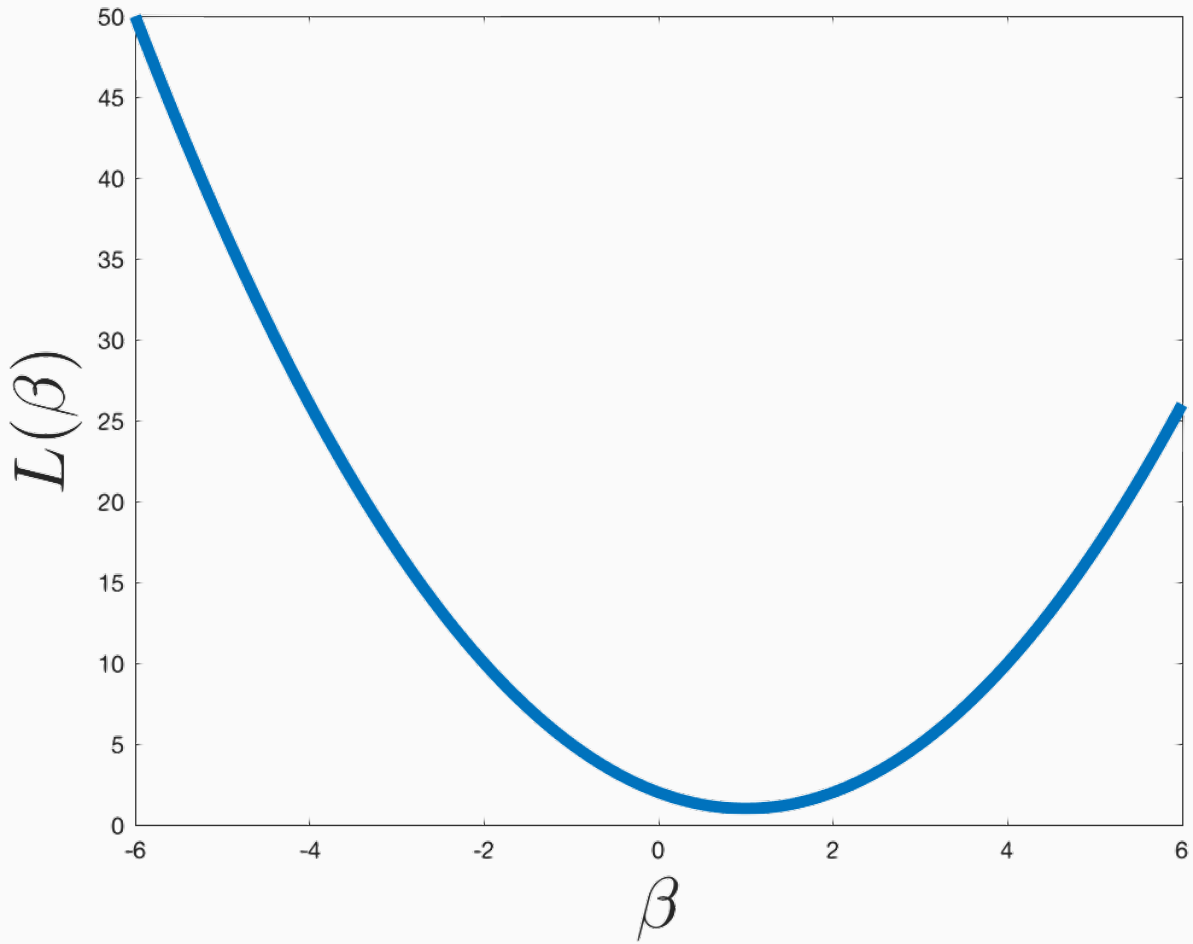
$\eta > 0$ is a step-size parameter. Also called the learning rate.

Why does this method work?

First observation: if we actually reach the minimizer β^* then we stop.

Intuition

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



Gradient descent in 1d

Mathematical way of thinking about it:

By definition, $L'(\beta) = \lim_{\eta \to 0} \frac{L(\beta+t)-L(\beta)}{t}$. So for small values of t, we expect that:

$$L(\beta + t) - L(\beta) \approx t \cdot L'(\beta).$$

We want $L(\beta + t)$ to be smaller than $L(\beta)$, so we want $t \cdot L'(\beta)$ to be negative.

This can be achieved by choosing $t = -\eta \cdot L'(\beta)$.

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

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

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

Directional derivatives

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 $L(\boldsymbol{\beta} + \mathbf{v}) < L(\boldsymbol{\beta})$?

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

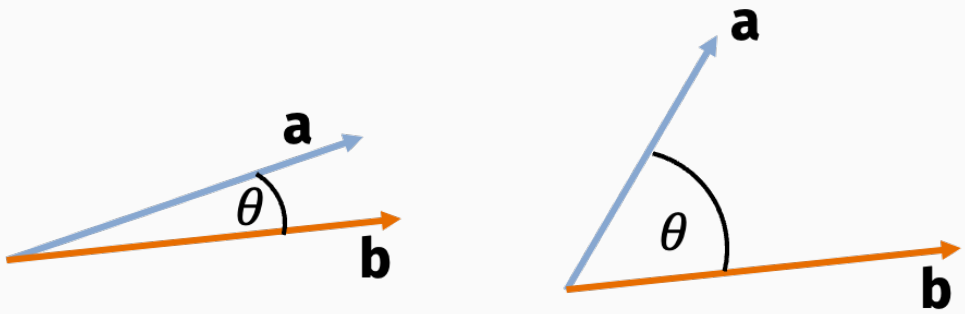
Steepest descent

Claim (Gradient descent = Steepest descent[5])

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

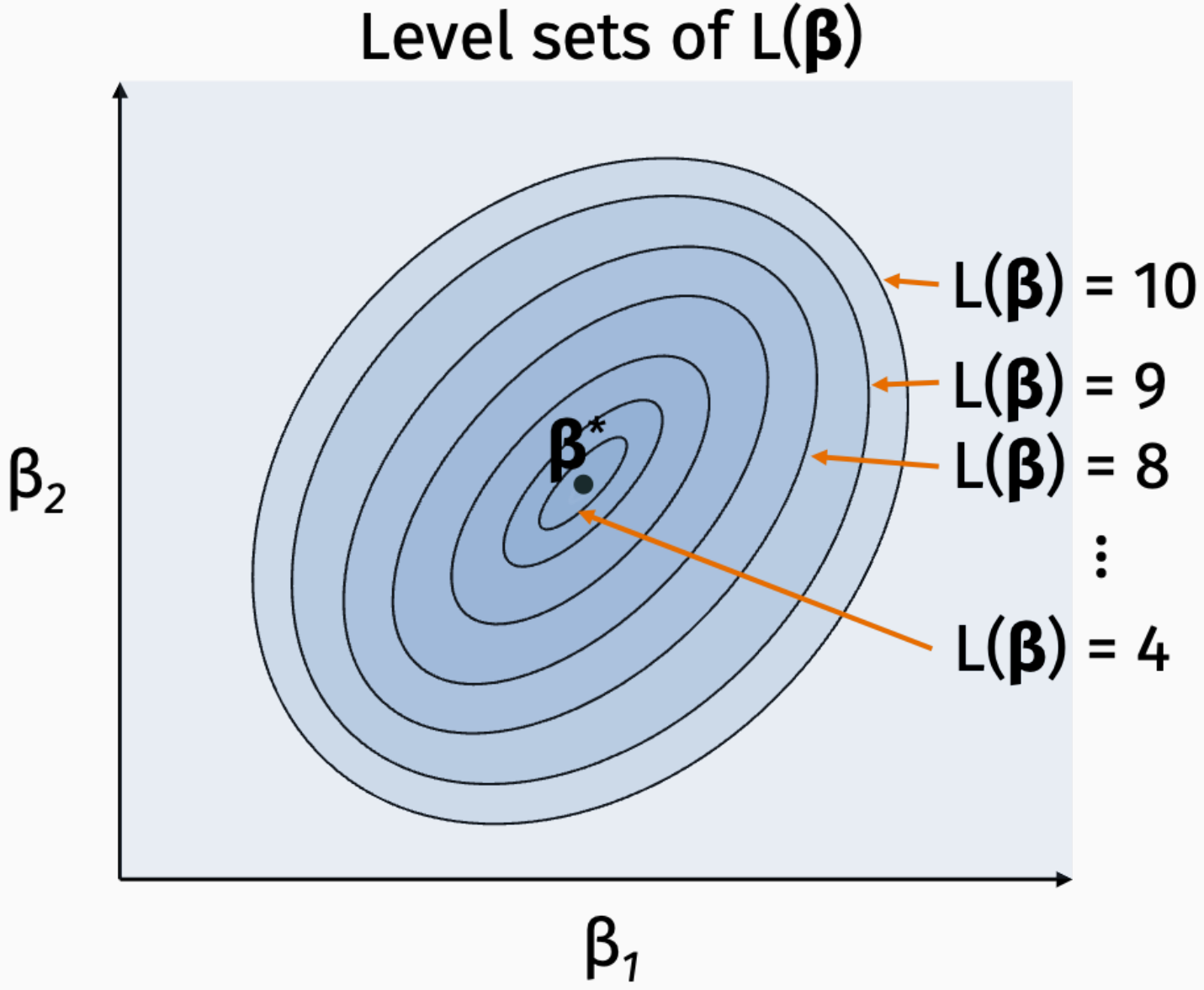
Recall: For two vectors $\mathbf{a}, \mathbf{b}$,

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



[5]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..

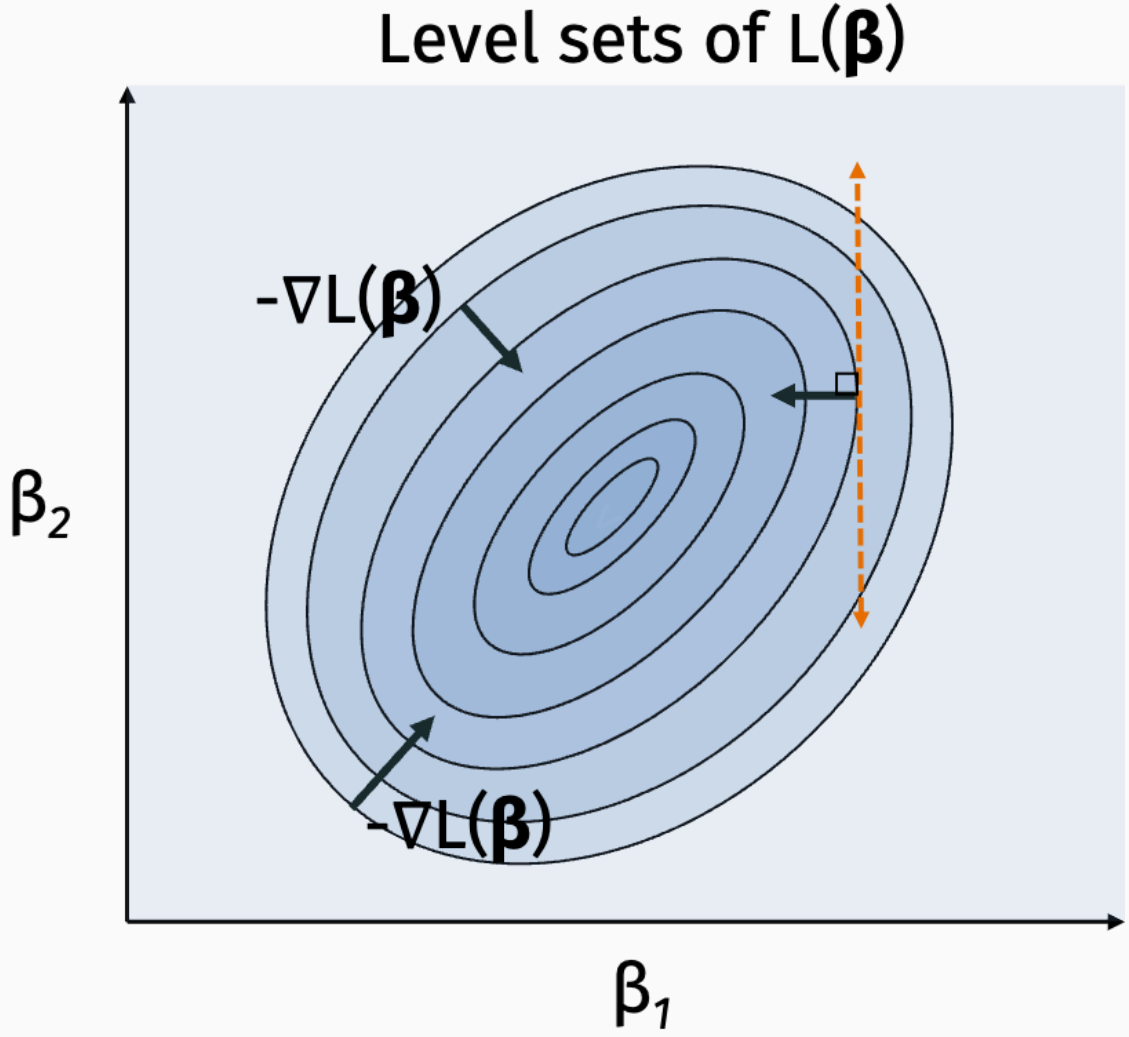
Visualizing in 2d



Steepest descent

Claim (Gradient descent = Steepest descent)

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



Gradient descent

Basic Gradient descent (GD) algorithm:

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

Basic claim

- For sufficiently small η, every step of GD either
  1. Decreases the function value.
  2. Get's stuck because the gradient term equals 0

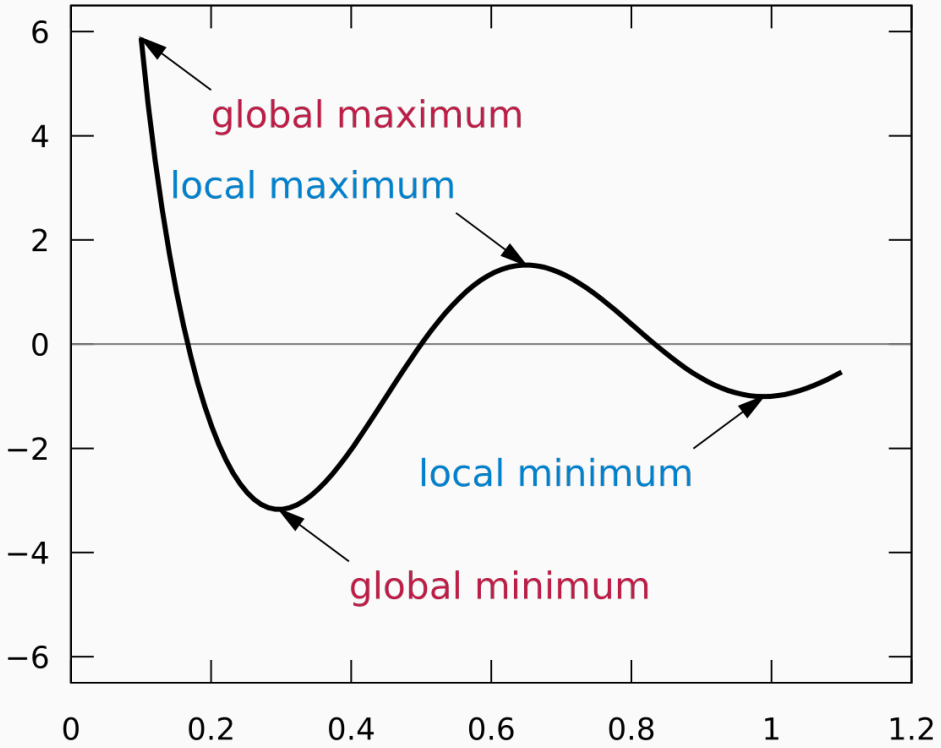
Claim

For sufficiently small η and a sufficiently large number of iterations T, gradient descent will converge to a **local minimum** *or* **stationary point** *of the loss function $\tilde{\boldsymbol{\beta}}^*$. I.e. with*

$$\nabla L(\tilde{\boldsymbol{\beta}}^*) = \mathbf{0}.$$

Basic claim

You can have stationary points that are not minima (local maxima, saddle points). In practice, always converge to local minimum.



Very unlikely to land precisely on another stationary point and get stuck. Non-minimal stationary points are "unstable".

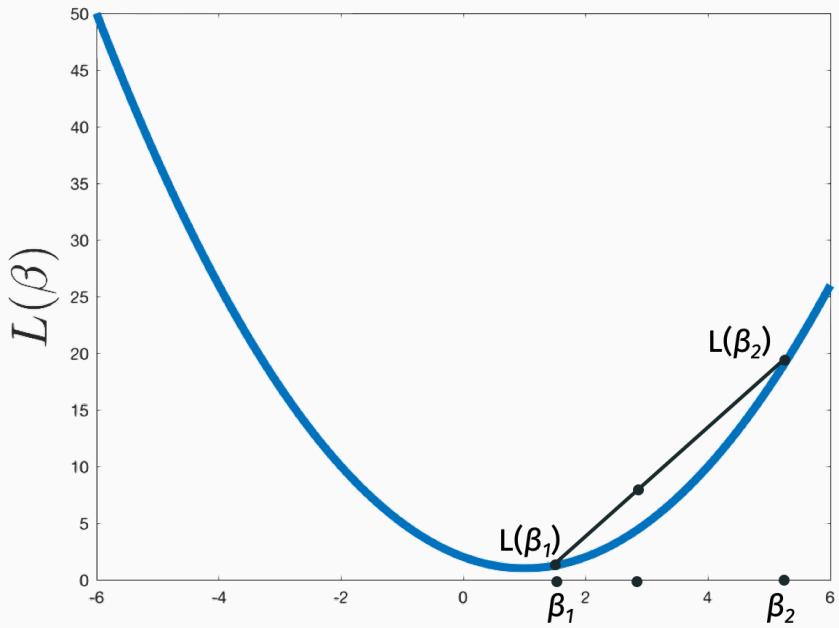
Convex function

For a broad class of functions, GD converges to global minima.

Definition (Convex)

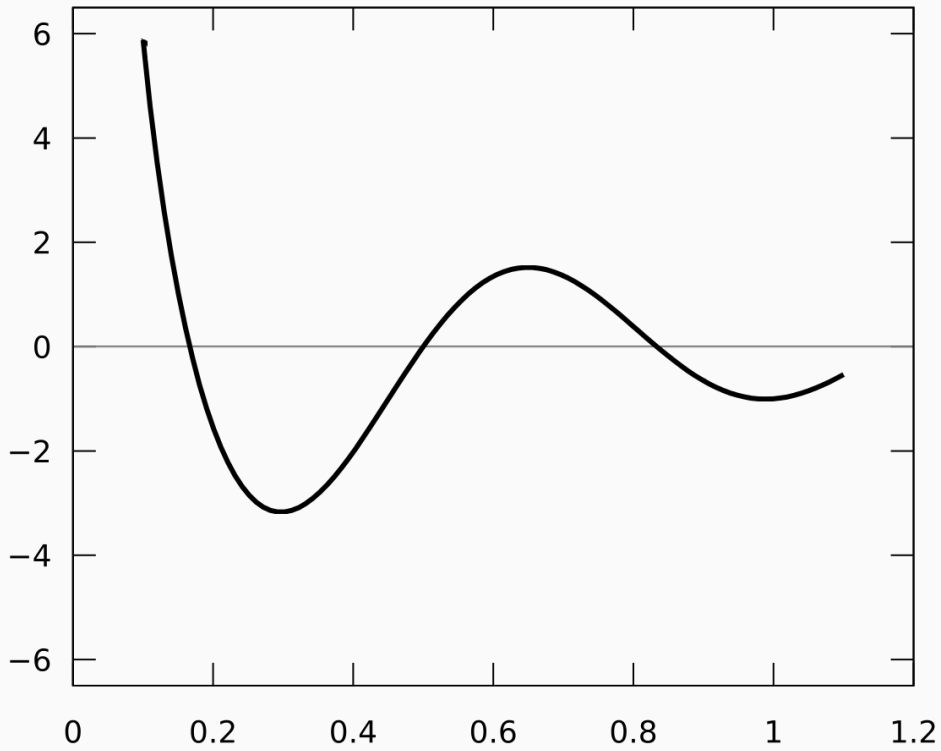
A function L is convex iff for any $\beta_1, \beta_2, \lambda \in [0, 1]$:

$$(1 - \lambda) \cdot L(\beta_1) + \lambda \cdot L(\beta_2) \geq L((1 - \lambda) \cdot \beta_1 + \lambda \cdot \beta_2)$$



Convex function

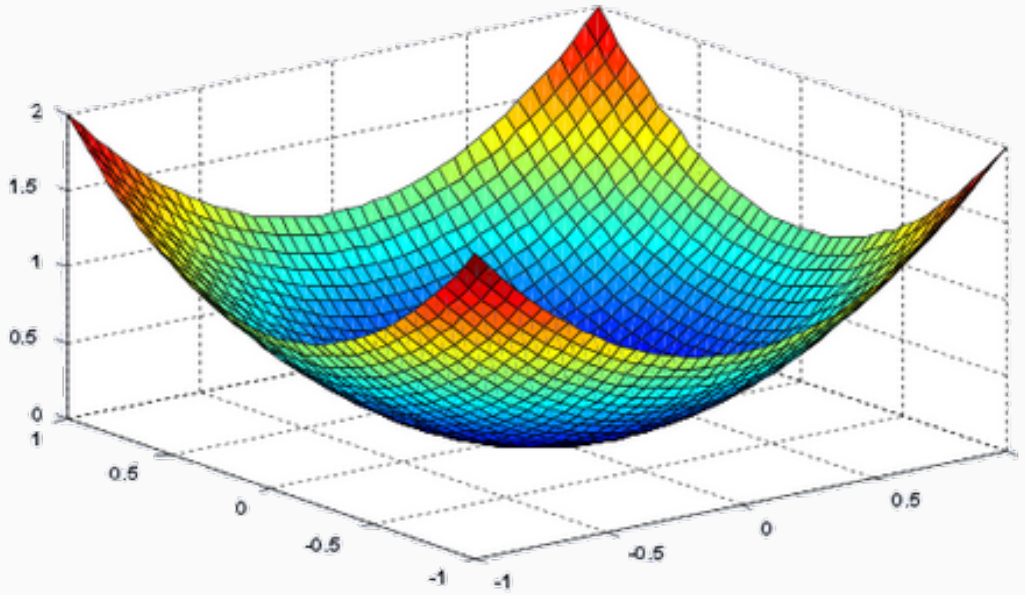
In words: A function is convex if a line between any two points on the function lies above the function. Captures the notion that a function looks like a bowl.



This function **is not** convex.

Convex function

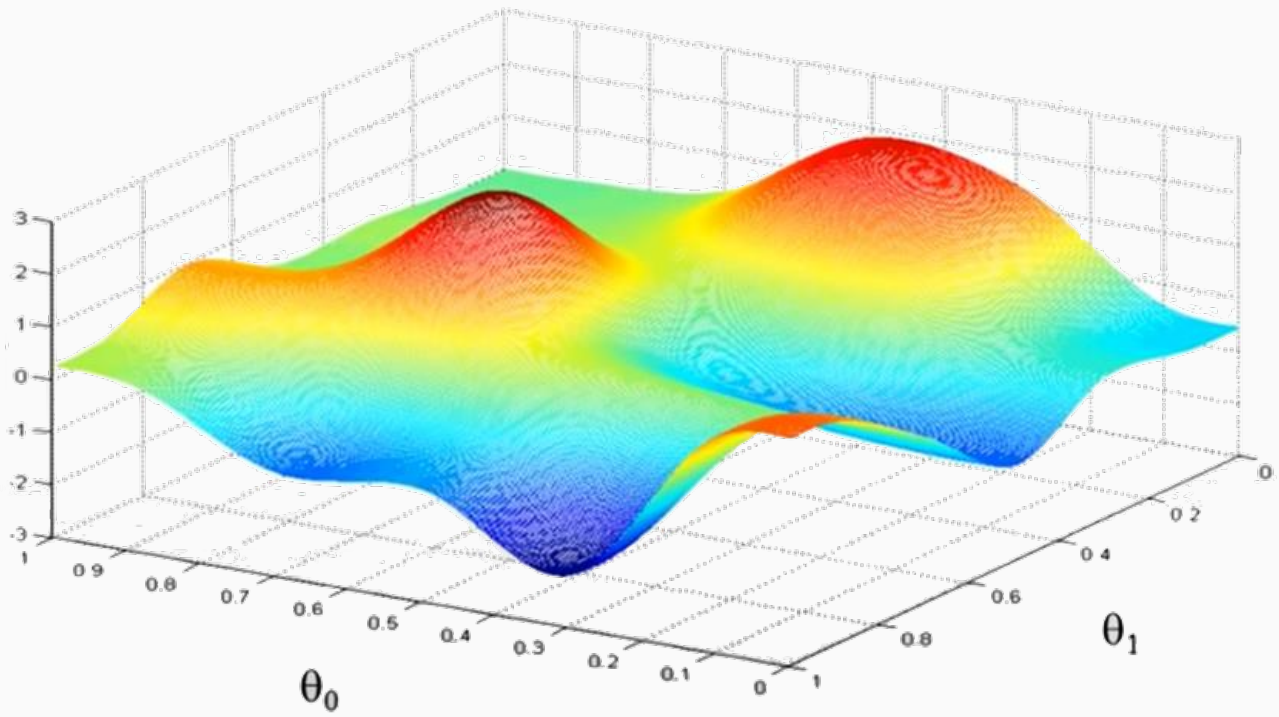
In words: A function is convex if a line between any two points on the function lies above the function. Captures the notion that a function looks like a bowl.



This function **is** convex.

Convex function

In words: A function is convex if a line between any two points on the function lies above the function. Captures the notion that a function looks like a bowl.



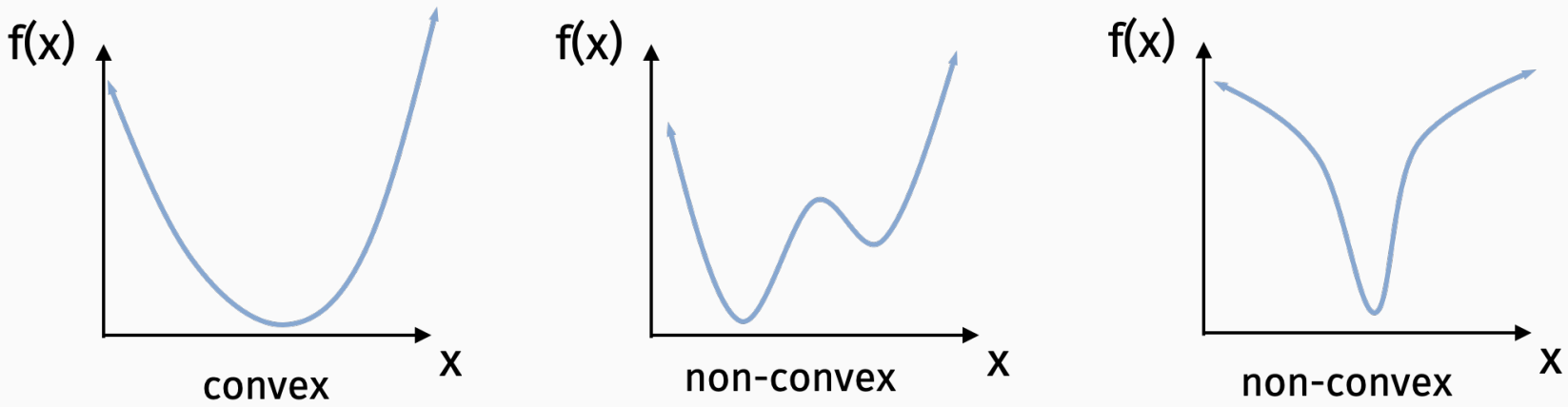
This function **is not** convex.

Convergence of gradient descent

What functions are convex?

- Least squares loss for linear regression.
- ℓ_1 loss for linear regression.
- Either of these with and ℓ_1 or ℓ_2 regularization penalty.
- Logistic regression! Logistic regression with regularization.
- Many other models in machine leaning.

Non-convex



What functions in machine learning are not convex? Loss functions involving neural networks, matrix completion problems, mixture models, many more.

Vary in how "bad" the non-convexity is. For example, some matrix factorization problems are non-convex but still only have global minima.

convexity warm up

Prove that $L(\beta) = \beta^2$ is convex.

To show: For any $\beta_1, \beta_2, \lambda \in [0, 1]$,
$\lambda L(\beta_1) + (1 - \lambda)L(\beta_2) \geq L(\lambda \cdot \beta_1 + (1 - \lambda) \cdot \beta_2)$

Convexity warm up

Prove that $L(\beta) = \beta^2$ is convex.

To show: For any $\beta_1, \beta_2, \lambda \in [0, 1]$,
$\lambda L(\beta_1) + (1 - \lambda) L(\beta_2) \geq L(\lambda \cdot \beta_1 + (1 - \lambda) \cdot \beta_2)$

Convexity of least squares regression loss

Prove that $L(\boldsymbol{\beta}) = \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2$ is convex. For now just consider $\lambda = \frac{1}{2}$ case. The general λ case is similar, but messier.

[5] **Useful identity:** $(a+b)^2 \leq 2(a^2+b^2)$

Convexity of least squares regression loss

Prove that $L(\boldsymbol{\beta}) = \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2$ is convex. I.e. that:

$$\|\mathbf{X}(\lambda\boldsymbol{\beta}_1 + (1-\lambda)\boldsymbol{\beta}_1) - \mathbf{y}\|_2^2 \leq \lambda\|\mathbf{X}\boldsymbol{\beta}_1 - \mathbf{y}\|_2^2 + (1-\lambda)\|\mathbf{X}\boldsymbol{\beta}_2 - \mathbf{y}\|_2^2$$

Rate of convergence for convex functions

We care about how fast gradient descent and related methods converge, not just that they do converge.

- Bounding iteration complexity requires placing some assumptions on $L(\beta)$.
- Stronger assumptions lead to better bounds on the convergence.

Understanding these assumptions can help us design faster variants of gradient descent (there are many!).

Next class: A canonical gradient descent analysis that every computer scientist should know.