Lecture 5: Linear Regression (Part 4) [& Logistic Regression (Part 1)]

CIS 4190/5190

Spring 2023

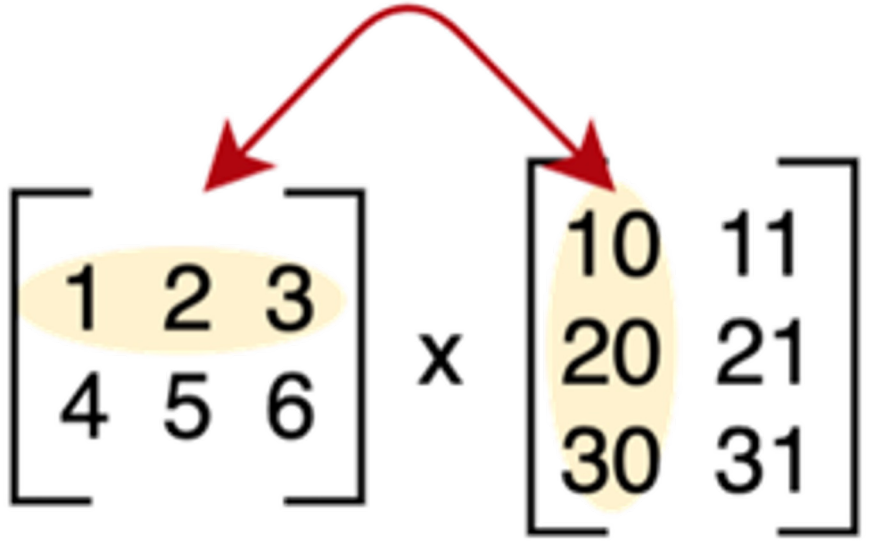
Administrivia

- HW2 in progress!
- Recitation recording on Canvas, for those of you that couldn't attend.
- Quiz due in a few days.
- Helpful Extra Readings (All Optional):
  - Hastie and Tibshirani Ch 3: Linear Regression. https://hastie.su.domains/Papers/ESLII.pdf
  - Hands-On ML Linear Regression Worksheet: https://github.com/ageron/handson-ml/blob/master/04_training_linear_models.ipynb
  - D2l.ai Interactive textbook on ML (Use in Pytorch mode): https://d2l.ai/index.html
  - Scikit-learn "algorithm cheatsheet". A flowchart for "which algorithm to use":
    - https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
  - Recht and Hardt Chapter 1. An introduction to the history of machine learning: https://mlstory.org/introduction.html
  - Leo Breiman, "Statistical Modeling: The Two Cultures"
    - http://www2.math.uu.se/~thulin/mm/breiman.pdf

Last class

- Cross-Validation
- Optimizing the unregularized linear regression loss (Train MSE)
  - Closed-form solution
  - Iterative optimization through gradient descent

  - Q: How do their computational complexities compare?
    - Note: Multiplying AxB matrix with BxC matrix = O(ABC)



Relative computational complexities

- Optimizing the unregularized linear regression loss (Train MSE)
  - Closed-form solution

$$\hat{\beta}(Z) = (X^\top X)^{-1} X^\top Y$$

$O(D^2 N)$ $O(D^3)$ $O(D^2 N + D^3)$

  - Iterative optimization through gradient descent:

$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)$$

$$\nabla_\beta L(\beta_t; Z) = \frac{2}{n} X^\top \underbrace{(\underbrace{X\beta}_{O(ND)} - Y)}_{O(ND)}$$

Repeating T times, O(NDT)

Suppose N is very large. Can we get a way from linear dependence on N in the gradient update?

For two matrices of dimensions A×B and B×C, multiplication is O(ABC)

Recall: MSE gradient as mean of sample-wise gradients

(Equivalent to our earlier matrix expression of gradient)

- By linearity of the gradient, we have:

$$\nabla_\beta L(\beta; Z) = \sum_{i=1}^{n} \nabla_\beta (y_i - \beta^\top x_i)^2 = \frac{1}{n} \sum_{i=1}^{n} 2(y_i - \beta^\top x_i) x_i$$

Gradient w.r.t a single sample O(D)

What if we just used the single-sample gradient of a randomly drawn sample as a noisy approximation to the mean of gradients?

“Stochastic” Gradient Descent

Batch Gradient Descent

Initialize β

Repeat T times till covergence {

$$\beta_j \leftarrow \beta_j - \alpha \sum_{i=1}^{N} 2(y_i - \beta^\top x_i) x_i \quad \forall d$$

}

We are descending the original loss function $L(\beta; Z)$.

O(NDT)

Stochastic Gradient Descent

Initialize β

Randomly shuffle dataset (T’ is typically 1 – 10xT)

Repeat T’ times until covergence {

For *i* = 1...*N*, do

$$\beta_j \leftarrow \beta_j - \alpha 2(y_i - \beta^\top x_i) x_i \quad \forall d$$

}

At each step, we are descending a different loss function specific to the chosen sample $L(\beta; Z_i = \{(x_i, y_i)\})$.

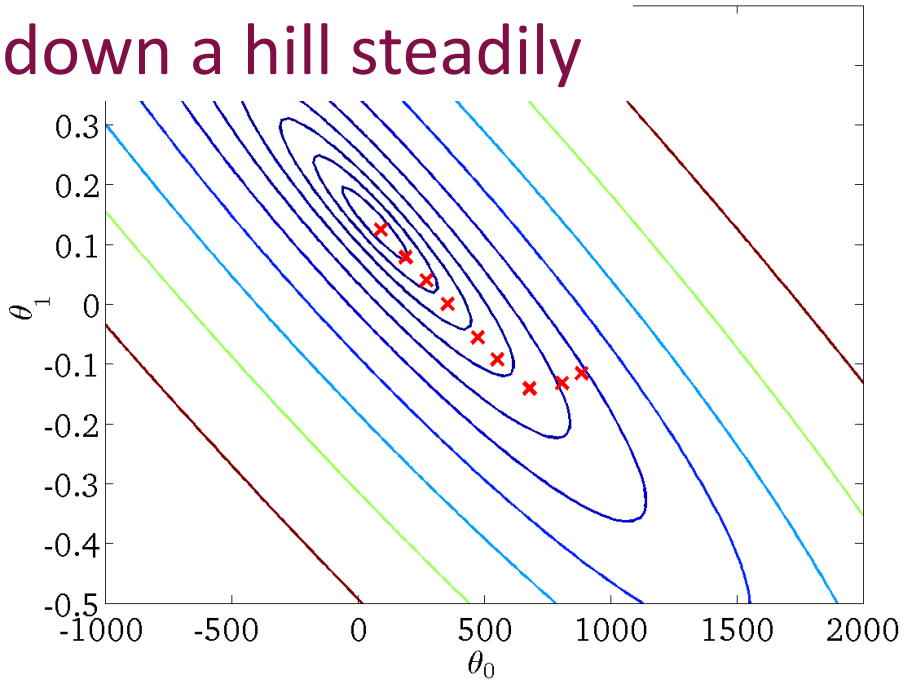
O(DT’)

We will look later at methods to speed up convergence so that T (or T') can be small above.

Noisy Gradients in Stochastic Gradient Descent

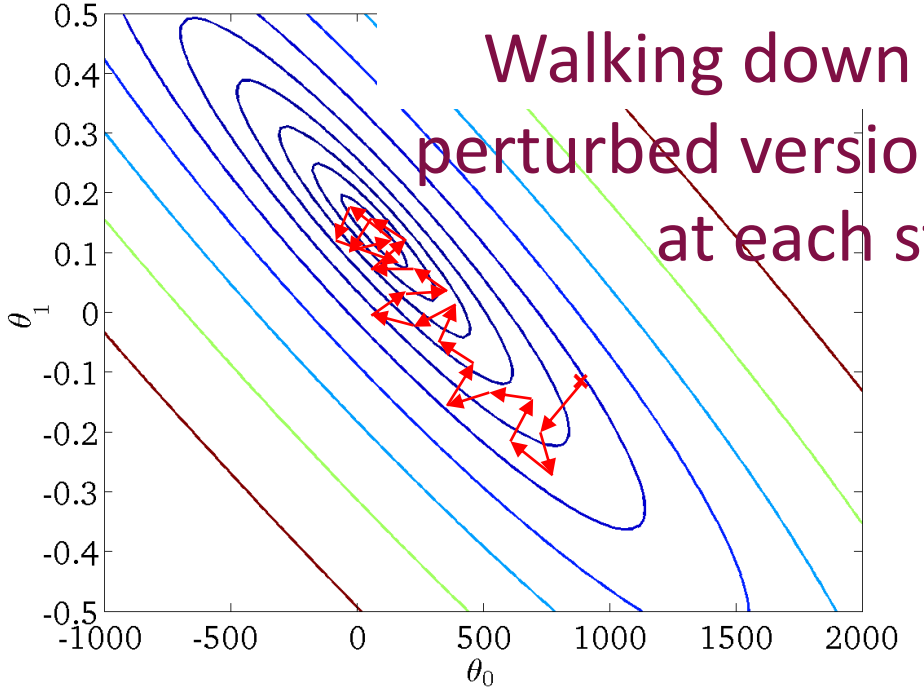
Full Dataset / "Batch" GD

Walking down a hill steadily



Stochastic GD

Walking down a slightly perturbed version of the hill at each step



- Learning rate α is typically held constant
- One heuristic is to decrease α over time to force $\boldsymbol{\theta}$ to converge: $\alpha_t = \frac{constant1}{iterationNumber\ t + constant2}$
- We'll encounter more sophisticated strategies soon!

Based on slide by Andrew Ng

Optimized *Regularized* Linear Regression

- For optimizing L2-regularized linear regression, the same two methods hold as before!
  - Closed form
  - Gradient descent

L_2 Regularized Linear Regression

- Recall that linear regression with L_2 regularization minimizes the loss

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^{d} \beta_j^2$$

L_2 Regularized Linear Regression

- Recall that linear regression with L_2 regularization minimizes the loss

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda\sum_{j=1}^{d}\beta_j^2 = \frac{1}{n}\|Y - X\beta\|_2^2 + \lambda\|\beta\|_2^2$$

- Gradient is

$$\nabla_\beta L(\beta; Z) = -\frac{2}{n}X^\top Y + \frac{2}{n}X^\top X\beta + 2\lambda\beta$$

Strategy 1: Closed-Form Solution

- Gradient is (from last slide)

$$\nabla_\beta L(\beta; Z) = -\frac{2}{n} X^\top Y + \frac{2}{n} X^\top X\beta + 2\lambda\beta$$

- Setting $\nabla_\beta L\left(\hat{\beta}; Z\right) = 0$, we have $(X^\top X + n\lambda I)\hat{\beta} = X^\top Y$

- Always invertible if $\lambda > 0$, so we have

$$\hat{\beta}(Z) = (X^\top X + n\lambda I)^{-1} X^\top Y$$

Strategy 2: Gradient Descent

- Gradient is (from last slide)

$$\nabla_\beta L(\beta; Z) = -\frac{2}{n} X^\top Y + \frac{2}{n} X^\top X\beta + 2\lambda\beta$$

- Same algorithm as vanilla linear regression (a.k.a. OLS)
- **Intuition:** The extra term $\lambda\beta$ in the gradient is often interpreted as a "**weight decay**" --- at every update, it decays $\beta \to \beta(1 - 2\alpha\lambda)$

Feature Preprocessing And Selection

Invariance To The Scales of Input Features

- Consider performing linear regression with a dataset that has height measurements as one of the input dimensions.
  - Does it matter what units height should be recorded in?
    - E.g. Inches? Centimers?
  - Those two measurements are simply scaled versions of each other, and contain exactly the same information.

- We would like our ML algorithms to not get affected in any substantive / meaningful way by such simple scalings: **“Scale Invariance”**

Scale Invariance in Unregularized Linear Regression

- The unregularized linear regression problem is:

$$\underset{\beta}{\operatorname{argmin}}\ L(\beta, Z) = \frac{1}{n}\sum_{i=1}(y_i - \beta^T x)^2$$

Does linearly scaling some dimensions in the data affect the solution?

- Suppose one data dimension x_j is scaled by some factor k
- For an *equivalent* β that produces *identical* predictions $\beta^T x$, we need only scale β_j by $\frac{1}{k}$, and leave all other parameters $\beta_{\neq j}$ the same as before.
- Crucially, the loss value for this β would also be the same, since it depends on x and β exclusively through the predictions $\beta^T x$
- The new argmin would thus select *equivalent* β that produces *identical* predictions $\beta^T x$. Unregularized linear regression is scale invariant!

Scale Invariance in *Regularized* Linear Regression

- The regularized linear regression problem is:

$$\underset{\beta}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda\left(\beta_2^2 + \cdots + \beta_j^2 + \cdots + \beta_d^2\right)$$

How would linearly scaling some dimensions in the data affect the solution?

- Suppose one data dimension is scaled by some factor k
- For an *equivalent* β that produces *identical* predictions $\beta^T x$, we need only scale β_j by $\frac{1}{k}$, and leave all other parameters $\beta_{\neq j}$ the same as before.

- **But this no longer leaves the loss unaffected!** The regularizer term depends directly on the parameter scale β.
  - This means that we cannot guarantee that the argmin above will select an equivalent solution.

Solution: Feature Standardization

- Rescale all features to zero mean and unit variance

$$x_{i,j} \leftarrow \frac{x_{i,j} - \mu_j^{train}}{\sigma_j^{train}} \qquad \mu_j^{train} = \frac{1}{N}\sum_{i=1}^{N} x_{i,j} \qquad \sigma_j^{train} = \frac{1}{N}\sum_{i=1}^{N}\left(x_{i,j} - \mu_j^{train}\right)^2$$

  - **Note:** When using intercept term, do not rescale $x_1 = 1$
  - Can be sensitive to outlier data samples (ways to fix this later in course)

- **Must use same transformation during training and for prediction**
  - Compute the standardization means μ_j^{train} and standard deviation σ_j^{train} on training data and use the same values on test data too:

$$x_{i,j}^{test} \leftarrow \frac{x_{i,j}^{test} - \mu_j^{train}}{\sigma_j^{train}}$$

Preprocessing Beyond Feature Standardization

- Standardization is one example of a “dataset preprocessing” step.
- Another is the implementation of the “feature map”: $x \rightarrow \phi(x)$

- Let us look at some more in the context of a specific problem.

Housing Dataset

- Sales of residential property in Ames, Iowa from 2006 to 2010
  - **Examples:** 1,022
  - **Features:** 79 total (real-valued + categorical), some are missing!
  - **Label**: Sales price

Common way to present data. Rows=samples, columns=features, last column=label.

MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	...	MoSold	YrSold	SaleType	SaleCondition	SalePrice
20	RL	80.0	10400	Pave	NaN	Reg	...	5	2008	WD	Normal	174000
180	RM	35.0	3675	Pave	NaN	Reg	...	5	2006	WD	Normal	145000
60	FV	72.0	8640	Pave	NaN	Reg	...	6	2010	Con	Normal	215200
20	RL	84.0	11670	Pave	NaN	IR1	...	3	2007	WD	Normal	320000
60	RL	43.0	10667	Pave	NaN	IR2	...	4	2009	ConLw	Normal	212000
80	RL	82.0	9020	Pave	NaN	Reg	...	6	2008	WD	Normal	168500
60	RL	70.0	11218	Pave	NaN	Reg	...	5	2010	WD	Normal	189000
80	RL	85.0	13825	Pave	NaN	Reg	...	12	2008	WD	Normal	140000
60	RL	NaN	13031	Pave	NaN	IR2	...	7	2006	WD	Normal	187500

Data from: De Cock. Journal of Statistics Education 19(3), 2011

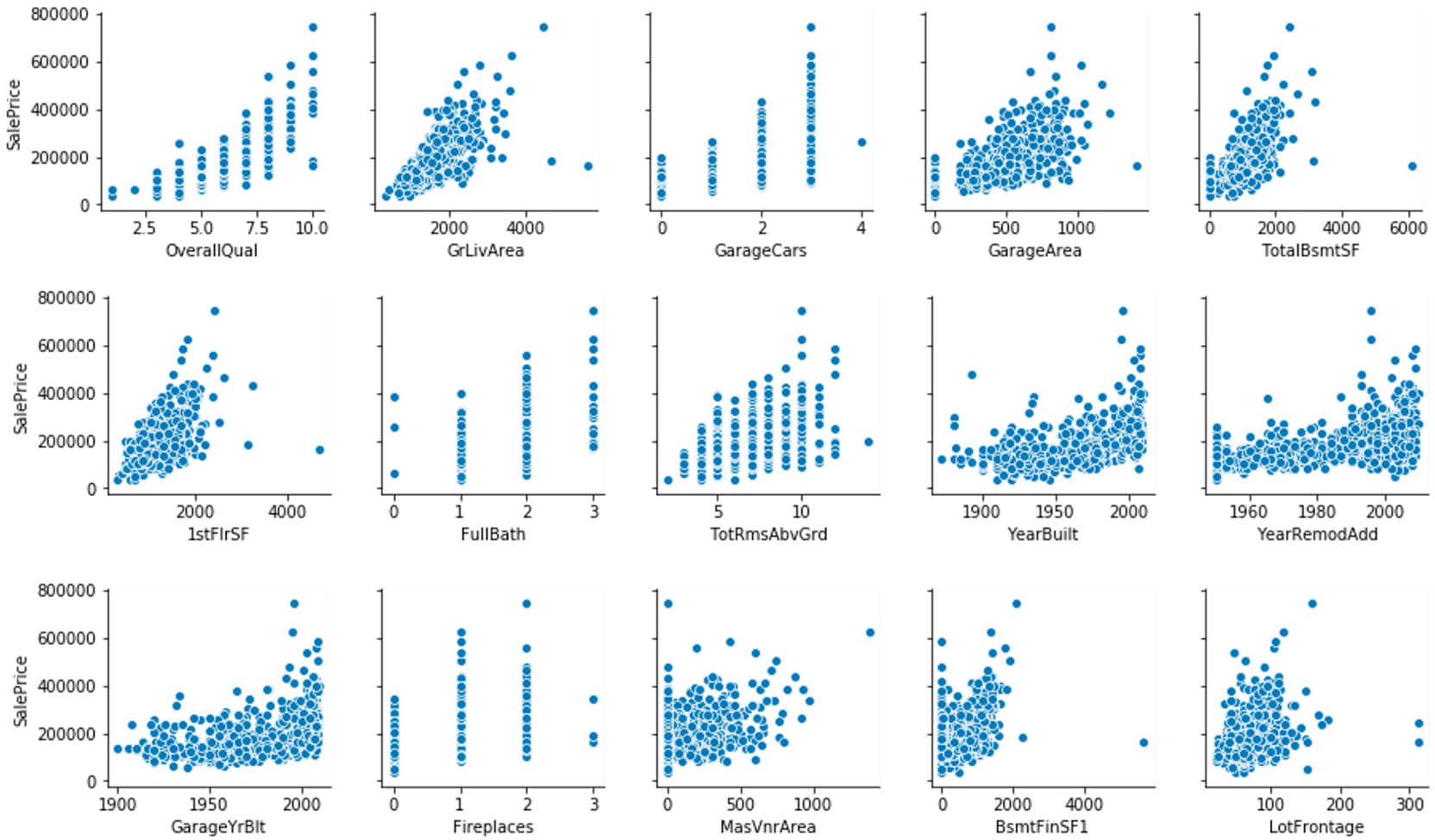
Housing Dataset

- dataframe.describe()

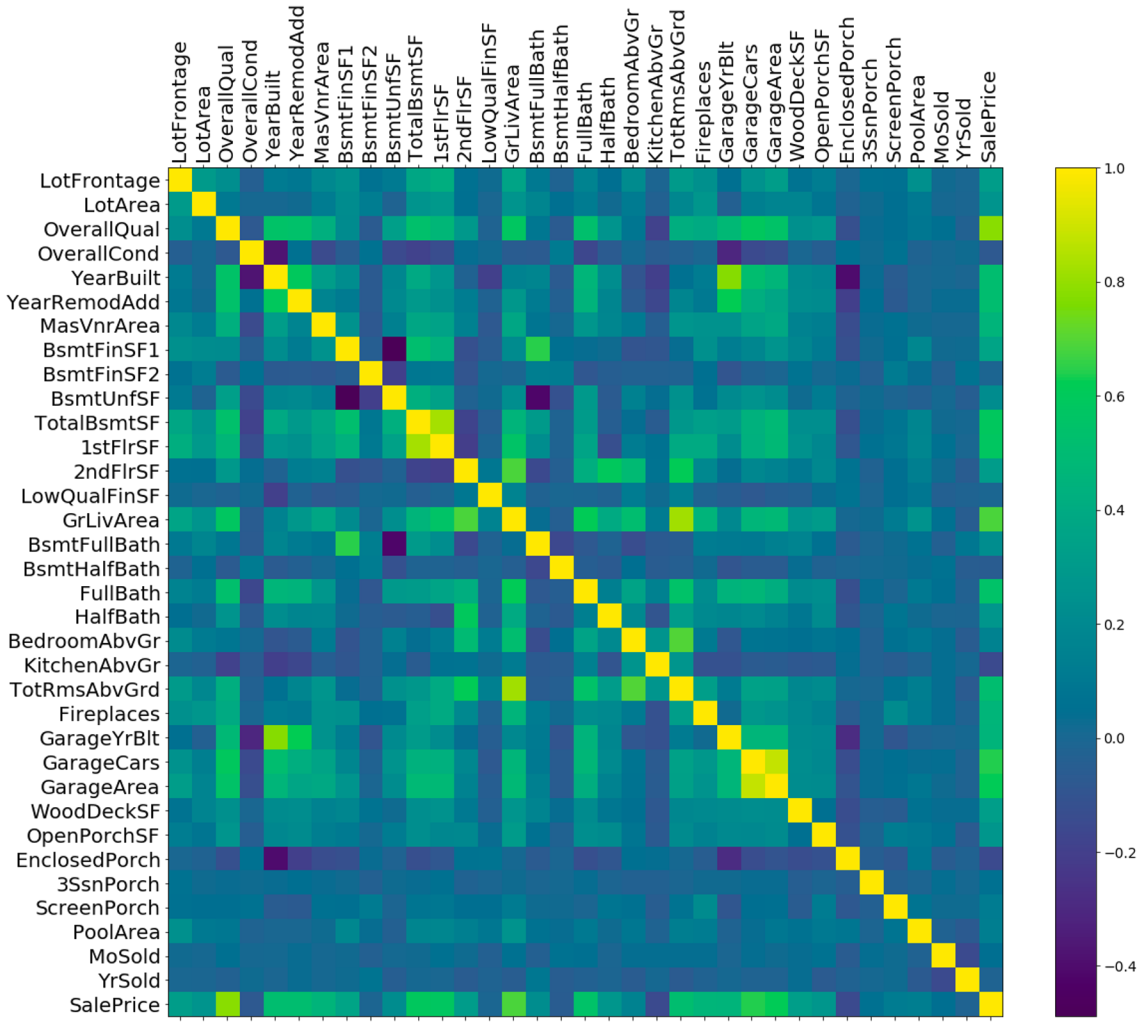
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	...	SalePrice
count	1022.000000	1022.000000	832.000000	1022.000000	1022.000000	1022.000000	1022.000000	1022.000000	1019.000000		1022.000000
mean	732.338552	57.059687	70.375000	10745.437378	6.128180	5.564579	1970.995108	1984.757339	105.261040		181312.692759
std	425.860402	42.669715	25.533607	11329.753423	1.371391	1.110557	30.748816	20.747109	172.707705		77617.461005
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	...	34900.000000
25%	367.500000	20.000000	59.000000	7564.250000	5.000000	5.000000	1953.000000	1966.000000	0.000000		130000.000000
50%	735.500000	50.000000	70.000000	9600.000000	6.000000	5.000000	1972.000000	1994.000000	0.000000		165000.000000
75%	1100.500000	70.000000	80.000000	11692.500000	7.000000	6.000000	2001.000000	2004.000000	170.000000		215000.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1378.000000		745000.000000

Features Most Correlated with Label

To control the size of the hypothesis class, could select features most correlated with label, such as these:



Feature Correlation Matrix Visualization



Could get rid of features that are heavily correlated with each other, to reduce redundancy.

Handling Missing Values

- If rarely missing, could discard such samples during training.
- If very common for some feature to be missing, omit the feature from the model entirely.

- Other possible ways to handle missing values
  - **Numerical:** Impute with mean
  - **Categorical:** Impute with mode

Feature	% Missing Values
PoolQC	99.5108
MiscFeature	96.0861
Alley	93.5421
Fence	80.2348
FireplaceQu	47.6517
LotFrontage	18.5910
GarageCond	05.2838
GarageType	05.2838
GarageYrBlt	05.2838
GarageFinish	05.2838
GarageQual	05.2838
BsmtFinType1	02.5440
…	

Recall: Other Preprocessing Steps We Have Seen

Converting all data to be numeric type:

- **Categorical:** Featurize using “one-hot encoding” vectors e.g. cat =[1, 0], dog =[0, 1]
- **Ordinal**
  - Convert to integer (e.g., low, medium, high → 1, 2, 3)
  - Does not fully capture relationships (try different featurizations!)

HouseStyle	FullBath	RoofMatl	BsmtCond	KitchenQual
1Story	2	CompShg	TA	TA
SLvl	1	CompShg	TA	TA
2Story	2	CompShg	TA	Gd
1Story	2	CompShg	Gd	Ex
2Story	2	CompShg	TA	Gd
SLvl	1	WdShngl	TA	TA
2Story	2	CompShg	TA	Gd
SLvl	1	CompShg	TA	TA
2Story	2	CompShg	TA	TA
2Story	2	CompShg	TA	Gd

HouseStyle	FullBath	RoofMatl	BsmtCond	KitchenQual
1Story	2	CompShg	3	3
SLvl	1	CompShg	3	3
2Story	2	CompShg	3	4
1Story	2	CompShg	4	5
2Story	2	CompShg	3	4
SLvl	1	WdShngl	3	3
2Story	2	CompShg	3	4
SLvl	1	CompShg	3	3
2Story	2	CompShg	3	3
2Story	2	CompShg	3	4

Automatic Feature Set Selection with L1 Regularization

~~L_0 Regularization~~ $\rightarrow$ L_1 Regularization

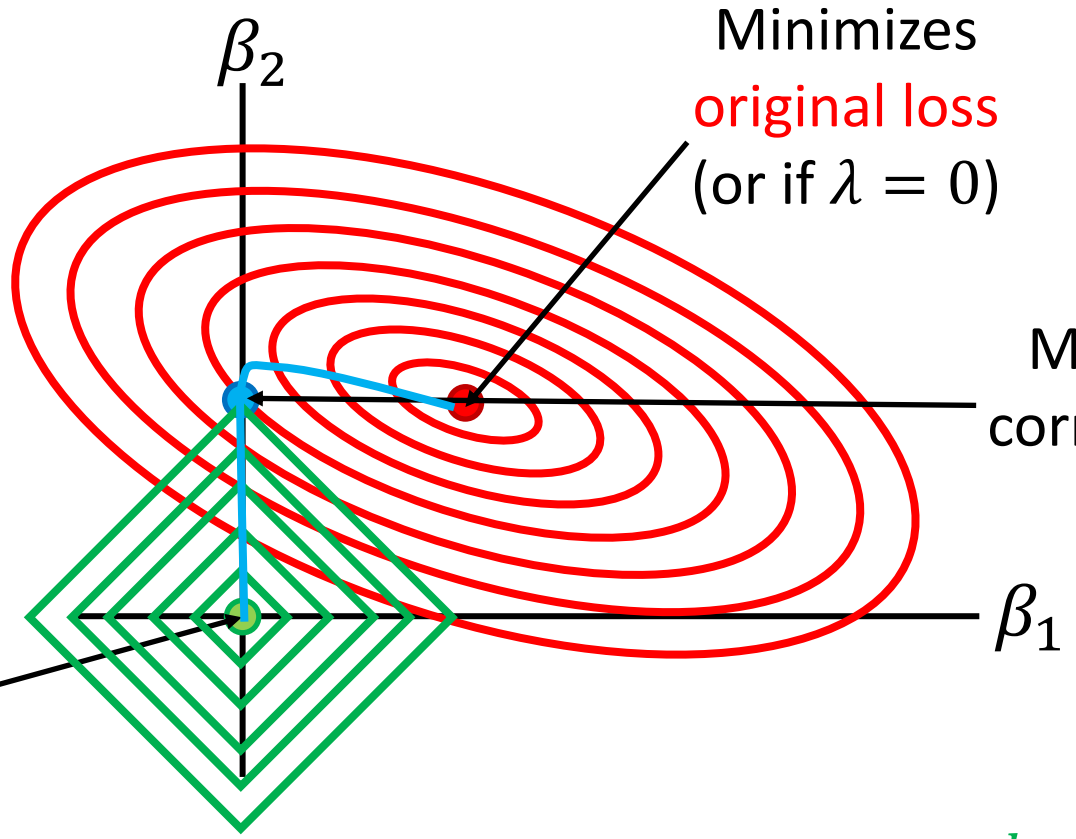
$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda||\beta||_0$$

- **Sparsity:** Can we minimize $\|\beta\|_0 = \left|\{ j \mid \beta_j \neq 0 \}\right|$, the number of non-zero components? (This is called L_0 regularization)
  - Automatic **feature selection**!
  - Improves interpretability.

- **Challenge:** $\|\beta\|_0$ is not differentiable, making it hard to optimize

- **Solution:** L_1 Regularization
  - We can instead use an L_1 norm $\|\beta\|_1$ as the regularizer!
  - Still harder to optimize than L_2 norm, but at least it is convex

Intuition on L_1 Regularization



Minimizes original loss (or if $\lambda = 0$)

Minimizer of full loss at corner → sparse ($\beta_1 = 0$)!

Minimizes regularization term (or if $\lambda \to \infty$)

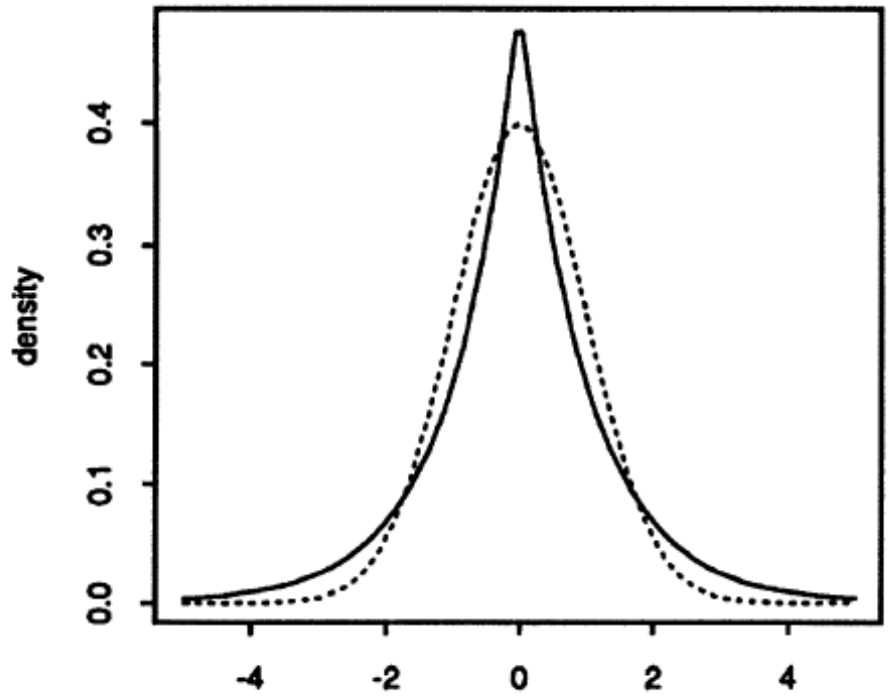
$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^{d}|\beta_j|$$

Intuition on L_1 Regularization

Recall: L2 regularization induces a gaussian prior on the values of β.

L1 regularization similarly induces a "Laplacian" prior on the values of β.

- More parameters drawn to 0.
- But also more parameters with large values, compared to L2.



--- L2 regularization

__ L1 regularization

L_1 Regularization for Feature Selection

- **Step 1:** Construct a lot of features and add to feature map
- **Step 2:** Use L_1 regularized regression to "select" subset of features
  - I.e., coefficient $\beta_j \neq 0$ → feature j is selected)
- **Optional:** Remove unselected features from the feature map and run vanilla linear regression (a.k.a. ordinary least squares)

Optimizing L_1 Regularized Linear Regression?

- Gradient descent still works!

- Specialized algorithms work better in practice
  - **Simple one:** Gradient descent + soft thresholding
  - Basically, if $\left|\beta_{t,j}\right| \leq \lambda$, just set it to zero
  - Good theoretical properties

Lecture 5: Logistic Regression (Part 1)

CIS 4190/5190

Spring 2023

Supervised Learning

Data $Z = \{(x_i, y_i)\}_{i=1}^{n}$

$\hat{\beta}(Z) = \arg\min_{\beta} L(\beta; Z)$

L encodes $y_i \approx f_{\beta}(x_i)$

Model $f_{\hat{\beta}(Z)}$

Regression

Data $Z = \{(x_i, y_i)\}_{i=1}^{n}$

$\hat{\beta}(Z) = \arg\min_{\beta} L(\beta; Z)$

L encodes $y_i \approx f_{\beta}(x_i)$

Model $f_{\hat{\beta}(Z)}$

Label is a **real value** $y_i \in \mathbb{R}$

Classification

Data $Z = \{(x_i, y_i)\}_{i=1}^{n}$

$\hat{\beta}(Z) = \arg\min_{\beta} L(\beta; Z)$

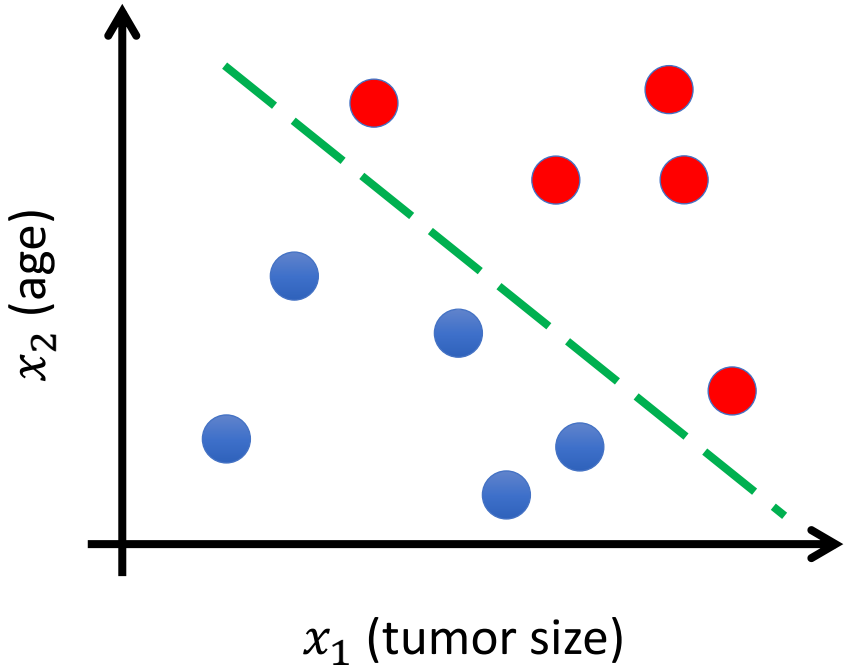
L encodes $y_i \approx f_{\beta}(x_i)$

Model $f_{\hat{\beta}(Z)}$

Label is a **discrete value** $y_i \in \mathcal{Y} = \{1, \ldots, k\}$

(Binary) Classification

- **Input:** Dataset $Z = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- **Output:** Model $y_i \approx f_\beta(x_i)$



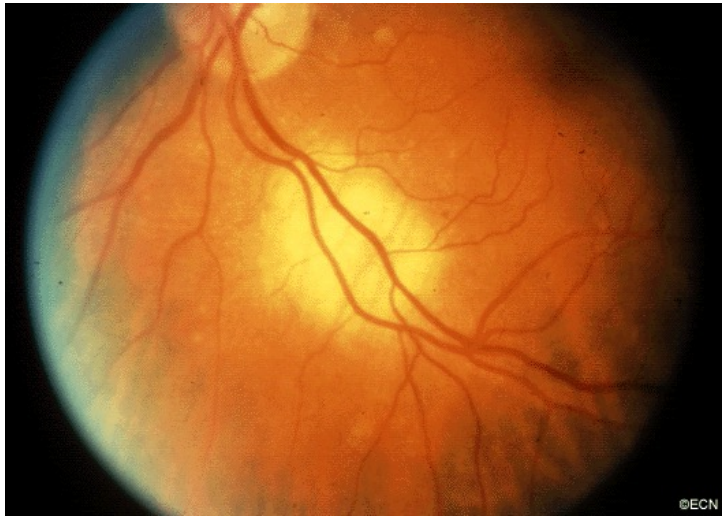


Image: https://eyecancer.com/uncategorized/choroidal-metastasis-test/

Example: Malignant vs. Benign Ocular Tumor

Loss Minimization View of ML

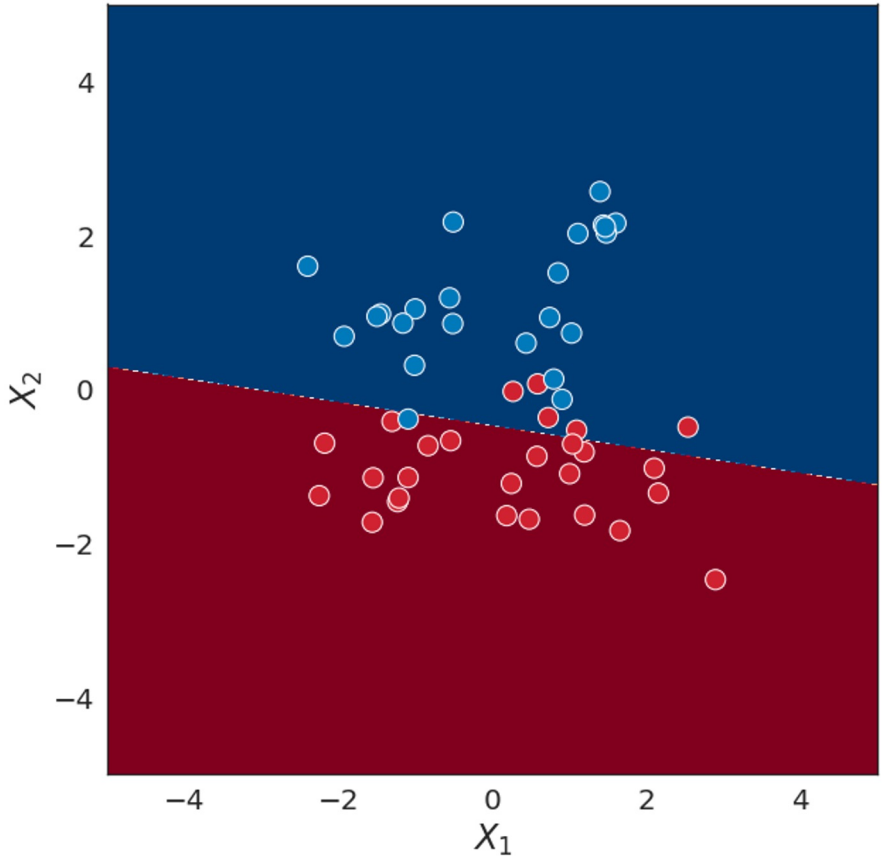
- **Three design decisions**
  - **Model family:** What are the candidate models f? (E.g., linear functions)
  - **Loss function:** How to define "approximating"? (E.g., MSE loss)
  - **Optimizer:** How do we optimize the loss? (E.g., gradient descent)

- How do we adapt to classification?

Linear Functions for (Binary) Classification

- **Input:** Dataset $Z = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

- **Classification:**
  - Labels $y_i \in \{0, 1\}$
  - Predict $y_i \approx 1(\beta^\top x_i \geq 0)$
  - $1(C)$ equals 1 if C is true and 0 if C is false
  - How to learn β? **Need a loss function!**

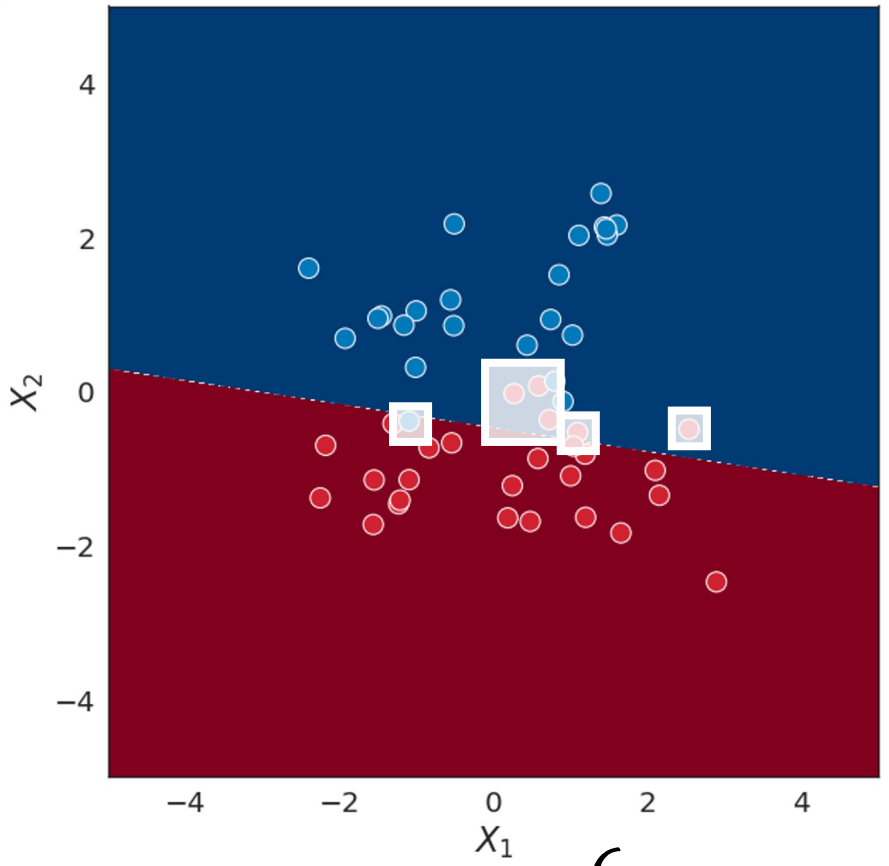


Loss Functions for Linear Classifiers

- **(In)accuracy:**

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n} 1\left(y_i \neq f_\beta(x_i)\right)$$

- Computationally intractable
- Often, but not always the "true" loss (e.g., imbalanced data)



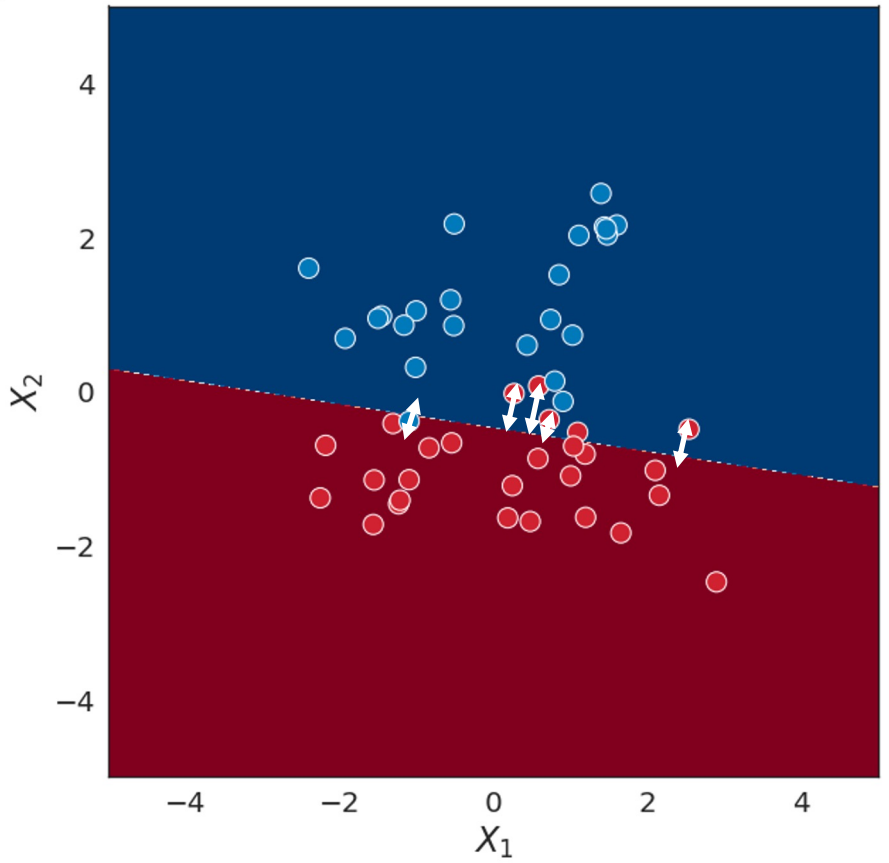
$$L(\beta; Z) = \frac{6}{50}$$

Loss Functions for Linear Classifiers

- **Distance:**

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n} \text{dist}(x_i, f_\beta) \cdot 1\left(f_\beta(x_i) \neq y_i\right)$$

- If $L(\beta; Z) = 0$, then 100% accuracy
- Variant of this loss results in SVM
- We consider a more general strategy



$$L(\beta; Z) = 1.2$$

Maximum Likelihood Estimation

- A **probabilistic** viewpoint on learning (from statistics)
- Given x_i, **suppose** y_i is drawn i.i.d. from distribution $p_{Y|X}(Y = y \mid x; \beta)$ with parameters β (or density, if y_i is continuous):

$$y_i \sim p_{Y|X}(\cdot \mid x_i; \beta)$$

Y is random variable, not vector

- Typically write $p_\beta(Y = y \mid x)$ or just $p_\beta(y \mid x)$
  - Called a **model** (and $\{p_\beta\}_\beta$ is the **model family**)
  - Will show up convert p_β to f_β later

Maximum Likelihood Estimation

- **Compare to loss function minimization:**
  - **Before:** $y_i \approx f_\beta(x_i)$
  - **Now:** $y_i \sim p_\beta(\cdot | x_i; \beta)$

- **Intuition the difference:**
  - $f_\beta(x_i)$ just provides a point that y_i should be close to
  - $p_\beta(\cdot | x_i; \beta)$ provides a score for each possible y_i

- Maximum likelihood estimation combines the **loss function** and **model family** design decisions

Maximum Likelihood Estimation

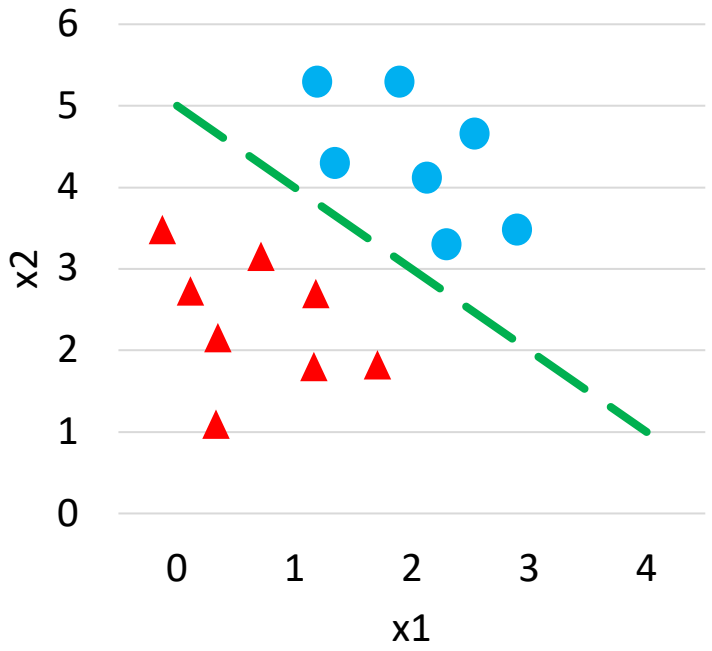
- **Likelihood:** Given model p_β, the probability of dataset Z (replaces loss function in loss minimization view):

$$L(\beta; Z) = p_\beta(Y \mid X) = \prod_{i=1}^{n} p_\beta(y_i \mid x_i)$$

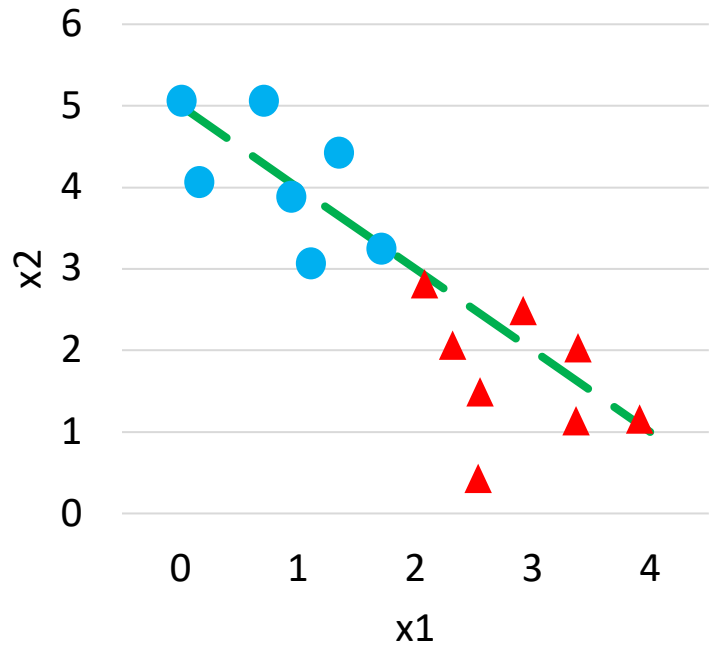
- **Negative Log-likelihood (NLL):** Computationally better behaved form:

$$\ell(\beta; Z) = -\log L(\beta; Z) = -\sum_{i=1}^{n} \log p_\beta(y_i \mid x_i)$$

Intuition on the Likelihood



High likelihood
(Low NLL)



Low likelihood
(High NLL)

Example: Linear Regression

- Assume that the conditional density is

$$p_\beta(y_i \mid x_i) = N(y_i; \beta^\top x_i, 1) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{(\beta^\top x_i - y_i)^2}{2}}$$

- $N(y; \mu, \sigma^2)$ is the density of the normal (a.k.a. Gaussian) distribution with mean μ and variance σ^2

Example: Linear Regression

- Then, the likelihood is

$$L(\beta; Z) = \prod_{i=1}^{n} p_\beta(y_i \mid x_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{(\beta^\top x_i - y_i)^2}{2}}$$

- The NLL is

$$\ell(\beta; Z) = -\sum_{i=1}^{n} \log p_\beta(y_i \mid x_i) = \underbrace{\frac{n \log(2\pi)}{2}}_{\text{constant}} + \underbrace{\sum_{i=1}^{n} (\beta^\top x_i - y_i)^2}_{\text{MSE!}}$$

Example: Linear Regression

- Loss minimization for maximum likelihood estimation:

$$\hat{\beta}(Z) = \arg\min_{\beta} \ell(\beta; Z)$$

- **Note:** Called maximum likelihood estimation since maximizing the likelihood equivalent to minimizing the NLL

Example: Linear Regression

- What about the model family?

$$f_\beta(x) = \arg\max_y p_\beta(y \mid x)$$

$$= \arg\max_y \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{\|\beta^\top x - y\|_2^2}{2}}$$

$$= \beta^\top x$$

- **Recovers linear functions!**

Loss Minimization View of ML

- **Three design decisions**
  - **Model family:** What are the candidate models f? (E.g., linear functions)
  - **Loss function:** How to define "approximating"? (E.g., MSE loss)
  - **Optimizer:** How do we optimize the loss? (E.g., gradient descent)

Maximum Likelihood View of ML

- **Two design decisions**
  - **Likelihood:** Probability $p_\beta(\, y \mid x \,)$ of data (x, y) given parameters β
  - **Optimizer:** How do we optimize the NLL? (E.g., gradient descent)

- **Corresponding Loss Minimization View:**
  - **Model family:** Most likely label $f_\beta(x) = \arg\max_y p_\beta(\, y \mid x \,)$
  - **Loss function:** Negative log likelihood (NLL) $\ell(\beta; Z) = -\sum_{i=1}^{n} \log p_\beta(\, y_i \mid x_i \,)$

- Very powerful framework for designing cutting edge ML algorithms
  - Write down the “right” likelihood, form tractable approximation if needed
  - Especially useful for thinking about non-i.i.d. data