Welcome to
Lab1: Linear Models

Date: 04.02.2026

Who I am: Ceren Gok - PhD student working on Continual Learning and Lecturer for MLE course 2026!

Welcome to
Lab1: Linear Models

Date: 04.02.2026

Who I am: Ceren Gok - PhD student working on Continual Learning and Lecturer for MLE course 2026!

What is these Labs are about: Guided-coding exercises, Try to remember what we learned from the lectures, Sharing our opinions with our peers, aaannd competing with fun way !

Welcome to
Lab1: Linear Models

Date: 04.02.2026

Who I am: Ceren Gok - PhD student working on Continual Learning and Lecturer for MLE course 2026!

What is these Labs are about: Guided-coding exercises, Try to remember what we learned from the lectures, Sharing our opinions with our peers, aaannd competing with fun way !

The Structure of Slides: Historical Facts, Revisiting Theories, Interview Questions, Live Coding Exercise for you.

Welcome to
Lab1: Linear Models

Intended Learning Outcomes:

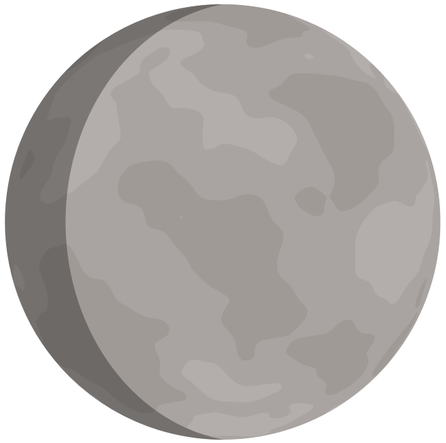
- Remember the fundamental approaches to solve linear models
- Implement these solutions together
- Observe the issue of overfitting
- Apply regularization for overfitting
- Analyze and interpret the model

Welcome to
Lab1: Linear Models

Lets start with a Game and see how much we remember from lecture notes!

Disclosure: We will have small awards throughout the session.

Historical Fact: First Regression problem in 1800s

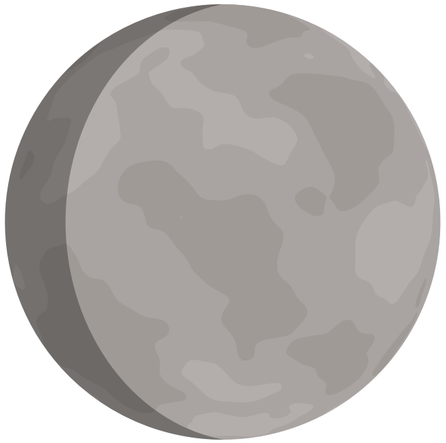


Ceres: dwarf planet

Ceres was discovered but quickly "lost" as it passed behind the sun.

Astronomers had only 40 days of observation data about time and location of the planet.The problem was to find the orbit of the Ceres and where it will reappear again.

Historical Fact: First Regression problem in 1800s



Ceres: dwarf planet

Ceres was discovered but quickly "lost" as it passed behind the sun.

Astronomers had only 40 days of observation data about time and location of the planet.The problem was to find the orbit of the Ceres and where it will reappear again.

24-year-old Carl Gauss used the method of Least Squares to find the orbit of Ceres with limited data points and predicted where it will reappear again based on its orbit.

This allowed astronomers to find Ceres exactly where he predicted.

Linear Models

regression

predicting exact stock price

Linear Models

regression

predicting exact stock price

classification

predicting whether increase/decrease

Formal Definition: Linear models make prediction using linear function of the input X by minimizing the loss of predicted values and actual observations.

Feature1	Feature2	Feature3	Real Outcome	Predicted Outcome
123	34	Yes	15	14
156	52	No	16	16
178	78	Yes	12	11
198	99	Yes	15	13
201	23	No	14	18
141	24	Yes	18	15

Formal Definition: Linear models make prediction using linear function of the input X by minimizing the loss of predicted values and actual observations.

Feature1	Feature2	Feature3	Real Outcome	Predicted Outcome
123	34	Yes	15	14
156	52	No	16	16
178	78	Yes	12	11
198	99	Yes	15	13
201	23	No	14	18
141	24	Yes	18	15

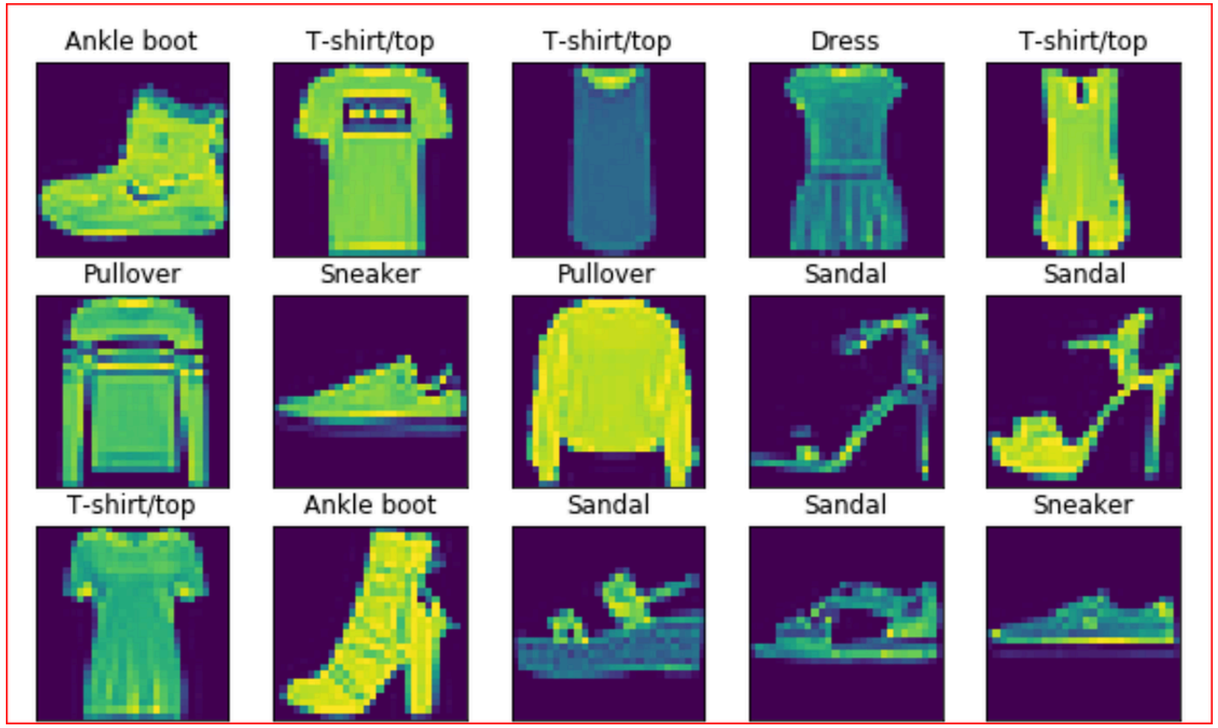
If we speak from the modelling side: we would like to find optimum weights (w) that minimizes the loss function.

$$\underset{\mathbf{w}}{\operatorname{argmin}} \mathcal{L}(f_{\mathbf{w}}(X))$$

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{p} w_i \cdot x_i + w_0$$

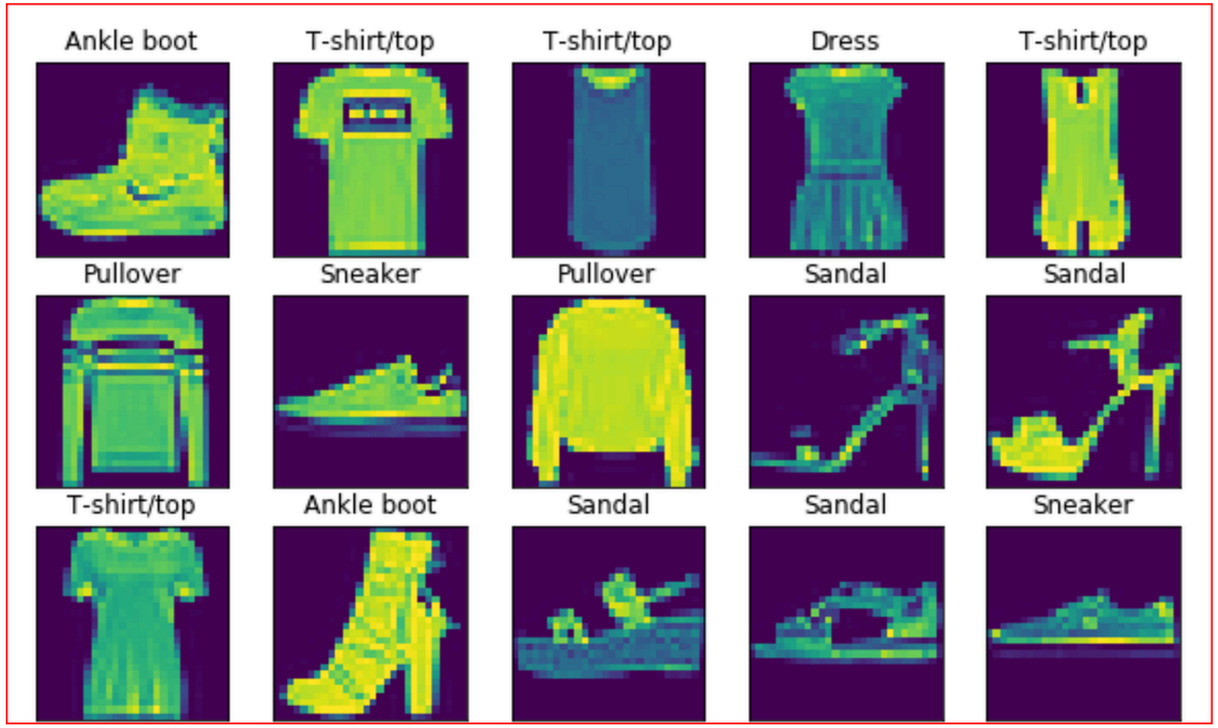
Our Mission Today: Classification

Dataset: FashionMNIST



Our Mission Today: Classification

Dataset: FashionMNIST



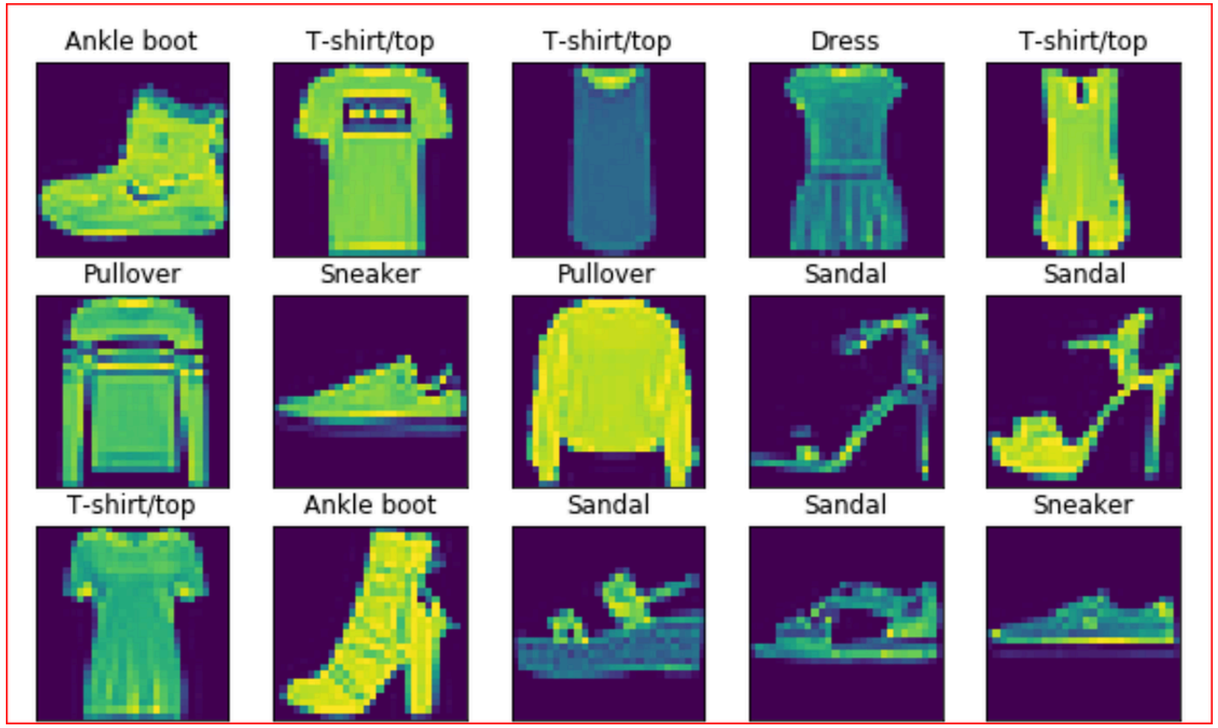
Who: Zalando Research

Why: MNIST was too easy

What: 70,000 images with 28x28 pixels

Our Mission Today: Classification

Dataset: FashionMNIST



Who: Zalando Research

Why: MNIST was too easy

What: 70,000 images with 28x28 pixels

OpenML Platform: It is an open-science platform for sharing machine learning datasets, tasks, and experiment results.***

The fundamental approach to solve convex optimization problem: Closed-form approach

$$w^* = (X^T X)^{-1} X^T Y$$

The fundamental approach to solve convex optimization problem: Closed-form approach

$$w^* = (X^T X)^{-1} X^T Y$$

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

The fundamental approach to solve convex optimization problem: Closed-form approach

$$w^* = (X^T X)^{-1} X^T Y$$

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

Binarize y: $y_{bin0} = [1, 0, 0]^T$

The fundamental approach to solve convex optimization problem: Closed-form approach

$$w^* = (X^T X)^{-1} X^T Y$$

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

Binarize y: $y_{bin0} = [1, 0, 0]^T$

$$X^T X = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

The fundamental approach to solve convex optimization problem: Closed-form approach

$$w^* = (X^T X)^{-1} X^T Y$$

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

Binarize y: $y_{bin0} = [1, 0, 0]^T$

$$X^T X = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$(X^T X)^{-1} = \frac{1}{(2)(2) - (1)(1)} \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 0.67 & -0.33 \\ -0.33 & 0.67 \end{bmatrix}$$

The fundamental approach to solve convex optimization problem: Closed-form approach

$$w^* = (X^T X)^{-1} X^T Y$$

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

Binarize y: $y_{bin0} = [1, 0, 0]^T$

$$X^T X = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$(X^T X)^{-1} = \frac{1}{(2)(2) - (1)(1)} \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 0.67 & -0.33 \\ -0.33 & 0.67 \end{bmatrix}$$

Calculate

$$X^T y_{bin0}: \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} [1, 0, 0]^T = [\mathbf{1}, \mathbf{0}]^{\mathbf{T}}$$

The fundamental approach to solve convex optimization problem: Closed-form approach

$$w^* = (X^T X)^{-1} X^T Y$$

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

Binarize y: $y_{bin0} = [1, 0, 0]^T$

$$X^T X = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$(X^T X)^{-1} = \frac{1}{(2)(2) - (1)(1)} \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 0.67 & -0.33 \\ -0.33 & 0.67 \end{bmatrix}$$

Calculate

$$X^T y_{bin0}: \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} [1, 0, 0]^T = [\mathbf{1}, \mathbf{0}]^{\mathbf{T}}$$

Solve

$$w_0: \begin{bmatrix} 0.67 & -0.33 \\ -0.33 & 0.67 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = [\mathbf{0.67}, \mathbf{-0.33}]^{\mathbf{T}}$$

The fundamental approach to solve convex optimization problem: Closed-form approach

$$w^* = (X^T X)^{-1} X^T Y$$

Repeat for other samples and finalize W*

$$W = \begin{bmatrix} 0.67 & 0.33 & -0.33 \\ -0.33 & 0.33 & 0.67 \end{bmatrix}$$

The fundamental approach to solve convex optimization problem: Closed-form approach

$$w^* = (X^T X)^{-1} X^T Y$$

Repeat for other samples and finalize W*

$$W = \begin{bmatrix} 0.67 & 0.33 & -0.33 \\ -0.33 & 0.33 & 0.67 \end{bmatrix}$$

Now we need to calculate scores to make prediction

$$XW = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.67 & 0.33 & -0.33 \\ -0.33 & 0.33 & 0.67 \end{bmatrix} = \begin{bmatrix} 0.67 & 0.33 & -0.33 \\ 0.33 & 0.67 & 0.33 \\ -0.33 & 0.33 & 0.67 \end{bmatrix}$$

The fundamental approach to solve convex optimization problem: Closed-form approach

$$w^* = (X^T X)^{-1} X^T Y$$

Repeat for other samples and finalize W*

$$W = \begin{bmatrix} 0.67 & 0.33 & -0.33 \\ -0.33 & 0.33 & 0.67 \end{bmatrix}$$

Now we need to calculate scores to make prediction

$$XW = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.67 & 0.33 & -0.33 \\ -0.33 & 0.33 & 0.67 \end{bmatrix} = \begin{bmatrix} 0.67 & 0.33 & -0.33 \\ 0.33 & 0.67 & 0.33 \\ -0.33 & 0.33 & 0.67 \end{bmatrix}$$

Sample	Score Class 0	Score Class 1	Score Class 2	Predicted (Argmax)	Actual Label
Sample 1	**0.67**	0.33	-0.33	**0**	**0**
Sample 2	0.33	**0.67**	0.33	**1**	**1**
Sample 3	-0.33	0.33	**0.67**	**2**	**2**

***

Interview Moment: Closed-form solution in actual life production

You show a closed-form equation to your engineering manager.

Your dataset has **n = 50,000 features** and **m = 10,000,000 samples.** You're training on a single machine with 64GB RAM.

She says '**Great! Ship it to production.**'

But then your data scientist says '**Wait, we should use gradient descent instead.**'

Your manager is confused: '**Why use an iterative approximation when we can reach the exact answer?**'

What do you tell her?"

Interview Moment: Closed-form solution in actual life production

Matrix inversion is expensive.

X^TX would be 50,000 × 50,000 which is huge, and inverting it takes $O(n^3)$ operations.

That's $50{,}000^3$ = 125 trillion operations, which is too slow.

Gradient descent is faster because it's an iterative approach

Good Answer

Interview Moment: Closed-form solution in actual life production

Matrix inversion is expensive.

X^TX would be 50,000 × 50,000 which is huge, and inverting it takes $O(n^3)$ operations.

That's $50{,}000^3$ = 125 trillion operations, which is too slow.

Gradient descent is faster because it's an iterative approach

Good Answer

Closed-form isn't feasible here for two reasons.
Memory: We need 4TB to store X, but we only have 64GB.
Closed-form requires loading all data at once that's impossible.

Second, even if we had the RAM: **computing and inverting X^T X would take many hours**.

This isn't about closed-form being 'bad', it's about scale, with smaller n(features) closed-form is actually faster and simpler but with
n = 50,000, **we're** well **into gradient descent territory.**

My recommendation is to use mini-batch SGD. I**t's the standard for this scale**, it's what every production system uses.

FAANG-level Answer

Let's talk about **Gradient Descent** - not as a backup plan, but as the way modern ML actually works at scale.

$$\mathbf{w}^{s+1} = \mathbf{w}^{s} - \eta \nabla \mathcal{L}(\mathbf{w}^{s}) \xrightarrow{\text{OR}} W_{new} = W_{init} - (\eta \cdot grad_i)$$

Let's talk about **Gradient Descent** - not as a backup plan, but as the way modern ML actually works at scale.

$$\mathbf{w}^{s+1} = \mathbf{w}^{s} - \eta \nabla \mathcal{L}(\mathbf{w}^{s}) \xrightarrow{\text{OR}} W_{new} = W_{init} - (\eta \cdot grad_i)$$

We have the training inputs X and ground truth y

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

Let's talk about **Gradient Descent** - not as a backup plan, but as the way modern ML actually works at scale.

$$\mathbf{w}^{s+1} = \mathbf{w}^{s} - \eta \nabla \mathcal{L}(\mathbf{w}^{s}) \xrightarrow{\text{OR}} W_{new} = W_{init} - (\eta \cdot grad_i)$$

We have the training inputs X and ground truth y

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

We initialize the weight matrix randomly (as a first step)

$$W_{init} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Let's talk about **Gradient Descent** - not as a backup plan, but as the way modern ML actually works at scale.

$$\mathbf{w}^{s+1} = \mathbf{w}^{s} - \eta \nabla \mathcal{L}(\mathbf{w}^{s}) \xrightarrow{\text{OR}} W_{new} = W_{init} - (\eta \cdot grad_i)$$

We have the training inputs X and ground truth y

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

We pick a sample

$$x_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

We initialize the weight matrix randomly (as a first step)

$$W_{init} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Let's talk about **Gradient Descent** - not as a backup plan, but as the way modern ML actually works at scale.

$$\mathbf{w}^{s+1} = \mathbf{w}^{s} - \eta \nabla \mathcal{L}(\mathbf{w}^{s}) \xrightarrow{\text{OR}} W_{new} = W_{init} - (\eta \cdot grad_i)$$

We have the training inputs X and ground truth y

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

We pick a sample

$$x_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

We initialize the weight matrix randomly (as a first step)

$$W_{init} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Multiply it with weights to get scores

$$z = x_i \times W = \begin{bmatrix} 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

Let's talk about **Gradient Descent** - not as a backup plan, but as the way modern ML actually works at scale.

$$\mathbf{w}^{s+1} = \mathbf{w}^{s} - \eta \nabla \mathcal{L}(\mathbf{w}^{s}) \xrightarrow{\text{OR}} W_{new} = W_{init} - (\eta \cdot grad_i)$$

We have the training inputs X and ground truth y

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

We pick a sample

$$x_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

We initialize the weight matrix randomly (as a first step)

$$W_{init} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Multiply it with weights to get scores

$$z = x_i \times W = \begin{bmatrix} 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

Then get the probability

$$P = \text{Softmax}(z) = \begin{bmatrix} 1/3 & 1/3 & 1/3 \end{bmatrix} \approx \begin{bmatrix} 0.33 & 0.33 & 0.33 \end{bmatrix}$$ where $$P_k = \frac{e^{z_k}}{\sum_{j=0}^{K-1} e^{z_j}}$$

Let's talk about **Gradient Descent** - not as a backup plan, but as the way modern ML actually works at scale.

$$\mathbf{w}^{s+1} = \mathbf{w}^{s} - \eta \nabla \mathcal{L}(\mathbf{w}^{s}) \xrightarrow{\text{OR}} W_{new} = W_{init} - (\eta \cdot grad_i)$$

Now we need to find error (predicted-true)

$$e = P - y_i = \begin{bmatrix} 0.33 - 1 & 0.33 - 0 & 0.33 - 0 \end{bmatrix} = \begin{bmatrix} -0.67 & 0.33 & 0.33 \end{bmatrix}$$

Let's talk about **Gradient Descent** - not as a backup plan, but as the way modern ML actually works at scale.

$$\mathbf{w}^{s+1} = \mathbf{w}^{s} - \eta \nabla \mathcal{L}(\mathbf{w}^{s}) \xrightarrow{\text{OR}} W_{new} = W_{init} - (\eta \cdot grad_i)$$

Now we need to find error (predicted-true)

$$e = P - y_i = \begin{bmatrix} 0.33 - 1 & 0.33 - 0 & 0.33 - 0 \end{bmatrix} = \begin{bmatrix} -0.67 & 0.33 & 0.33 \end{bmatrix}$$

Find the gradient matrix for the specific sample

$$grad_i = x_i^T \times e = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \times \begin{bmatrix} -0.67 & 0.33 & 0.33 \end{bmatrix} = \begin{bmatrix} -0.67 & 0.33 & 0.33 \\ 0 & 0 & 0 \end{bmatrix}$$

Let's talk about **Gradient Descent** - not as a backup plan, but as the way modern ML actually works at scale.

$$\mathbf{w}^{s+1} = \mathbf{w}^{s} - \eta \nabla \mathcal{L}(\mathbf{w}^{s}) \xrightarrow{\text{OR}} W_{new} = W_{init} - (\eta \cdot grad_i)$$

Now we need to find error (predicted-true)

$$e = P - y_i = \begin{bmatrix} 0.33 - 1 & 0.33 - 0 & 0.33 - 0 \end{bmatrix} = \begin{bmatrix} -0.67 & 0.33 & 0.33 \end{bmatrix}$$

Find the gradient matrix for the specific sample

$$grad_i = x_i^T \times e = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \times \begin{bmatrix} -0.67 & 0.33 & 0.33 \end{bmatrix} = \begin{bmatrix} -0.67 & 0.33 & 0.33 \\ 0 & 0 & 0 \end{bmatrix}$$

Find the new weight matrix

$$W_{new} = W_{init} - (\eta \cdot grad_i)$$

$$W_{new} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} -0.067 & 0.033 & 0.033 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{W_{new}} = \begin{bmatrix} 0.067 & -0.033 & -0.033 \\ 0 & 0 & 0 \end{bmatrix}$$

Let's talk about **Gradient Descent** - not as a backup plan, but as the way modern ML actually works at scale.

$$\mathbf{w}^{s+1} = \mathbf{w}^{s} - \eta \nabla \mathcal{L}(\mathbf{w}^{s}) \xrightarrow{\text{OR}} W_{new} = W_{init} - (\eta \cdot grad_i)$$

Now we need to find error (predicted-true)

$$e = P - y_i = \begin{bmatrix} 0.33 - 1 & 0.33 - 0 & 0.33 - 0 \end{bmatrix} = \begin{bmatrix} -0.67 & 0.33 & 0.33 \end{bmatrix}$$

Find the gradient matrix for the specific sample

$$grad_i = x_i^T \times e = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \times \begin{bmatrix} -0.67 & 0.33 & 0.33 \end{bmatrix} = \begin{bmatrix} -0.67 & 0.33 & 0.33 \\ 0 & 0 & 0 \end{bmatrix}$$

Find the new weight matrix

$$W_{new} = W_{init} - (\eta \cdot grad_i)$$

$$W_{new} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} -0.067 & 0.033 & 0.033 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{W_{new}} = \begin{bmatrix} 0.067 & -0.033 & -0.033 \\ 0 & 0 & 0 \end{bmatrix}$$

We move on to next sample(s) to find new weights.
But weights are not initialized randomly this time !

$$z = x_2 \times W = \begin{bmatrix} 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0.067 & -0.033 & -0.033 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

***

Why Walk When You Can Jump: Second-order Solution (Newton's Method)

Gradient Descent Weight Update

$$W_{new} = W_{init} - (\eta \cdot grad_i)$$

Newton's Weight Update

$$W_{new} = W_{old} - (H^{-1} \times G)$$

Why Walk When You Can Jump: Second-order Solution (Newton's Method)

Gradient Descent Weight Update

$$W_{new} = W_{init} - (\eta \cdot grad_i)$$

Only needs to store the Gradients.

Newton's Weight Update

$$W_{new} = W_{old} - (H^{-1} \times G)$$

Must store the Hessian matrix. .

Why Walk When You Can Jump: Second-order Solution (Newton's Method)

Gradient Descent Weight Update

$$W_{new} = W_{init} - (\eta \cdot grad_i)$$

Only needs to store the Gradients.

Newton's Weight Update

$$W_{new} = W_{old} - (H^{-1} \times G)$$

Must store the Hessian matrix. .

***

Interview Moment: Second-order solution in actual life production

You're training a logistic regression model with **1000 features**.
Gradient descent is taking **5,000 iterations to converge** and your manager is annoyed that it takes too much iterations.

Your PhD colleague says '**Use Newton's method** - it'll converge in under 10 iterations.'

Your manager says 'Perfect, ship it!'

But you know something they forgot about.

What's the problem, and **when** would you actually **use Newton's method**?

Interview Moment: Second-order solution in actual life production

Newton's method is misleading here.

Yes, it converges in 10 iterations but **each Newton iteration is about 100 times more expensive** than a gradient descent iteration.

Gradient descent is actually faster overall.

Good Answer

Interview Moment: Second-order solution in actual life production

Newton's method is misleading here.

Yes, it converges in 10 iterations but **each Newton iteration is about 100 times more expensive** than a gradient descent iteration.

Gradient descent is actually faster overall.

Good Answer

My colleague is **technically correct but practically wrong**.

Here's why: At 1,000 features, **Newton has to build and invert** a 1,000×1,000 matrix every single iteration.

When Newton actually wins: Under 100 features and you need machine-precision.
At 1,000 features, **it's questionable.**

My recommendation: Use L-BFGS instead.

It's what scikit-learn uses by default for logistic regression.

We'll get **Newton-like convergence without the computational overhead.**

Best of both worlds.

FAANG-level Answer

***

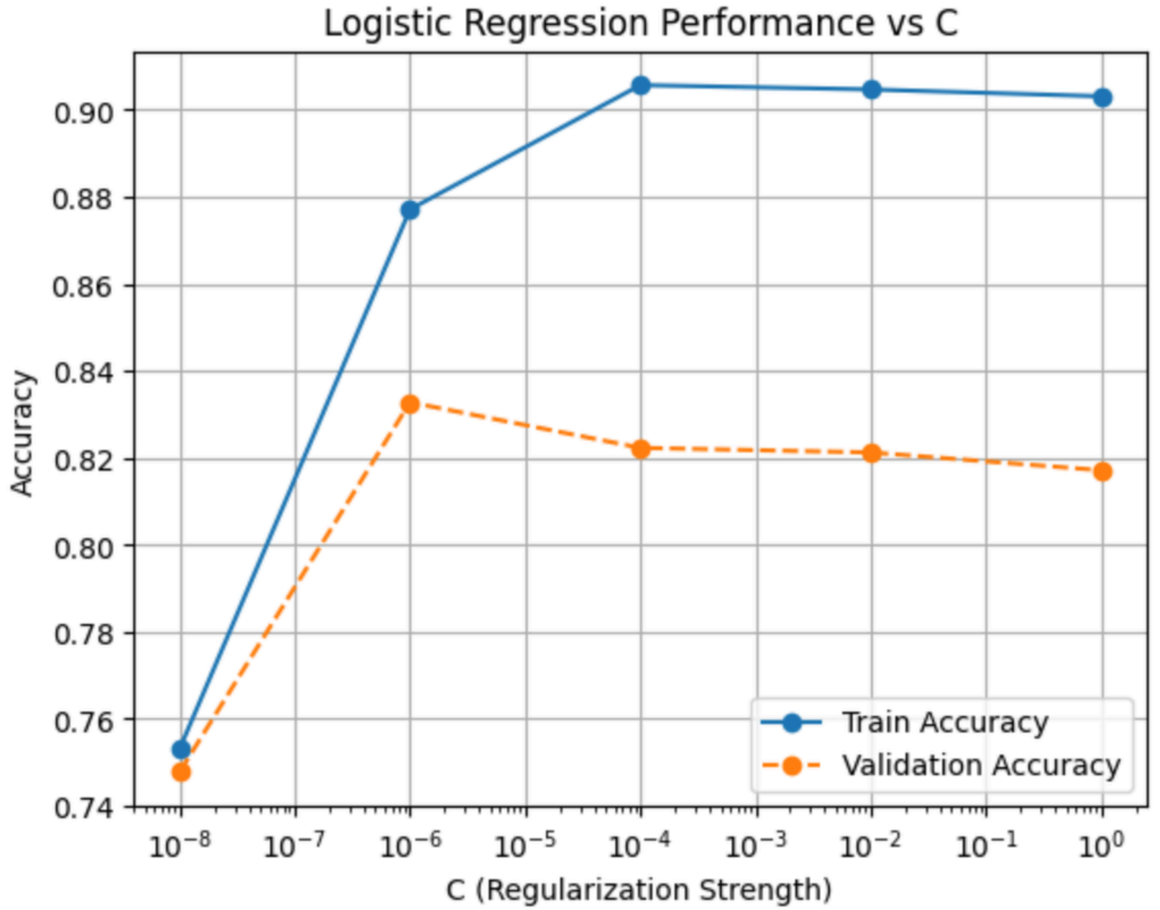
Interview Moment: Sweet Spot for Regularization

In this regularization sweep, **training accuracy increases as we decrease regularization** (increase C), which makes sense.

But **validation accuracy peaks in the middle** and then slightly decreases.

Why does MORE regularization (smaller C) **hurt validation performance**?

Isn't regularization supposed to prevent **overfitting**?



Interview Moment: Sweet Spot for Regularization

When C is too small, the **model becomes too simple** and **can't learn the patterns** in the data, so **validation accuracy goes down.**

Good Answer

Interview Moment: Sweet Spot for Regularization

When C is too small, the **model becomes too simple** and **can't learn the patterns** in the data, so **validation accuracy goes down.**

Good Answer

This demonstrates the **bias-variance tradeoff**
When C is tiny, we're 'ignore the data, just try to minimize weights, creating **high bias.** It's **too simple to capture the true patterns** which is classic **underfitting**. Train and validation are both low. we're not even fitting the training data well.

At sweet spot, validation peaks. C value perfectly balances:

- Enough flexibility to learn patterns
- Enough constraint to avoid noise

Weak regularization. Training keeps improving but **validation dips slightly.** The model starts fitting training-specific noise. **Regularization isn't free.** If we increase it too much the model becomes so constrained it can't learn real patterns. **We should never assume 'more regularization = better.'**

FAANG-level Answer

***

Historical Fact: Regularization Rescued Netflix

Netflix offered $1,000,000 to anyone who could improve their movie recommendation algorithm Cinematch.

The dataset was massive: 100 million ratings from 480,000 users.

The Overfitting Problem: The data was extremely "sparse" (most users had only rated a few movies), the models would easily "memorize" the specific ratings of a few users rather than finding general movie preferences.

Without constraints, the model's parameters would grow excessively large to perfectly match the training ratings, leading to poor performance on the "unseen" test set.

Historical Fact: Regularization Rescued Netflix

The Regularization Solution: The winning teams, integrated L2 Regularization (weight decay) directly into their optimization.

- They added a penalty term (λ) to the loss function that penalized the squared magnitude of the model's parameters.
- This "penalty for complexity" forced the model to keep weights small, preventing it from latching onto the "noise" or quirks of individual users.

Historical Fact: Regularization Rescued Netflix

The Outcome: Regularization allowed the model to generalize across the entire user base. This technique, combined with ensemble methods, finally broke the 10% improvement barrier to win the grand prize.

The BellKor Solution to the Netflix Grand Prize

Yehuda Koren

August 2009

I. Introduction

This article describes part of our contribution to the "BellKor's Pragmatic Chaos" final solution, which won the Netflix Grand Prize. The other portion of the contribution was created while working at AT&T with Robert Bell and Chris Volinsky, as reported in our 2008 Progress Prize report [3]. The final solution includes all the predictors described there. In this article we describe only the newer predictors.

So what is new over last year's solution? First we further improved the baseline predictors (Sec. III). This in turn improves our other models, which incorporate those predictors, like the matrix factorization model (Sec. IV). In addition, an extension of the neighborhood model that addresses temporal dynamics was introduced (Sec. V). On the Restricted Boltzmann Machines (RBM) front, we use a new RBM model with superior accuracy by conditioning the visible units (Sec. VI). The final addition is the introduction of a new blending algorithm, which is based on gradient boosted decision trees (GBDT) (Sec. VII).

therefore harder to predict. In a way, this represents real requirements for a collaborative filtering (CF) system, which needs to predict new ratings from older ones, and to equally address all users, not just the heavy raters.

We reserve special indexing letters to distinguish users from movies: for users u, v, and for movies i, j. A rating r_{ui} indicates the preference by user u of movie i. Values are ranging from 1 (star) indicating no interest to 5 (stars) indicating a strong interest. We distinguish predicted ratings from known ones, by using the notation $\hat{r}_{ui}$ for the predicted value of r_{ui}.

The scalar t_{ui} denotes the time of rating r_{ui}. Here, time is measured in days, so t_{ui} counts the number of days elapsed since some early time point. About 99% of the possible ratings are missing, because a user typically rates only a small portion of the movies. The (u,i) pairs for which r_{ui} is known are stored in the *training set* $\mathcal{K} = \{(u,i) \mid r_{ui} \text{ is known}\}$. Notice that $\mathcal{K}$ includes also the Probe set. Each user u is associated with a set of items denoted by $R(u)$, which contains all the items for which ratings by u are available. Likewise, $R(i)$ den

Now is your turn

Task: Regression

Goals:

- Best performance R score
- The earliest and best-score answer wins

You can work in Teams

Total given time: 20 minutes

You need to do tell us how did you achieve this success