Optimization in Machine Learning

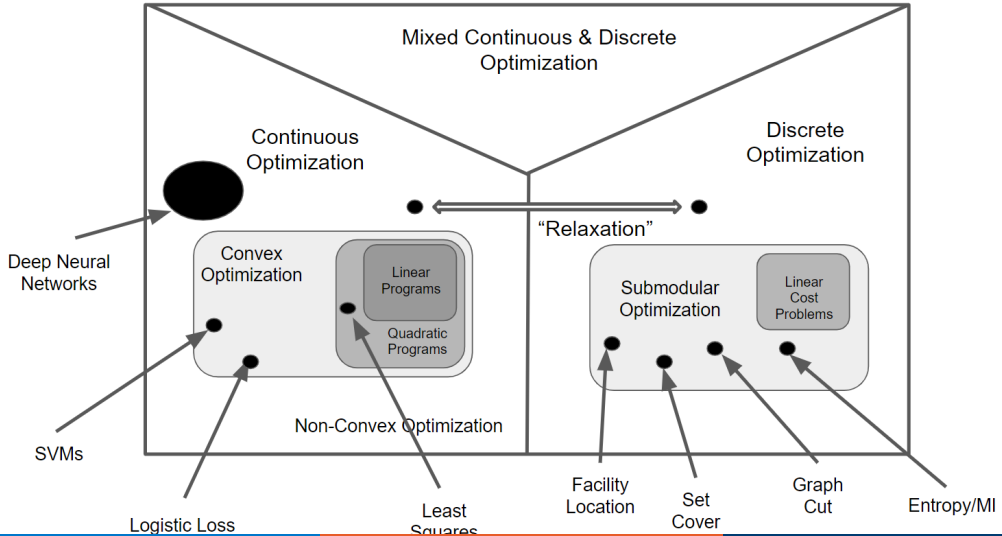
Lecture 2: Continuous Optimization Examples (PCA, Matrix Factorization, Low Rank and NMF, Clustering, Contextual Bandits) and Calculus Brushup for Optimization in Machine Learning

Ganesh Ramakrishnan

Department of Computer Science
Dept of CSE, IIT Bombay
https://www.cse.iitb.ac.in/~ganesh

January, 2025

[Recap] Big Picture: Types of Optimization Problems



Logistics, Grading etc: Mostly moodle, some at https://www.cse.iitb.ac.in/~ganesh/cs769/

- Credit/Audit Requirements Anyone who does an exceptional course project that has the potential to be a publishable paper is eligible for a straight AA grade. Otherwise the grading breakup would be:
  - 20% Mid-semester exam
  - 25% End semester exam
  - 35% Project: A basic project will take any of the algorithms we study or any related papers, implement the algorithms in the paper, do a basic performance study and diagnose the performance. However, I would expect most projects to suggest ideas for improvement (atleast in specific settings such as multi core or multiple nodes or reasonable assumptions on matrices etc in the problem for which greater speedup is possible). A more advanced project would take a problem specification for which no solution is publicly available, figure out how to solve it, and implement the solution.
  - 20% Reading and paper presentation.
- Lectures: In Slot 9, Lectures on MS Teams (Code: **cnsjv56**)
- All lecture recordings and slides will be organized on moodle (CS 769-2024-2— Course name Optimization in Machine Learning)
- TA(s): Prateek Chand, Priya Mishra, Suraj Racha, Vedant Goswami (and possibly more TAs might get added)

Outline

- Why take this course? [Done]
- Prerequisites [Done]
- Course plan [Done]
- Course Logistics [Done]
- Continuous Optimization in Machine Learning: Applications include
  1. Supervised Learning [Done]
  2. Principal Component Analysis
  3. Low Rank and Non Negative Matrix Factorization
  4. Clustering
  5. Contextual Bandits and Learning from Logged Data
- Discrete Optimization in Machine Learning

Recap: Continuous Optimization in Machine Learning

Continuous Optimization often appears as *relaxations* of empirical risk minimization problems.

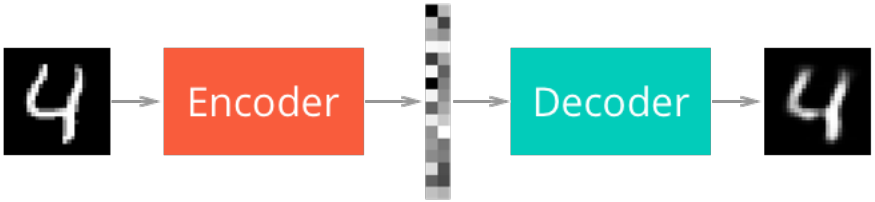
- Supervised Learning: Fine tuning of Deep Models, Fine tuning of LLMs
  - Canonical examples of Logistic Regression, Least Squares, Support Vector Machines [Done]
- Unsupervised Learning: Pre-training of Deep Models, Pre-training of LLMs,
  - Principal Component Analysis, k-Means Clustering
- Reinforcement Leaning from Human Feedback (RLHF), Agentic Systems
  - Canonical examples of Contextual Bandits and Reinforcement Learning: Soft-Max Estimators, Policy Exponential Models
  - Canonical examples of Recommender Systems: Matrix Completion, Non-Negative Matrix Factorization, Collaborative Filtering

Application 2: Unsupervised Learning & Representation Learning

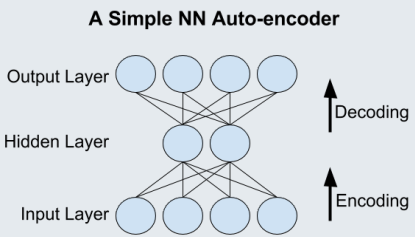
https://www.youtube.com/watch?v=DxNomXXbYPg&list=PLyo3HAXSZD3zfv9O-y9DJhvrWQPscqATa&index=24

Application 2: Unsupervised Learning: Autoencoders

Autoencoders: Training to Copy Input to Output

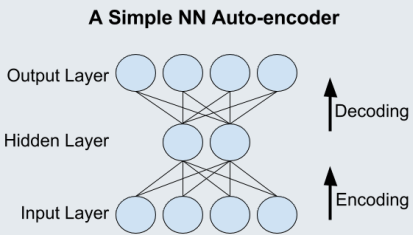


Autoencoders: Training to Copy Input to Output



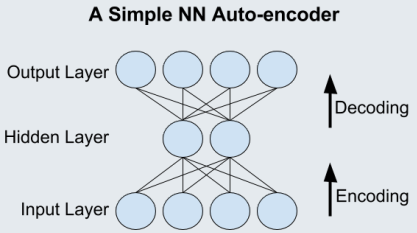
- Hidden layer $\mathbf{h} \in \Re^p$ describes a code used to represent the input/output $\mathbf{x} \in \Re^n$.
  1. Decoder produces a reconstruction g: $= g(\mathbf{h})$
  2. Encoder encodes using function f: $\mathbf{h} = f(\mathbf{x})$
- Loss function $L(\mathbf{x})$ (eg: $\|\mathbf{x} - g(f(\mathbf{x}))\|^2$) designed so that $g(f(\mathbf{x})) \approx \mathbf{x}$.
- Why $\approx$ (approximate) and not exact $=$?
- Learn salient aspects of input and prioritize what to copy and what not to! (Recap: PCA)

Autoencoders: Generalization of PCA



- For $\mathbf{x} \in \Re^n$ and $\mathbf{h} \in \Re^p$, consider the following simplification without activations:
  1. Decoder produces a reconstruction $g \sim W_1 \in \Re^{n \times p}$: $= W_1 \mathbf{h}$
  2. Encoder encodes using function $f \sim W_2 \in \Re^{p \times n}$: $\mathbf{h} = W_2 \mathbf{x}$
- Loss function $L(\mathbf{x}) = |\mathbf{x} - W_1 W_2 \mathbf{x}|$ is minimized
- Roughly PCA: $W_1^T W_1 = I$ (decorrelated hidden layer).

Autoencoders: Overcomplete & Undercomplete



- Hidden layer $\mathbf{h} \in \Re^p$ describes a code used to represent the input/output $\mathbf{x}, \in \Re^n$.
  1. Decoder produces a reconstruction g: $= g(\mathbf{h})$
  2. Encoder encodes using function f: $\mathbf{h} = f(\mathbf{x})$
- Overcomplete: When $p > n$. Problem: $\mathbf{h}$ can become $\mathbf{x}$ padded with 0's!
  - Solution1 (Denoising Autoencoder): Corrupt $\mathbf{x}$ with noise to make $\mathbf{h}$ learn to clean the noise
  - Solution2 (Variational Autoencoder): Insist that $\mathbf{h}$ looks like a Gaussian
- Undercomplete: When $p < n$.

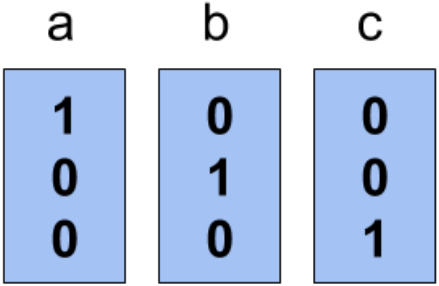
See `https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3_NeuralNetworks/autoencoder.ipynb`

Application 2: Unsupervised Learning: Representation Learning

https://www.youtube.com/watch?v=DxNomXXbYPg&list=PLyo3HAXSZD3zfv9O-y9DJhvrWQPscqATa&index=24

One Hot Encoding for Characters

- With 3 characters in vocabulary, a,b and c, what would be the best encoding to inform each character occurrence to the network?
- One Hot Encoding: Give a unique key k to each character in alpha-numeric order, and encode each character with a vector of vocabulary size, with a 1 for the k^{th} element, and 0 for all other elements.



Embeddings (Eg: for Words)

How to encode the words for the task of labeling a drama reviews as "liked" or "not liked" ?

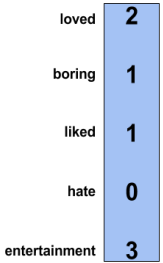
- Review 1: The drama was interesting, loved the way each scene was directed. I simply loved everything in the drama.
- Review 2: I had three boring hours. Very boring to watch.
- Review 3: I liked the role each that was assigned to each super star. Especially loved the performance of actor.
- Review 4: Though I hate all the dramas of the director, this one was an exception with lot of entertainment.

Encoding Words

How to encode the words for the task of labeling a ''*drama*' reviews as "liked" or "not liked" ?

- One Hot Encoding of Words.
- Bag Of Words, similar to one hot encoding of characters
  - Use the vocabulary of highly frequent words in reviews.
  - Use the word frequency in each review instead of "1".

A review in Bag Of Words Form:-



(Word) Embedding: Motivation

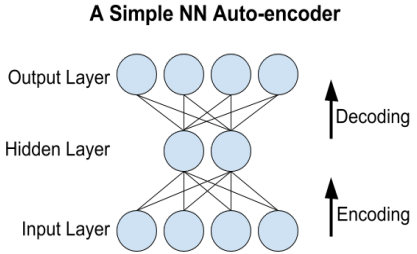
Limitations of Bag of Words or One Hot Encoding for words

- High Dimension: In real life scenario, the vocabulary size could be huge.
- Lacks Contextual Similarity - e.g. liked and loved are contextually similar words.

(Word) Embedding: Motivation

Dimensionality Reduction techniques.

- Bag of Frequent Words: Contextual similarity is still lacking.
- **What happens if one passes a one hot encoded word as both input and output to a NN?**
- **NN Auto-encoder: Output has same form as input. Extract the encoded vector from the hidden layer.** Another Eg: Auto Encoders using CNNs (intrusion detection in videos: `https://arxiv.org/pdf/1701.01546.pdf`)



(Word) Embedding: Motivation

After unsupervised training with lot of online data, can a machine answer questions like:-

- King - Man + Woman = ?
- If France:Paris, then Japan:?

(Word) Embedding: Motivation

A Hypothetical Word Vector Representation

	King	Queen	Woman	Princess
Royalty	0.98	0.98	0.01	0.93
Masculinity	0.98	0.04	0.02	0.02
Femininity	0.05	0.92	0.99	0.95
Age	0.7	0.6	0.5	0.2

- What would be the vector for Man?
- King - Man + Woman = ?
- If King:Man then Queen:?

(Word) Embedding: Motivation

A Hypothetical Word Vector Representation

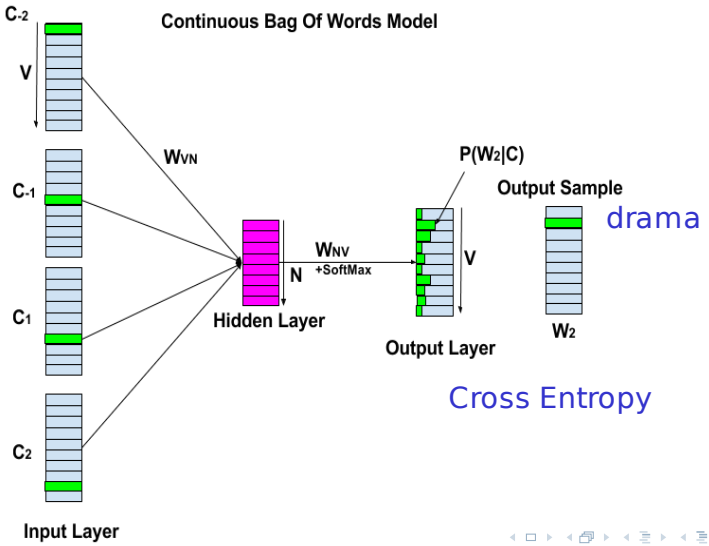
	King	Queen	Woman	Princess
Royalty	0.98	0.98	0.01	0.93
Masculinity	0.98	0.04	0.02	0.02
Femininity	0.05	0.92	0.99	0.95
Age	0.7	0.6	0.5	0.2

- What would be the vector for Man? **[0.01, 0.98, 0.05, 0.6]'**
- King - Man + Woman = ? Queen (Subtraction/addition approximates vector for Queen)
- If King:Man then Queen:? Woman (Vector differences of both pairs nearly same)

Simplest (Word) Embedding

- **(Word) Embedding:** Building a low-dimensional vector representation from corpus of text, which preserves contextual similarity.
- **In simple terms:** Need an efficient language of numbers which deep neural networks can understand as close as possible to the way we understand words.
- **Training:** Continuous Bag of Words Model.
  - Take words in one hot encoded form.
  - Consider the sentence, "... I really liked the **drama**....".
  - Take a N (say 5) word window around each such word w.
  - Train the Neural Network with (top V frequent) context words set $\mathcal{C}$ as input and the central word w as output.
  - For the example above use C = {"I", "really", "the" "drama"} as input and w = **"liked"** as output.

(Word) Embedding: Unsupervised Training

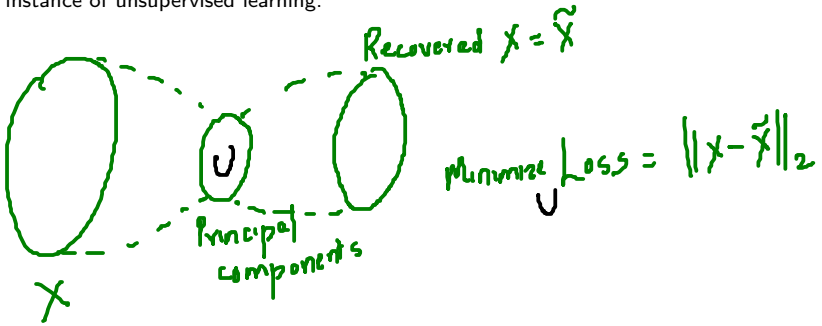


Application 2: One formulation as Principal Component Analysis

- This is an instance of unsupervised learning.

Application 2: One formulation as Principal Component Analysis

- This is an instance of unsupervised learning.



PCA: Error Reduction Perspective

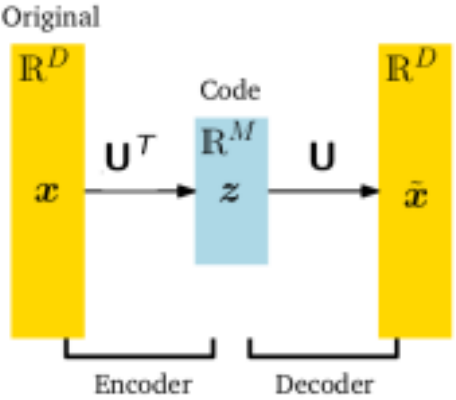


Figure: Illustration of Error Reduction Perspective of PCA. Picture courtesy: "Mathematics for Machine Learning", Marc Peter Deisenroth, A. Aldo Faisal & Cheng Soon Ong

Application 2: One formulation as Principal Component Analysis

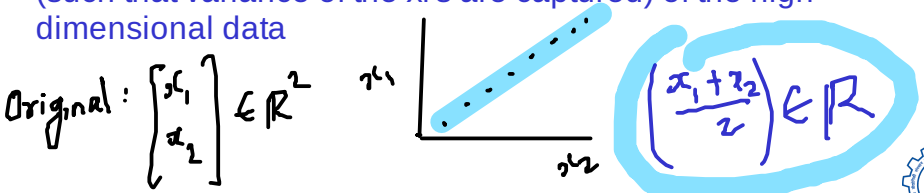
- This is an instance of unsupervised learning.
- Data: Given unlabeled data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ are the feature vectors.

Application 2: One formulation as Principal Component Analysis

- This is an instance of unsupervised learning.
- Data: Given unlabeled data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ are the feature vectors.

Goal: To find a lower dimensional meaningful representation (such that variance of the xi's are captured) of the high dimensional data

EXAMPLE:

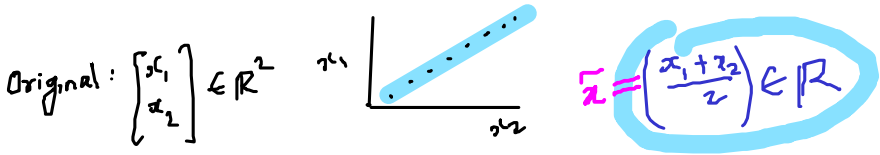


(Another Example: Face features in the extra reading)
Note: Spread = Variance

Application 2: One formulation as Principal Component Analysis

- This is an instance of unsupervised learning.
- Data: Given unlabeled data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ are the feature vectors.
- Goal: Find compressors $U \in \mathbb{R}^{m \times k}$ (and correspondingly decompressors $V \in \mathbb{R}^{k \times m}$) such that x_i is *close* to UVx_i.

Application 2: One formulation as Principal Component Analysis



- This is an instance of unsupervised learning.
- Data: Given unlabeled data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ are the feature vectors.
- Goal: Find compressors $U \in \mathbb{R}^{m \times k}$ (and correspondingly decompressors $V \in \mathbb{R}^{k \times m}$) such that x_i is *close* to UVx_i.

$x_1 = \tilde{x}$

$x_2 = \tilde{x}$

Application 2: One formulation as Principal Component Analysis

- This is an instance of unsupervised learning.
- Data: Given unlabeled data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ are the feature vectors.
- Goal: Find compressors $U \in \mathbb{R}^{m \times k}$ (and correspondingly decompressors $V \in \mathbb{R}^{k \times m}$) such that x_i is *close* to UVx_i.

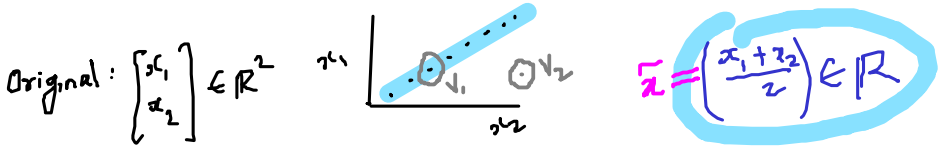
1) Scale preserving: U has bounded magnitude

2) In an expected sense:
If x_i and x_j are uncorrelated then
Ux_i and Ux_j are also uncorrelated

Application 2: One formulation as Principal Component Analysis

- This is an instance of unsupervised learning.
- Data: Given unlabeled data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ are the feature vectors.
- Goal: Find compressors $U \in \mathbb{R}^{m \times k}$ (and correspondingly decompressors $V \in \mathbb{R}^{k \times m}$) such that x_i is *close* to UVx_i.
- It turns out the compressor must satisfy $V = U^T$ such that $U^T U = I$.

Application 2: One formulation as Principal Component Analysis



- This is an instance of unsupervised learning.
- Data: Given unlabeled data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ are the feature vectors.
- Goal: Find compressors $U \in \mathbb{R}^{m \times k}$ (and correspondingly decompressors $V \in \mathbb{R}^{k \times m}$) such that x_i is *close* to UVx_i.
- It turns out the compressor must satisfy $V = U^T$ such that $U^T U = I$.

Under fairly natural constraints

When solved using constrained optimization (Lagrange multipliers)

We assume: 1)Compressor U has bounded magnitude

Eg: $\|U_i\|_2 = 1$

2) In an expected sense:
If x_i and x_j are uncorrelated then
Ux_i and Ux_j are also uncorrelated

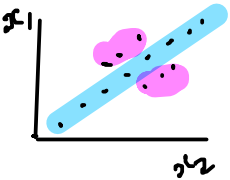
Application 2: One formulation as Principal Component Analysis

- This is an instance of unsupervised learning.
- Data: Given unlabeled data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ are the feature vectors.
- Goal: Find compressors $U \in \mathbb{R}^{m \times k}$ (and correspondingly decompressors $V \in \mathbb{R}^{k \times m}$) such that x_i is *close* to UVx_i.
- It turns out the compressor must satisfy $V = U^T$ such that $U^T U = I$.
- Optimization Problem: The PCA optimization problem is:

$$\min_{U: U^T U = I} \sum_{i=1}^{n} ||x_i - UU^T x_i||_2^2$$

Application 2: One formulation as Principal Component Analysis

- This is an instance of unsupervised learning.
- Data: Given unlabeled data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ are the feature vectors.
- Goal: Find compressors $U \in \mathbb{R}^{m \times k}$ (and correspondingly decompressors $V \in \mathbb{R}^{k \times m}$) such that x_i is *close* to UVx_i.
- It turns out the compressor must satisfy $V = U^T$ such that $U^T U = I$.
- Optimization Problem: The PCA optimization problem is:



$$\min_{U : U^T U = I} \sum_{i=1}^{n} ||x_i - UU^T x_i||_2^2$$

$U \in \mathbb{R}^{k \times m}$

contribute to sum of errors

PCA through Illustration

- Let $\mathbf{X}$ be a random vector and $\Gamma = \mathbf{E}[(\mathbf{X} - \mathbf{E}[\mathbf{X}])(\mathbf{X} - \mathbf{E}[\mathbf{X}])^T]$ its covariance matrix. Let $\mathbf{e}_1, \ldots, \mathbf{e}_n$ be the n (normalized) eigenvectors of Γ.
- The n principal components of $\mathbf{X}$ are said to be $\mathbf{e}_1^T\mathbf{X}$, $\mathbf{e}_2^T\mathbf{X}$, ..., $\mathbf{e}_n^T\mathbf{X}$. See https://arxiv.org/abs/1804.10253

1. Let $p(X_1) = \mathcal{N}(0, 1)$ and $p(X_2) = \mathcal{N}(0, 1)$ and $cov(X_1, X_2) = \theta$. Find all the principal components of the random vector $\mathbf{X} = [X_1, X_2]^T$.
2. Now, let $\mathbf{Y} = \mathcal{N}(\mathbf{0}, \Sigma) \in \Re^p$ where $\Sigma = \lambda^2 I_{p\times p} + \alpha^2 ones(p, p)$ for any $\lambda, \alpha \in \Re$. Here, $I_{p\times p}$ is a $p \times p$ identity matrix while $ones(p, p)$ is a $p \times p$ matrix of 1's. Find atleast one principal component of $\mathbf{Y}$.

Solution:

1. We first note that the 2×2 matrix

$$\Gamma = \begin{bmatrix} 1 & \theta \\ \theta & 1 \end{bmatrix}$$

2. To compute the eigenvalues of Γ from the characteristic polynomial:

$$\left| \begin{bmatrix} 1-\lambda & \theta \\ \theta & 1-\lambda \end{bmatrix} \right| = (1-\lambda)^2 - \theta^2 = 0$$

$\Rightarrow \lambda_1 = 1+\theta$ and $\lambda_2 = 1-\theta$ are the two solutions/eigenvalues.

3. For eigenvalue λ_1 we find its eigenvector v_1:

$$\begin{bmatrix} 1 & \theta \\ \theta & 1 \end{bmatrix} v_1 = (1+\theta) v_1$$

which is easily solvable as

$$v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

or its normalized version

$$v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

Solution:

1. We first note that the 2×2 matrix

$$\Gamma = \begin{bmatrix} 1 & \theta \\ \theta & 1 \end{bmatrix}$$

2. To compute the eigenvalues of Γ from the characteristic polynomial:

$$\left| \begin{bmatrix} 1-\lambda & \theta \\ \theta & 1-\lambda \end{bmatrix} \right| = (1-\lambda)^2 - \theta^2 = 0$$

$\Rightarrow \lambda_1 = 1+\theta$ and $\lambda_2 = 1-\theta$ are the two solutions/eigenvalues.

3. For eigenvalue λ_1 we find its eigenvector v_1:

$$\begin{bmatrix} 1 & \theta \\ \theta & 1 \end{bmatrix} v_1 = (1+\theta) v_1$$

which is easily solvable as

$$v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

or its normalized version

$$v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

4. Similarly, for eigenvalue λ_2 we find its eigenvector v_2:

$$\begin{bmatrix} 1 & \theta \\ \theta & 1 \end{bmatrix} v_2 = (1-\theta) v_2$$

which is easily solvable as

$$v_2 = \begin{bmatrix} 1 \\ -1/ \end{bmatrix}$$

or its normalized version

$$v_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$

5. Thus, the principal components of $\mathbf{X}$ are $X_1/\sqrt{2} + X_2/\sqrt{2}$ and $X_1/\sqrt{2} - X_2/\sqrt{2}$.

- Now, let $\mathbf{Y} = \mathcal{N}(\mathbf{0}, \Sigma) \in \Re^p$ where $\Sigma = \lambda^2 I_{p\times p} + \alpha^2 ones(p,p)$ for any $\lambda, \alpha \in \Re$. Here, $I_{p\times p}$ is a $p \times p$ identity matrix while $ones(p,p)$ is a $p \times p$ matrix of 1's. Find atleast one principal component of $\mathbf{Y}$.

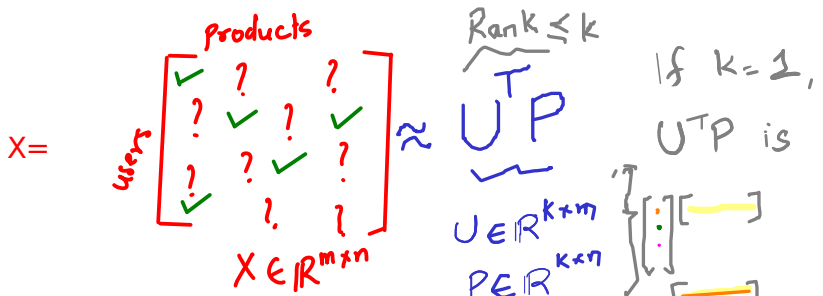
Application 3: Matrix Completion

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.

Application 3: Matrix Completion

Eg: A_j could be matrix premultiplication based on costs

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.



Goal: Fill up the missing (?) entries by profiling user behaviour and product behaviour

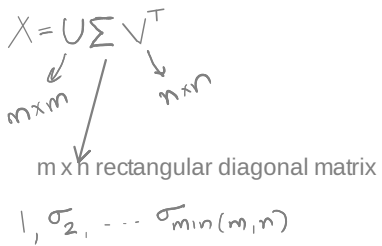
Application 3: Matrix Completion

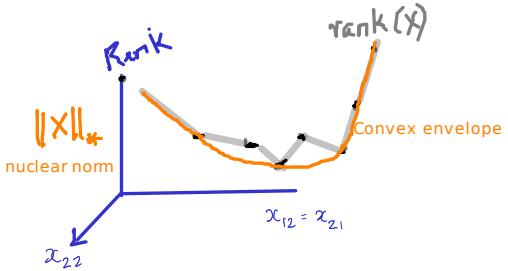
- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.
- Goal: Find the *simplest* matrix X s.t $A_j(X) \approx y_j, \forall j \in 1, \cdots, n$

Application 3: Matrix Completion

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.
- Goal: Find the *simplest* matrix X s.t $A_j(X) \approx y_j, \forall j \in 1, \cdots, n$

RANK CONSTRAINED





$V^T V = I \quad U^T U = I$

Singular value decomp: $X = \underset{m\times n}{V} \underset{m\times m}{\Sigma} \overset{m\times n}{\underset{n\times n}{U^T}}$ $\quad \Sigma = \begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_k \\ & & & \ddots 0 \end{bmatrix}$ k = rank of $X \leq \min(m,n)$

Recap: Eigenvalue decomp. for $A \in \mathbb{R}^{p\times p}$ $\quad Ax = \lambda x$

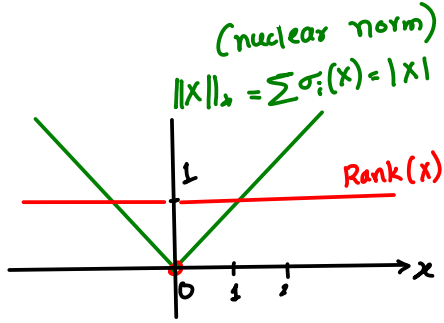
$AX = X\Lambda$ → Diagonal Matrix of Eigenvalues

$X^T X = U\Sigma^T V^T V \Sigma U^T = U\Sigma^2 U^T$

$XX^T = V\Sigma^T \Sigma V^T$

} Eigenvalues of $X^T X$ & of XX^T are squares of singular values of X

Thus, if X is n×n real symmetric matrix with non-negative eigenvalues, then eigenvalues and singular values coincide, but it is not generally the case!



1) Rank function is certainly not continuous.
2) But nuclear norm is continuous and also convex
3) Nuclear norm and rank function have the same minimzer
4) The nuclear norm is a convex envolope (tightest surrogate) for the rank function (which means they have the same global minimizer)

Application 3: Matrix Completion

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.
- Goal: Find the *simplest* matrix X s.t $A_j(X) \approx y_j, \forall j \in 1, \cdots, n$
- Optimization Problem: Matrix completion optimization problem is:

$$\min_X \sum_{j=1}^{n} ||y_j - A_j(X)||_2^2 + ||X||_*$$

Application 3: Matrix Completion

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.
- Goal: Find the *simplest* matrix X s.t $A_j(X) \approx y_j, \forall j \in 1, \cdots, n$
- Optimization Problem: Matrix completion optimization problem is:

$$\min_X \sum_{j=1}^{n} ||y_j - A_j(X)||_2^2 + ||X||_* = \sum_{i=1}^{\min(m,n)} |\sigma_i[X]|$$

scaling/weightage hyperparam = 1

Can we extend the PCA kind of algorithmic solution to solve for X here?
Ans: Yes - Singular Thresholding Algorithm
Q: What would the algorithm look like?

Application 3: Matrix Completion

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.
- Goal: Find the *simplest* matrix X s.t $A_j(X) \approx y_j, \forall j \in 1, \cdots, n$
- Optimization Problem: Matrix completion optimization problem is:

$$\min_X \sum_{j=1}^{n} ||y_j - A_j(X)||_2^2 + ||X||_*$$

- Can be solved using Iterative Singular Value Thresholding!

Application 3: Matrix Completion

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.
- Goal: Find the *simplest* matrix X s.t $A_j(X) \approx y_j, \forall j \in 1, \cdots, n$
- Optimization Problem: Matrix completion optimization problem is:

$$\min_X \sum_{j=1}^{n} ||y_j - A_j(X)||_2^2 + ||X||_*$$

by including only first i singular values and vectors

$\tilde{X}_i = U_i \Sigma_i V_i$

- Can be solved using Iterative Singular Value Thresholding!

$X_0 \longrightarrow X_0 = U\Sigma_0 V^T \longrightarrow$ sort $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_{min(m,n)}$

Solution to $\text{argmin} \sum_{i=1}^{n} ||y_j - A_j(X)||^2$

Keep including σ_i to generate the completed $\tilde{X}_i$ until objective decreases and including corresponding eigenvector

Application 3: Matrix Completion

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.
- Goal: Find the *simplest* matrix X s.t $A_j(X) \approx y_j, \forall j \in 1, \cdots, n$
- Optimization Problem: Matrix completion optimization problem is:

$$\min_X \sum_{j=1}^{n} ||y_j - A_j(X)||_2^2 + ||X||_*$$

- Can be solved using Iterative Singular Value Thresholding!
- Another way is to explicitly model this is by assuming $X = LR$ where $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ (and hence X is rank k), and optimize for L and R.

Application 3: Matrix Completion

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.
- Goal: Find the *simplest* matrix X s.t $A_j(X) \approx y_j, \forall j \in 1, \cdots, n$
- Optimization Problem: Matrix completion optimization problem is:

$$\min_X \sum_{j=1}^{n} ||y_j - A_j(X)||_2^2 + ||X||_*$$

- Can be solved using Iterative Singular Value Thresholding!
- Another way is to explicitly model this is by assuming $X = LR$ where $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ (and hence X is rank k), and optimize for L and R.

Non-negative matrix factorization (NMF)

Application 3: Matrix Completion

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.
- Goal: Find the *simplest* matrix X s.t $A_j(X) \approx y_j, \forall j \in 1, \cdots, n$
- Optimization Problem: Matrix completion optimization problem is:

$$\min_X \sum_{j=1}^{n} ||y_j - A_j(X)||_2^2 + ||X||_*$$

- Can be solved using Iterative Singular Value Thresholding!
- Another way is to explicitly model this is by assuming $X = LR$ where $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ (and hence X is rank k), and optimize for L and R.

Application 3: Matrix Completion

- Data: Given observations $y_1, \cdots, y_n$, such that each $y_j = A_j(X)$ where A_j could be a single element or a combination of elements in $X \in \mathbb{R}^{m \times n}$. Consider for example X being product recommendation matrix.
- Goal: Find the *simplest* matrix X s.t $A_j(X) \approx y_j, \forall j \in 1, \cdots, n$
- Optimization Problem: Matrix completion optimization problem is:

$$\min_X \sum_{j=1}^{n} ||y_j - A_j(X)||_2^2 + ||X||_*$$

- Can be solved using Iterative Singular Value Thresholding!
- Another way is to explicitly model this is by assuming $X = LR$ where $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ (and hence X is rank k), and optimize for L and R.

Non-negative matrix factorization (NMF)

Application 4: Low Rank and Non Negative Matrix Factorization

- Goal: Find matrices L, R with $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ s.t $A_j(LR) \approx y_j, \forall j \in 1, \cdots, n$ so that X has low rank.

Application 4: Low Rank and Non Negative Matrix Factorization

- Goal: Find matrices L, R with $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ s.t $A_j(LR) \approx y_j, \forall j \in 1, \cdots, n$ so that X has low rank.

Even in regression, if we wanted weights w's to be positive, there is an approach called projected gradient descent

ML1-LinearRegression.ipynb
https://colab.research.google.com/drive/13ILGLUU_k4VS_tCpsMivLc9nnvOluxAN#scrollTo=PyeQbbkjdE6v

1. The update equations are

$$\begin{pmatrix} w^{new} \\ b^{new} \end{pmatrix} = \begin{pmatrix} w^{old} \\ b^{old} \end{pmatrix} - \eta \nabla mse(w^{old}, b^{old})$$

2. Here, $\nabla mse(w^{old}, b^{old})$ denotes the 'gradient' vector $\nabla mse(w, b)$ evaluated at w^{old}, b^{old}. Further, $\nabla mse(w, b)$ is defined as

$$\nabla mse(w, b) = \begin{pmatrix} \frac{\partial mse(w,b)}{\partial w} \\ \frac{mse(w,b)}{\partial b} \end{pmatrix}$$

Application 4: Low Rank and Non Negative Matrix Factorization

- Goal: Find matrices L, R with $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ s.t $A_j(LR) \approx y_j, \forall j \in 1, \cdots, n$ so that X has low rank.
- Optimization Problem: Matrix Factorization optimization problem is:

$$\min_{L,R} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$$

No need of matrix regularization.

Application 4: Low Rank and Non Negative Matrix Factorization

- Goal: Find matrices L, R with $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ s.t $A_j(LR) \approx y_j, \forall j \in 1, \cdots, n$ so that X has low rank.
- Optimization Problem: Matrix Factorization optimization problem is:

$$\min_{L,R} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$$

No need of matrix regularization.

- We can also add non-negativity constraints and this becomes non-negative Matrix Factorization: $\min_{L,R:L \geq 0, R \geq 0} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$

Application 4: Low Rank and Non Negative Matrix Factorization

- Goal: Find matrices L, R with $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ s.t $A_j(LR) \approx y_j, \forall j \in 1, \cdots, n$ so that X has low rank.
- Optimization Problem: Matrix Factorization optimization problem is:

$$\min_{L,R} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$$

& add projection step to L & R

No need of matrix regularization.

- We can also add non-negativity constraints and this becomes non-negative Matrix Factorization: $\min_{L,R:L \geq 0, R \geq 0} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$

In several algorithms such as the one we will see, the projection step is much neater and results from some nice rescaling - satisfy the constrained optimization (KKT later) conditions

Application 4: Low Rank and Non Negative Matrix Factorization

- Goal: Find matrices L, R with $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ s.t $A_j(LR) \approx y_j, \forall j \in 1, \cdots, n$ so that X has low rank.
- Optimization Problem: Matrix Factorization optimization problem is:

$$\min_{L,R} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$$

No need of matrix regularization.

- We can also add non-negativity constraints and this becomes non-negative Matrix Factorization: $\min_{L,R:L\geq 0,R\geq 0} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$

This is projected/proximal gradient descent

L_0, R_0 Initialize to non-negative values

iteratively project to L, R>=0

$$[L_t]_{[i,j]} = [L_{t-1}]_{[i,j]} \frac{[L_0 R_0 R_{t-1}^T]_{[i,j]}}{[L_{t-1} R_{t-1} R_{t-1}^T]_{[i,j]}}$$

$$[R_t]_{[i,j]} = [R_{t-1}]_{[i,j]} \frac{[L_t^T L_0 R_0]_{[i,j]}}{[L_t^T L_t R_{t-1}]_{[i,j]}}$$

L_0, R_0 initialization includes y_j & A_j

Application 4: Low Rank and Non Negative Matrix Factorization

- Goal: Find matrices L, R with $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ s.t $A_j(LR) \approx y_j, \forall j \in 1, \cdots, n$ so that X has low rank.
- Optimization Problem: Matrix Factorization optimization problem is:

$$\min_{L,R} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$$

No need of matrix regularization.

- We can also add non-negativity constraints and this becomes non-negative Matrix Factorization: $\min_{L,R:L \geq 0, R \geq 0} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$
- Can be solved using Iterative Singular Value Thresholding!

Application 4: Low Rank and Non Negative Matrix Factorization

- Goal: Find matrices L, R with $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ s.t $A_j(LR) \approx y_j, \forall j \in 1, \cdots, n$ so that X has low rank.
- Optimization Problem: Matrix Factorization optimization problem is:

$$\min_{L,R} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$$

No need of matrix regularization.

- We can also add non-negativity constraints and this becomes non-negative Matrix Factorization: $\min_{L,R:L \geq 0, R \geq 0} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$
- Can be solved using Iterative Singular Value Thresholding!

Can have nuclear norm for L and R
if they must have low rank as well!!!

Application 4: Low Rank and Non Negative Matrix Factorization

- Goal: Find matrices L, R with $L \in \mathbb{R}^{m \times k}$ and $R \in \mathbb{R}^{k \times n}$ s.t $A_j(LR) \approx y_j, \forall j \in 1, \cdots, n$ so that X has low rank.
- Optimization Problem: Matrix Factorization optimization problem is:

$$\min_{L,R} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$$

No need of matrix regularization.

- We can also add non-negativity constraints and this becomes non-negative Matrix Factorization: $\min_{L,R:L\geq 0,R\geq 0} \sum_{j=1}^{n} ||y_j - A_j(LR)||_2^2$
- Can be solved using Iterative Singular Value Thresholding!
- Sometimes Y is fully observed and we want a non-negative low rank factorization of $Y \approx LR$. The optimization problem is: $\min_{L,R:L\geq 0,R\geq 0} \sum_{j=1}^{n} ||Y - LR||_2^2$.

Application 5: Clustering

- This is an instance of unsupervised learning.

Application 5: Clustering

- This is an instance of unsupervised learning.
- Data: Given unsupervised data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ is the feature vectors.

Application 5: Clustering

Continuous relaxation.

- This is an instance of unsupervised learning.
- Data: Given unsupervised data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ is the feature vectors.

Simplicity for clustering?

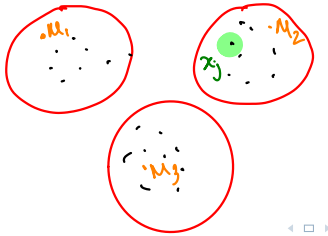
$$\min \|\mu_i - P_{ij} x_j\|^2$$

$$P_{ij} \in \{0, 1\}$$

Suggestion: $\sum_i P_{ij} - 1 = 0$?

$$P_{ij} \in [0, 1]$$

The idea behind soft-k-means clustering



$$P_{ij}^t \propto \left[d(x_j, \mu_i^{t-1}) \right]^{-1}$$

$$\mu_i^t = \sum_j P_{ij}^t x_j$$

Application 5: Clustering

- This is an instance of unsupervised learning.
- Data: Given unsupervised data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ is the feature vectors.
- Goal: Find clusters (sets) $C_1, C_2, \cdots, C_k$ with each cluster consisting of *similar* instances. Denote $V = \{1, \cdots, n\}$. Then $\cup_{i=1}^{k} C_i = V$.

Application 5: Clustering

- This is an instance of unsupervised learning.
- Data: Given unsupervised data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ is the feature vectors.
- Goal: Find clusters (sets) $C_1, C_2, \cdots, C_k$ with each cluster consisting of *similar* instances. Denote $V = \{1, \cdots, n\}$. Then $\cup_{i=1}^{k} C_i = V$.

$C_m \cap C_n = \emptyset$
$\forall\ m \neq n$

What if we want membership P such that the data is partitioned

$P_{ij} \in \{0, 1\}$

Will be discrete (+ continuous) optimization problems

Application 5: Clustering

- This is an instance of unsupervised learning.
- Data: Given unsupervised data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ is the feature vectors.
- Goal: Find clusters (sets) $C_1, C_2, \cdots, C_k$ with each cluster consisting of *similar* instances. Denote $V = \{1, \cdots, n\}$. Then $\cup_{i=1}^{k} C_i = V$.
- Optimization Problem: The k-means optimization problem is:

$$\min_{C_1, C_2, \cdots, C_k} \sum_{i=1}^{k} \sum_{x \in C_i} |x - 1/|C_i| \sum_{x_j \in C_i} x_j|_2^2$$

Application 5: Clustering

- This is an instance of unsupervised learning.
- Data: Given unsupervised data $\{x_1, x_2, \cdots, x_n\}$ where $x_i \in \mathbf{R}^m$ is the feature vectors.
- Goal: Find clusters (sets) $C_1, C_2, \cdots, C_k$ with each cluster consisting of *similar* instances. Denote $V = \{1, \cdots, n\}$. Then $\cup_{i=1}^{k} C_i = V$.
- Optimization Problem: The k-means optimization problem is:

$$\min_{C_1, C_2, \cdots, C_k} \sum_{i=1}^{k} \sum_{x \in C_i} |x - 1/|C_i| \sum_{x_j \in C_i} x_j|_2^2$$

- This problem can actually be viewed as a joint discrete and continuous problem.

Backup slides on Lasso

Regularization for Generalizability

Recall: Complex models could lead to overfitting. How to counter?
Regularization: The main idea is to modify the error function so that model complexity is also explicitly penalized

$$\text{Loss}_{\text{reg}}(\mathbf{w}) = \text{Loss}_{\mathcal{D}}(\mathbf{w}) + \lambda \cdot \text{Reg}(\mathbf{w})$$

A squared penalty on the weights, i.e. $\text{Reg}(\mathbf{w}) = ||\mathbf{w}||^2$ is a popular penalty function and is known as L_2 regularization.

MAP objective and regularization

Bayesian view of regularization: Regularization can be achieved using different types of priors on the parameters

$$\mathbf{w}_{\text{MAP}} = \mathbf{w} \frac{1}{2\sigma^2} \sum_j (y_j - \mathbf{w}^T \mathbf{x}_j)^2 + \frac{\lambda}{2} ||\mathbf{w}||_2^2$$

- We get an L_2 regularized solution for the linear regression problem using a Gaussian prior on the weights.
- What happens when $||\mathbf{w}||_2^2$ is replaced with $||\mathbf{w}||_1$? Contrast their level curves!

Contrasting Level Curves

https://colab.research.google.com/drive/1mQRvnbZpDWxZa2I6c92lMbqD32mD-YXP#scrollTo=gaglj9Wi1cmg

lambda_1e-15	0.96	0.22	1.1	-0.37	0.00089	0.0016	-0.00012	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.4e-09	1.8e-10	-2e-10	-9.2e-11
lambda_1e-10	0.96	0.22	1.1	-0.37	0.00088	0.0016	-0.00012	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.4e-09	1.8e-10	-2e-10	-9.2e-11
lambda_1e-08	0.96	0.22	1.1	-0.37	0.00077	0.0016	-0.00011	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.3e-09	2e-10	-1.9e-10	-9.3e-11
lambda_1e-05	0.96	0.5	0.6	-0.13	-0.038	-0	0	0	0	7.7e-06	1e-06	7.7e-08	0	0	0	-0	-7e-11
lambda_0.0001	1	0.9	0.17	-0	-0.048	-0	-0	0	0	9.5e-06	5.1e-07	0	0	0	-0	-0	-4.4e-11
lambda_0.001	1.7	1.3	-0	-0.13	-0	-0	-0	0	0	0	0	0	1.5e-08	7.5e-10	0	0	0
lambda_0.01	3.6	1.8	-0.55	-0.00056	-0	-0	-0	-0	-0	-0	-0	0	0	0	0	0	0
lambda_1	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0
lambda_5	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0
lambda_10	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0

Lasso Regularized Least Squares Regression

- The general **Penalized Regularized L.S Problem**:

$$\mathbf{w}_{Reg} =_{\mathbf{w}} \ ||\Phi\mathbf{w} - \mathbf{y}||_2^2 + \lambda\Omega(\mathbf{w})$$

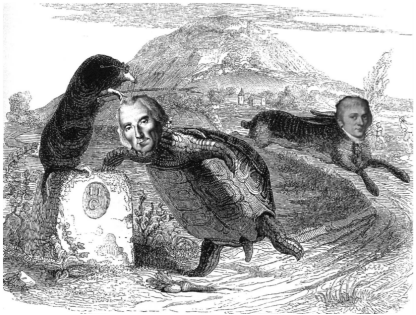
  - $\Omega(\mathbf{w}) = ||\mathbf{w}||_1 \Rightarrow$ **Lasso**
- Lasso Regression

$$\mathbf{w}_{lasso} =_{\mathbf{w}} \ ||\Phi\mathbf{w} - \mathbf{y}||_2^2 + \lambda||\mathbf{w}||_1$$

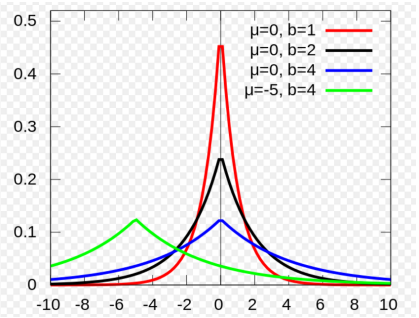
- Lasso is the MAP estimate of Linear Regression subject to Laplace Prior on $\mathbf{w} \sim Laplace(0, \theta)$

$$Laplace(w_i \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|w_i - \mu|}{b}\right)$$

Gaussian Hare vs. Laplacian Tortoise



- Gaussian easier to estimate



- Laplacian yields more sparsity

Lasso: Iterative Soft Thresholding Algorithm (ISTA)

The **LASSO Regularized L.S Problem**:

$$\mathbf{w}_{Lasso} =_{\mathbf{w}} \; E_{Lasso}(\mathbf{w}) =_{\mathbf{w}} \; E_{LS}(\mathbf{w}) + \lambda|\mathbf{w}|_1$$

where $E_{LS}(\mathbf{w}) = ||\Phi\mathbf{w} - \mathbf{y}||_2^2$

- while relative drop in $E_{Lasso}(\mathbf{w}^t)$ across $t = k$ and $t = k+1$ is significant:
  - *LS Iterate:* $\mathbf{w}_{LS}^{k+1} = \mathbf{w}_{Lasso}^{k} - \eta\nabla E_{LS}(\mathbf{w}_{Lasso}^{k})$
  - *Proximal*[1] *Step:*

$$\left[\mathbf{w}_{Lasso}^{k+1}\right]_i = \begin{cases} \left[\mathbf{w}_{LS}^{k+1}\right]_i - \lambda\eta & \text{if } \left[\mathbf{w}_{LS}^{k+1}\right]_i > \lambda\eta \\ \left[\mathbf{w}_{LS}^{k+1}\right]_i + \lambda\eta & \text{if } \left[\mathbf{w}_{LS}^{k+1}\right]_i < -\lambda\eta \\ 0 & \text{otherwise} \end{cases}$$

[1] See slide 1 of https://www.cse.iitb.ac.in/~cs709/notes/enotes/24-23-10-2018-generalized-proximal-projected-gradientdescent-examples-geometry-convergence-accelerated-annotated.pdf

Note how LASSO yields greater sparsity

https://colab.research.google.com/drive/1mQRvnbZpDWxZa2I6c92IMbqD32mD-YXP#scrollTo=gaglj9Wi1cmg

lambda_1e-15	0.96	0.22	1.1	-0.37	0.00089	0.0016	-0.00012	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.4e-09	1.8e-10	-2e-10	-9.2e-11
lambda_1e-10	0.96	0.22	1.1	-0.37	0.00088	0.0016	-0.00012	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.4e-09	1.8e-10	-2e-10	-9.2e-11
lambda_1e-08	0.96	0.22	1.1	-0.37	0.00077	0.0016	-0.00011	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.3e-09	2e-10	-1.9e-10	-9.3e-11
lambda_1e-05	0.96	0.5	0.6	-0.13	-0.038	-0	0	0	0	7.7e-06	1e-06	7.7e-08	0	0	0	-0	-7e-11
lambda_0.0001	1	0.9	0.17	-0	-0.048	-0	-0	0	0	9.5e-06	5.1e-07	0	0	0	-0	-0	-4.4e-11
lambda_0.001	1.7	1.3	-0	-0.13	-0	-0	-0	0	0	0	0	0	1.5e-08	7.5e-10	0	0	0
lambda_0.01	3.6	1.8	-0.55	-0.00056	-0	-0	-0	-0	-0	-0	-0	0	0	0	0	0	0
lambda_1	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0
lambda_5	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0
lambda_10	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0

Backup slides on Dimensionality reduction: PCA & Kernel PCA

Model selection

Occam's Razor: All other things being equal, pick the simplest solution.
What is simple?

Why is simple better?

Dimensionality Reduction

Dimensionality reduction is a feature extraction technique where you want to represent input data with fewer dimensions[2]

Discovering true intrinsic dimensionality of the data

Fewer dimensions $\Rightarrow$ Potentially easier to learn, and lesser chances of overfitting

[2] Also connected to optional material on VC dimensions

Dimensionality reduction using linear projections

Consider an input example $\mathbf{x} \in \mathbb{R}^d$.
We want to reduce its dimensionality and project it to a k-dimensional vector $\mathbf{z}$ ($k << d$).

$$\mathbf{z} = \mathbf{U}^T \mathbf{x}$$

where $\mathbf{U}$ is a $d \times k$ projection matrix.

Dimensionality reduction

Principal Components Analysis (PCA): Simple Illustration

Dimensionality reduction using linear projections

Consider an input example $\mathbf{x} \in \mathbb{R}^d$.
We want to reduce its dimensionality and project it to a k-dimensional vector $\mathbf{z}$ ($k << d$).

$$\mathbf{z} = \mathbf{U}^T \mathbf{x}$$

where $\mathbf{U}$ is a $d \times k$ projection matrix.
How do we find $\mathbf{U}$? What are PCA's objectives?

PCA: Minimizing reconstruction error

Project down $\mathbf{x}$ to $\mathbf{z}$: $\mathbf{z} = \mathbf{U}^T\mathbf{x}$
Reconstruct $\hat{\mathbf{x}}$ using $\mathbf{U}$: $\hat{\mathbf{x}} = \mathbf{U}\mathbf{z}$
Reconstruction error $||\mathbf{x} - \hat{\mathbf{x}}||$ should be small
Objective:

$$\min_{\mathbf{U}} \sum_i ||\mathbf{x}_i - \mathbf{U}\mathbf{U}^T\mathbf{x}_i||^2$$

PCA: Error Reduction Perspective

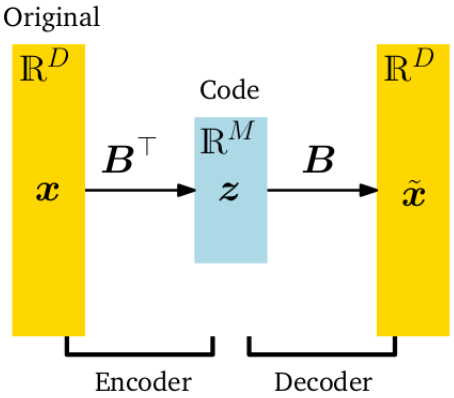


Figure: Illustration of Error Reduction Perspective of PCA. Picture courtesy: "Mathematics for Machine Learning", Marc Peter Deisenroth, A. Aldo Faisal & Cheng Soon Ong

PCA: Maximizing variance of projected data

Assume the data points are centered i.e. $\mathbb{E}[\mathbf{x}] = 0$.
Variance of projected data: $\mathbb{E}[||\mathbf{U}^T\mathbf{x}||^2]$
Objective:

$$\max_{\mathbf{U}, \mathbf{U}^T\mathbf{U}=\mathbf{I}} \mathbb{E}[||\mathbf{U}^T\mathbf{x}||^2]$$

Minimizing reconstruction error is equivalent to maximizing projected variance (Prove in Tutorial 6)

Thus, another perspective to PCA is that

- it finds a rotation of the original coordinate system in the form of $\mathcal{P}$
- and expresses $\mathbf{X}$ in that system $\mathcal{P}$ so that
- each new coordinate expresses as much as possible of the variability in $\mathbf{X}$ as can be expressed by a linear combination of the n entries of $\mathbf{X}$.

This has application in data transformation, feature discovery, feature selection and so on.

PCA: Direction with Maximal Variance

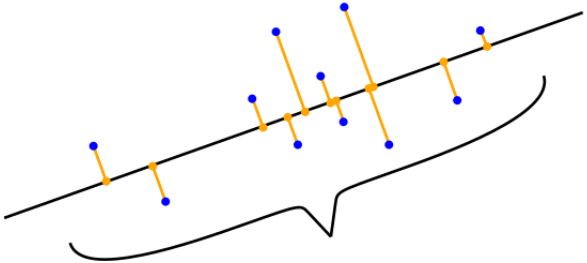


Figure: Illustration of PCA wrt Direction with Maximal Variance. Picture courtesy: "Mathematics for Machine Learning", Marc Peter Deisenroth, A. Aldo Faisal & Cheng Soon Ong

Maximizing variance of projected data

Finding one *principal component*:

$$\max_{||\mathbf{u}||=1} \mathbb{E}[(\mathbf{u}^T\mathbf{x})^2]$$

Maximizing variance of projected data

Finding one *principal component*:

$$\max_{||\mathbf{u}||=1} \mathbb{E}[(\mathbf{u}^T\mathbf{x})^2]$$

$$= \max_{||\mathbf{u}||=1} \frac{1}{n}||\mathbf{u}^T\mathbf{X}||^2$$

Maximizing variance of projected data

Finding one *principal component*:

$$\max_{||\mathbf{u}||=1} \mathbb{E}[(\mathbf{u}^T\mathbf{x})^2]$$
$$= \max_{||\mathbf{u}||=1} \frac{1}{n}||\mathbf{u}^T\mathbf{X}||^2$$
$$= \max_{||\mathbf{u}||=1} \mathbf{u}^T\left(\frac{1}{n}\mathbf{X}\mathbf{X}^T\right)\mathbf{u}$$

Maximizing variance of projected data

Finding one *principal component*:

$$\begin{aligned}
&\max_{||\mathbf{u}||=1} \mathbb{E}[(\mathbf{u}^T\mathbf{x})^2] \\
&= \max_{||\mathbf{u}||=1} \frac{1}{n}||\mathbf{u}^T\mathbf{X}||^2 \\
&= \max_{||\mathbf{u}||=1} \mathbf{u}^T\left(\frac{1}{n}\mathbf{X}\mathbf{X}^T\right)\mathbf{u} \\
&= \max_{||\mathbf{u}||=1} \mathbf{u}^T\mathbf{S}\mathbf{u}
\end{aligned}$$

Maximizing variance of projected data

Finding one *principal component*:

$$\begin{aligned}
&\max_{||\mathbf{u}||=1} \mathbb{E}[(\mathbf{u}^T\mathbf{x})^2] \\
&= \max_{||\mathbf{u}||=1} \frac{1}{n}||\mathbf{u}^T\mathbf{X}||^2 \\
&= \max_{||\mathbf{u}||=1} \mathbf{u}^T\left(\frac{1}{n}\mathbf{X}\mathbf{X}^T\right)\mathbf{u} \\
&= \max_{||\mathbf{u}||=1} \mathbf{u}^T\mathbf{S}\mathbf{u}
\end{aligned}$$

Maximizing variance of projected data

Finding one *principal component*:

$$\begin{aligned}
&\max_{||\mathbf{u}||=1} \mathbb{E}[(\mathbf{u}^T\mathbf{x})^2] \\
&= \max_{||\mathbf{u}||=1} \frac{1}{n}||\mathbf{u}^T\mathbf{X}||^2 \\
&= \max_{||\mathbf{u}||=1} \mathbf{u}^T\left(\frac{1}{n}\mathbf{X}\mathbf{X}^T\right)\mathbf{u} \\
&= \max_{||\mathbf{u}||=1} \mathbf{u}^T\mathbf{S}\mathbf{u}
\end{aligned}$$

where $\mathbf{S}$ is the covariance matrix of the data.

Finding the first principal component

The first principal component is the top eigenvector of $\mathbf{S}$ with the largest eigenvalue

PCA: Direction with Maximal Variance

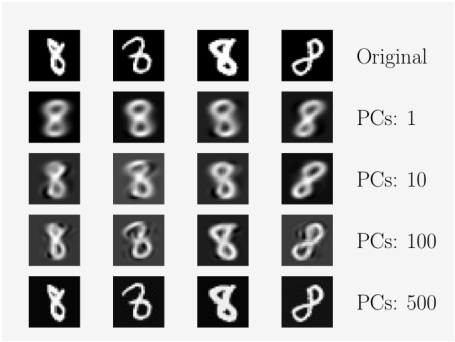


Figure: Illustration of PCA: Compressed vs. Non-compressed datasets. Picture courtesy: "Mathematics for Machine Learning", Marc Peter Deisenroth, A. Aldo Faisal & Cheng Soon Ong

Well-known use of PCA using all Eigenvectors/Eigenfaces

The PCA Algorithm

1. Compute mean of datapoints, $\bar{\mathbf{x}}$.
2. Mean-center the data to get a resulting data matrix, $\mathbf{X}$.
3. Compute the covariance matrix of data, $\mathbf{S}$

$$\mathbf{S} = \frac{1}{n}\mathbf{X}\mathbf{X}^T$$

4. Do eigenvalue decomposition of $\mathbf{S}$. Take the **top k** eigenvectors, $\mathbf{u}_1, \ldots, \mathbf{u}_k$.
5. $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_k]$ is the projection matrix.

The PCA Algorithm

1. Compute mean of datapoints, $\bar{\mathbf{x}}$.
2. Mean-center the data to get a resulting data matrix, $\mathbf{X}$.
3. Compute the covariance matrix of data, $\mathbf{S}$

$$\mathbf{S} = \frac{1}{n}\mathbf{X}\mathbf{X}^T$$

4. Do eigenvalue decomposition of $\mathbf{S}$. Take the **top k** eigenvectors, $\mathbf{u}_1, \ldots, \mathbf{u}_k$.
5. $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_k]$ is the projection matrix.

How would you 'Kernelize' PCA?

The Kernel PCA Algorithm

1. Consider the Singular Value Decomposition of $X = ADB^T$. Then $nS = XX^T = AD^2A^T$ and $X^TX = BD^2B^T$
2. $U = AD$ is the matrix of principal component variables.
3. Consider the kernel/gram matrix $\mathcal{K}$ of the data, in place of $\mathbf{X}^T\mathbf{X}$

$$\mathcal{K} = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \dots & K(\mathbf{x}_1, \mathbf{x}_n) \\ \dots & K(\mathbf{x}_i, \mathbf{x}_j) & \dots \\ K(\mathbf{x}_m, \mathbf{x}_1) & \dots & K(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

4. Then, $nS = XX^T = AD^2A^T$ and $\mathcal{K} = X^TX = BD^2B^T$ and the projections XU of our data X onto those components $U = AD$ can be computed from the eigendecomposition of $\mathcal{K}$
5. Do eigenvalue decomposition of $\mathcal{K}$. Take the eigenvectors, $\mathbf{v}_1, \dots, \mathbf{v}_k$.
6. Kernel-PCA computes not the principal components themselves, but the projections of our data onto those components:

- See Extra & Optional slides on Mercer's Theorem and RKHS Kernel for conditions under which $K(\mathbf{x}_i, \mathbf{x}_j)$ will be a **valid Kernel function**
- For more discussion on Kernel PCA, see Section 14.5.4 of the Tibshirani book posted on moodle.