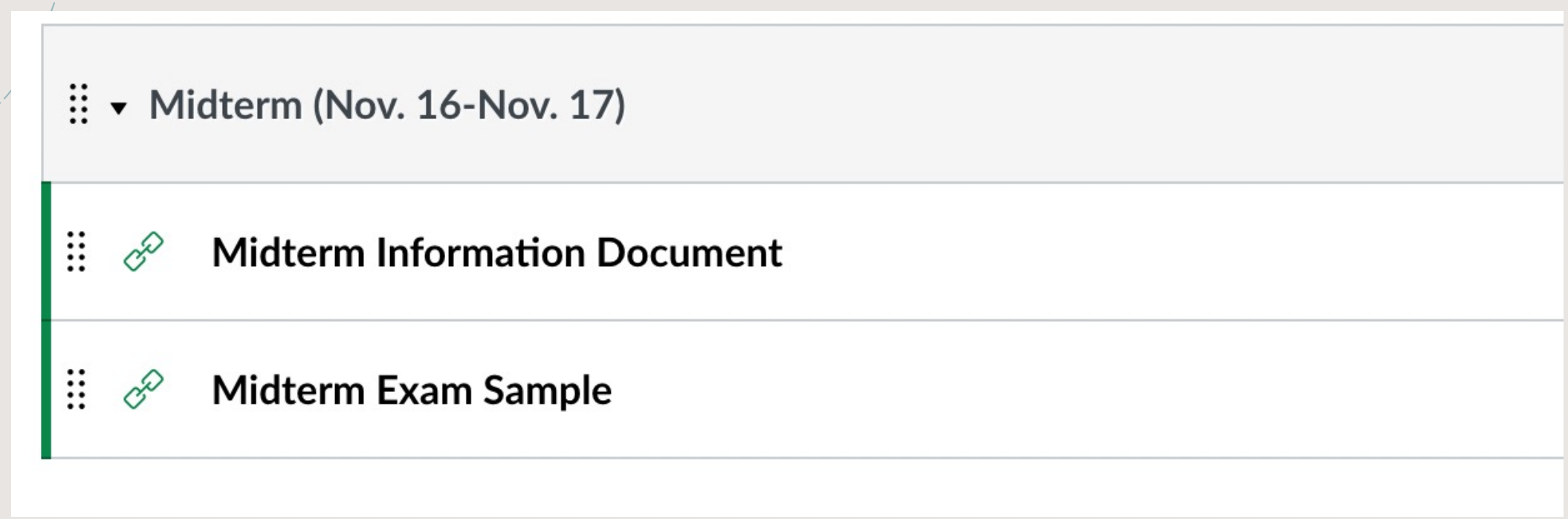
Midterm Recap

Instructor: Hui Guan

Outline

- Midterm policies and details (Posted on CANVAS)
  - https://docs.google.com/document/d/1nB9lFiPHM1AqZrO5HXhUe3xGdpd94QKQa1oBnIkhhZE/edit?usp=sharing
- Midterm recap

Also on CANVAS (will include this slides as well)



Midterm Policy

- Midterm is **Closed book**. No electronic device allowed. No need to use a calculator for the midterm.
- Plagiarism will not be tolerated and will result in **zero** grade for the midterm exam. Total marks is 100. The weight of the midterm exam grade for the entire course is 30%, as specified in the course syllabus.
- You will be given 50 T/F statements and each T/F statement is worth 2 points. Your answer should be either **True or False**.
- There is **NO** need to give any justification for your answer to avoid graders mis-interpreting your justification. Instead, the grade of **the last X T/F statements** that are considered poorly received by the entire class will be dropped (everyone gets the points for these statements no matter whether your answers are correct or not). The exact value of X is determined by the teaching stuff after the midterm.

Content coverage

- The midterm checks everything we covered in the lecture before the midterm exam. This include:
  - Chapter 2: Zero-order optimization
  - Chapter 3: First-order optimization
  - Chapter 5: Linear regression
  - Chapter 6: Linear two-class classification
  - Chapter 7: Linear multi-class classification
  - Chapter 8: Linear unsupervised learning
  - Chapter 9: Feature engineering and selection
  - Chapter 10: Principles of nonlinear feature engineering
  - Chapter 11: Principles of feature learning
  - Chapter 12: Kernel methods

Content coverage

- You can get the textbook from this github repo. We are using the 2nd edition.
- Any subsections in these chapters we didn't cover in the lectures won't be in the midterm either. You can find the course schedule on our course website: https://sites.google.com/umass.edu/compsci589-fall23/home

Regular Exam

Date and Time	Thu, Nov 16 2023 **5:30 PM - 6:45 PM**	*Online students will **have an extra 15 min** to take a picture of the solution and upload to gradescope.
Location	In person: • Morrill Sci Center II Room 131 • Morrill Sci Ctr I – Room N375	*Double seating; *Room assignment will be announced before the midterm. *Each room has two TAs to proctor the exam
	Online section: • *Use the office hour zoom link (available on CANVAS)* • *During the exam, make sure the **Camera + Audio is ON.*** • ***Exam questions will be emailed to you around 5:30 PM, Nov. 16***	*Only students registered in the online section are allowed to take midterm online! **Arrive at least 5 min before the midterm so that TA can confirm your name and email address!** *No need to print out the exam question. Prepare a white paper and write the answer in the white paper; submit answers only * In case any Q/A is needed, speak to the TA with the question. **DON'T TYPE in the chat.** *Please ensure you have a good internet connection to ensure the camera and audio is stable and clear.

Makeup exam 1

Makeup Midterm 1		*This makeup exam happens because of an exam conflict for a significant group of students in the class. Please offer your thanks to the TA Chhandak for willing to proctor the makeup exam.
Date and Time	Friday, Nov 17th, 2023 **5:15 PM - 6:30 PM**	***Students need permissions to participate in the makeup midterm**
Location	In person: • CS142	*TA Chhandak Bagchi will proctor the makeup exam. *The room will be locked before the TA opens the door.

Makeup exam 2 (online only)

Makeup Midterm 2 (Online section only)		*This makeup exam is for online students who cannot make to the regular exam time due to time zone difference. **Instructor approval required to take this exam.**
Date and Time	TBD	
Location	• *Use the office hour zoom link (available on CANVAS)* • *During the exam, make sure the **Camera + Audio is ON.*** • *Exam questions will be available on Canvas at the start of the exam*	***Arrive at least 5 min before the midterm so that TA can confirm your name and email!**

Tips

- Why?
  - What is the motivation of the approach?
- How?
  - How does the approach work?
  - What is the fundamental assumption of the approach?
- Then what?
  - Pros and cons?

Chapter 2: Zero-order optimization

+Minima and Maxima

+The Zero-Order Condition for Optimality

+Global optimization methods

+Local optimization methods

+Random search

+Coordinate search and descent

Zero-Order Condition for Optimality

The zero order condition for optimality: A point $\mathbf{w}^\star$ is

- - a global minimum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \leq g(\mathbf{w})$ for all $\mathbf{w}$
- - a global maximum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \geq g(\mathbf{w})$ for all $\mathbf{w}$
- - a local minimum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \leq g(\mathbf{w})$ for all $\mathbf{w}$ near $\mathbf{w}^\star$
- - a local maximum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \geq g(\mathbf{w})$ for all $\mathbf{w}$ near $\mathbf{w}^\star$

Why are these called *zero-order* conditions? Because each of their definitions involves only a function itself - and nothing else.

Global optimization methods

+In this Section we describe the first approach one might take to approximately minimize an arbitrary function:

- **evaluate the function using a large number of input points**
- **treat the input that provides the lowest function value as the approximate global minimum of the function.**

random / grid sampling / uniform

+Random sampling vs. Uniform sampling

Curse of dimensionality

- The global optimization approach fails quickly as we try to tackle functions of larger dimensional input.
- **As the dimension of the input space *increases* the density of our sampling decreases exponentially.**

Local Optimization Methods

- *Local optimization methods* work by taking a single sample input and refine it **sequentially**, driving it towards an approximate minimum point.
- Local optimization methods are by far the **most popular mathematical optimization schemes used in machine learning today**.
- Random local search, coordinate descent/search

Steplength parameter

- Because of this potential problem many local optimization schemes come equipped with what is called a *steplength parameter* (also called a *learning rate* parameter).
- We control this value, and it helps us more **directly control the length of each update step** (hence the name *steplength parameter*).

- A local optimization algorithm - *random local search*.
  - we look locally around the current point in a fixed number of random directions for a point that has a lower evaluation,
  - and if we find one we move to it.

② hyperparameter tuning $g(w) + \lambda h(w)$

① $g(w) = \|X^T w - y\| + \lambda \|w\|_0$

- The coordinate search and descent algorithms
  - zero order local methods that get around the inherent scaling issues of random search
  - **restrict the set of search directions to the coordinate axes of the input space.**
- The concept is simple: random search was designed to minimize a function $g(w_1, w_2, \ldots, w_N)$

  with respect to all of its parameters

  simultaneously.

Chapter 3: First Order Optimization

+First order optimality condition

+Coordinate descent

+Compute gradients efficiently

+Gradient descent

+Issues with gradient descent

First order optimality condition

- This system of ***N*** equations is naturally referred to as the *first order system of equations (first-order optimality condition)*.
- We can write the first order system more compactly using gradient notation as

$$\nabla g\left(\mathbf{v}\right) = \mathbf{0}_{N\times 1}.$$

- The first order condition also equally characterizes *maxima* and *saddle points* of a function

- *local minima* or points that are the smallest with respect to their immediate neighbors, like the one around the input value ***w=2*** in the right panel
- *local and global maxima* or points that are the largest with respect to their immediate neighbors, like the one around the input value ***w=-2*** in the right panel
- *saddle points* like the one shown in the middle panel, that are neither maximal nor minimal with respect to their immediate neighbors
- *Taken together all such points are collectively referred to as stationary points or critical points*.

Coordinate descent

- The idea is this**: instead of trying to solve the system of equations *at once* we solve each partial derivative equation *one at-a-time***.
- This is often called *coordinate descent*, since in solving each we move along the coordinate axes coordinate-wise (one at-a-time).

Gradient descent

- Remember, a general local optimization method looks like

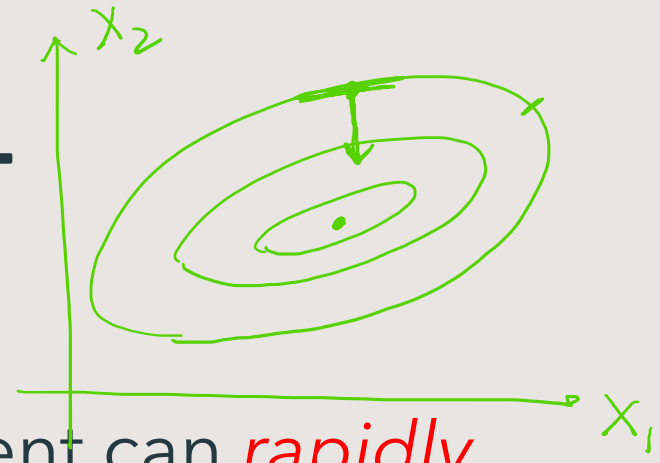
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{d}^k.$$

- Here $\mathbf{d}^k$ are *descent direction* vectors and α is called the *steplength* parameter. The basic form of gradient descent is:

$$\mathbf{d}^k = -\nabla g\left(\mathbf{w}^{k-1}\right)$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

Issues with gradient descent



- Issue 1: The *direction* of the negative gradient can *rapidly oscillate* or *zig-zag* during a run of gradient descent, often producing *zig-zagging* steps that take considerable time to reach a near mininum point.
- Issue 2: The *magnitude* of the negative gradient can *vanish rapidly* near stationary points, leading gradient descent to slowly crawl near minima and saddle points.

$$\vec{w} = \vec{w} + \alpha \vec{d}$$

$$\vec{d} = -g(w) = 0$$

$$\vec{d} = -\frac{g'(w)}{\|g'(w)\|} = -1/1$$

Chapter 5 Linear Regression

- Least Squares Linear Regression
- Least Absolute Deviations
- Regression Quality Metrics

MAE

MAD

$model(x, w) = x^T w$

$x^T w + w_0$

bias

feature touching weights

"regularization"

Linear regression

- *linear regression* -- the fitting of a representative line (or hyperplane in higher dimensions) to a set of input/output data points.
- Note: **each dimension of the *input*** is referred to in the jargon of machine learning as **a *feature*** or ***input feature***.
- So we will often refer to $w_{1,p}, w_{2,p}, \ldots, w_{N,p}$ as the *feature-touching weights* (the only weight *not* touching a feature is the bias).

Least Squares cost function

- Since we want all ***P*** such values to be small we can take their average - forming a *Least Squares* cost function for linear regression

$$g\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P} g_p\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P}\left(\dot{\mathbf{x}}_p^T\mathbf{w} - y_p\right)^2$$

- Note that the Least Squares cost function is not just a function of the weights $\mathbf{w}$, but of the data as well.
- However, when we express the function in mathematical shorthand as $g\left(\mathbf{w}\right)$ we only show dependency on the weights $\mathbf{w}$.

LAD Cost

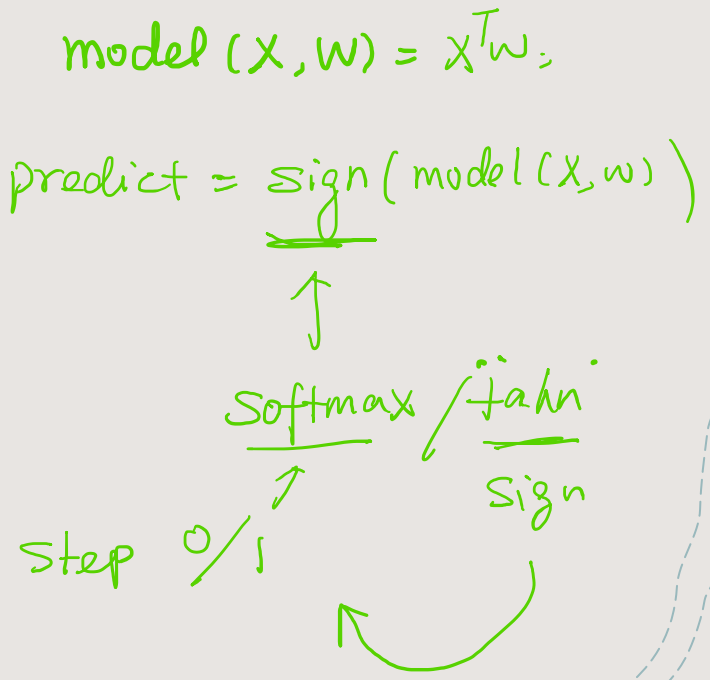
- A slight twist on the derivation of the Least Squares cost function that leads to an alternative cost for linear regression called *Least Absolute Deviations*.
- This alternative cost function is much in a dataset than the original Least Squares.

Quality Metric

- Mean Square Error
- Mean Absolute Error

Chapter 6 Linear classification

- Logistic regression and the cross entropy cost
- Logistic regression and the softmax cost
- Perceptron
- SVM
- Classification Quality Metrics



Linear classification

- This is the simplest sort of dataset with binary output we could aim to perform regression on - one with a *linear* boundary.
- More generally the boundary could certainly be nonlinear.
- We will deal with this more general potentiality later on - when discussing neural networks, trees, and kernel-based methods.
- But first we deal with the current scenario: **How can we perform regression on a dataset like the ones described in the figure above**?

Cross entropy cost

$y \in \{0, 1\}$

- To optimally tune the parameters $\mathbf{w}$ we want to *minimize* the Cross Entropy cost as

$$\underset{\mathbf{w}}{\text{minimize}} \; -\frac{1}{P}\sum_{p=1}^{P} y_p \log\left(\sigma\left(\dot{\mathbf{x}}_p^T\mathbf{w}\right)\right) + (1 - y_p)\log\left(1 - \sigma\left(\dot{\mathbf{x}}_p^T\mathbf{w}\right)\right)$$

The Cross Entropy cost is *always convex* regardless of the dataset used

Softmax cost

$y \in \{-1, 1\}$

- Taking the average of this point-wise cost over all *P* points we have a more common appearance of the *Softmax cost for logistic regression*

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P} g_p\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P}\log\left(1 + e^{-y_p \dot{\mathbf{x}}_p^T \mathbf{w}}\right)$$

- This cost function, like the Cross Entropy cost, is *always convex regardless of the dataset used*.

Perceptron cost

$y \in \{-1, 1\}$

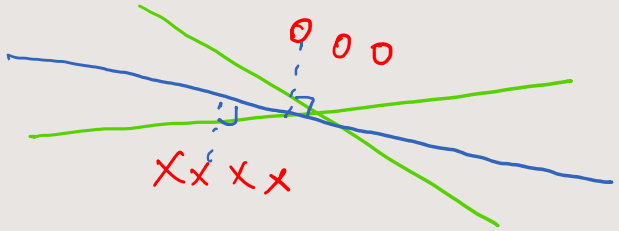
- Note that the expression $\max\left(0, -y_p \dot{\mathbf{x}}_p^T \mathbf{w}\right)$ is always nonnegative.
- The functional form of this point-wise cost $\max\left(0, \cdot\right)$ is called a *rectified linear unit (ReLU)*.
- Because these point-wise costs are nonnegative and equal *zero* when our weights are tuned correctly, we can take their average over the entire dataset to form a proper cost function as

$$g\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P} g_p\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P} \max\left(0, -y_p \dot{\mathbf{x}}_p^T \mathbf{w}\right).$$

softmax

This cost function goes by many names such as the *perceptron cost*, the *rectified linear unit cost* (or *ReLU cost* for short), and the *hinge cost*.

SVM cost



In order to find a separating hyperplane for the data with minimum length normal vector we can simply combine...

- ...our desire to minimize $\|\boldsymbol{\omega}\|_2^2$ ...
- ...subject to the constraint that the hyperplane perfectly separates the data (given by the margin criterion described above).
- This gives the so-called *hard-margin support vector machine* problem

$$\begin{cases} \underset{b,\,\boldsymbol{\omega}}{\text{minimize}} \ \|\boldsymbol{\omega}\|_2^2 \\ \text{subject to} \ \max\left(0,\, 1 - y_p\left(b + \mathbf{x}_p^T\boldsymbol{\omega}\right)\right) = 0, \quad p = 1, \ldots, P. \end{cases}$$

SVM cost

- This *regularized* form of the Margin-Perceptron cost function is referred to as the *soft-margin support vector machine cost*.

$$g\left(b, \boldsymbol{\omega}\right) = \sum_{p=1}^{P} \max\left(0,\, 1 - y_p\left(b + \mathbf{x}_p^T \boldsymbol{\omega}\right)\right) + \lambda \|\boldsymbol{\omega}\|_2^2$$

regularization

Quality Metrics

- Accuracy
- Balanced accuracy
- Confusion matrix

Chapter 6 Multi-class classification

- One-versus-All Multi-Class Classification ← "Fusion Rule"
- Multi-class classification and the Perceptron max → softmax
- Which approach produces the best results?
- Classification Quality Metrics
- Mini-batch training

OvA

One-versus-All multi-class classification

1: **Input:** multiclass dataset $\left\{\left(\mathbf{x}_p,\, y_p\right)\right\}_{p=1}^{P}$ where $y_p \in \{0, \ldots, C-1\}$, two-class classification scheme and optimizer

2: `for` $j = 0, \ldots, C-1$

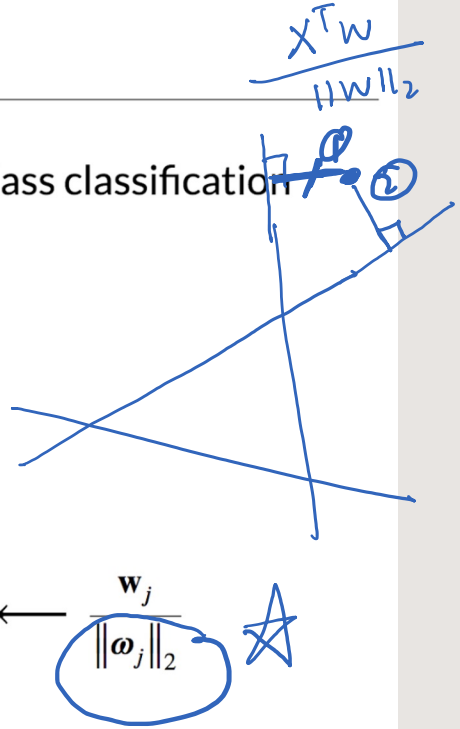
3: form temporary labels $\tilde{y}_p = \begin{cases} +1 & \text{if } y_p = j \\ -1 & \text{if } y_p \neq j \end{cases}$ ← one class ← rest classes

4: solve two-class subproblem on $\left\{\left(\mathbf{x}_p,\, \tilde{y}_p\right)\right\}_{p=1}^{P}$ to find weights $\mathbf{w}_j$

5: normalize classifier weights by magnitude of feature-touching portion $\mathbf{w}_j \longleftarrow \dfrac{\mathbf{w}_j}{\|\boldsymbol{\omega}_j\|_2}$

6: `end for`

7: To assign label y to a point $\mathbf{x}$, apply the fusion rule: $y = \underset{c=0,\ldots,C-1}{\operatorname{argmax}} \; \mathring{\mathbf{x}}^T \mathbf{w}_c$



Multi-class classification

- Taking the *average* of this point-wise cost over the entire dataset we have

$$g\left(\mathbf{w}_0, \ldots, \mathbf{w}_{C-1}\right) = \frac{1}{P}\sum_{p=1}^{P}\left[\left(\max_{c=0,\ldots,C-1} \dot{\mathbf{x}}_p^T\mathbf{w}_c\right) - \dot{\mathbf{x}}_p^T\mathbf{w}_{y_p}\right]$$

∴ softmax

gives us the **multi-class Perceptron** cost function

- The function is **convex**
- It has a **trivial solution at zero** (which is often avoided by initializing local optimization away from the origin).

Approximately solve it by *relaxing the constraints*

$$\frac{1}{P}\sum_{p=1}^{P}\left[\left(\max_{c=0,\ldots,C-1} b_c + \mathbf{x}_p^T\boldsymbol{\omega}_c\right) - \left(b_{y_p} + \mathbf{x}_p^T\boldsymbol{\omega}_{y_p}\right)\right] + \lambda\sum_{c=0}^{C-1}\|\boldsymbol{\omega}_c\|_2^2$$

- For simplicity we choose a single regularization parameter $\lambda \geq 0$ that is used to penalize the magnitude of all normal vectors simultaneously.
- Even though this regularized form will not necessarily guarantee that. $\|\boldsymbol{\omega}_c\|_2^2 = 1$ for all 'c', it will generally force the length of all normal vectors to behave well.

This cost function goes by *many* names in practice including: the **multi-class Softmax**, **Multiclass Cross Entropy**, **Softplus**, and **Multiclass Logistic** cost to name a few.

$$g\left(\mathbf{w}_0, \ldots, \mathbf{w}_{C-1}\right) = \frac{1}{P}\sum_{p=1}^{P}\left[\log\left(\sum_{c=0}^{C-1} e^{\dot{\mathbf{x}}_p^T \mathbf{w}_c}\right) - \dot{\mathbf{x}}_p^T \mathbf{w}_{y_p}\right].$$

As with the multi-class Perceptron, it is common to *regularize* the Multiclass Softmax via its feature touching weights as

$$\frac{1}{P}\sum_{p=1}^{P}\left[\log\left(\sum_{c=0}^{C-1}e^{b_c+\mathbf{x}_p^T\boldsymbol{\omega}_c}\right)-\left(b_{y_p}+\mathbf{x}_p^T\boldsymbol{\omega}_{y_p}\right)\right]+\lambda\sum_{c=0}^{C-1}\|\boldsymbol{\omega}_c\|_2^2$$

Classification Quality Metrics

- Accuracy
- Balanced accuracy
- Confusion matrix

Mini-batch training

- As opposed to a standard (also called full batch) local optimization technique that takes individual steps by sweeping through an *entire* set of training data *simultaneously*, the mini-batch approach has us take smaller steps sweeping through training data *sequentially*.
- One complete sweep through the data being referred to as an *epoch* of mini-batch learning.

Unsupervised learning

- Background: Fixed Spanning Sets, Orthonormality, and Projections
- The Linear Autoencoder and Principal Component Analysis
- K-means Clustering

(B) dimension reduction

Orthonormality

$C \in \mathbb{R}^{N \times K}$ K vectors

- Note that the same constraints expressed in terms of the concatenated basis matrix $\mathbf{C}$ can be written compactly as

$$\mathbf{C}^T\mathbf{C} = \mathbf{I}_{N\times N}.$$

- Often such a spanning sets are likewise referred to as *orthonormal* or - sometimes - *orthogonal*
- Technically speaking the latter term should only be applied when a set of vectors consists of perpendicular elements that are not necessarily unit-length, it is unfortunately often used to refer to unit-length perpendicular vectors as well.

PCA

- The most fundamental unsupervised learning method is known as Principal Component Analysis PCA for short.

$C \in \mathbb{R}^{N \times K}$ $X \in \mathbb{R}^{N \times P}$ $\vec{W} = C^T X \in \mathbb{R}^{K \times P}$

C

$g(W) = \| C(C^T X) - X \|_2^2$

$s.t \quad C^T C = I_{N \times N}$

- Learn the proper weights to best represent input data over a given fixed spanning set
- Learn a proper spanning set as well.

$C \sim$ eigenvectors of $XX^T \in \mathbb{R}^{N \times N}$

K-means clustering

The K-means algorithm

input: dataset $\mathbf{x}_1, \ldots, \mathbf{x}_P$, initializations for centroids $\mathbf{c}_1, \ldots, \mathbf{c}_K$, and maximum number of iterations J

`for` $j = 1, \ldots, J$

    `# Update cluster assignments`

    `for` $p = 1, \ldots, P$

        $a_p = \underset{k=1,\ldots,K}{\operatorname{argmin}} \|\mathbf{c}_k - \mathbf{x}_p\|_2$

    `end for`

    `# Update centroid locations`

    `for` $k = 1, \ldots, K$

        denote $\mathcal{S}_k$ the index set of points $\mathbf{x}_p$ currently assigned to the k^{th} cluster

        update $\mathbf{c}_k$ via $\mathbf{c}_k = \frac{1}{|\mathcal{S}_k|} \sum_{p \in \mathcal{S}_k} \mathbf{x}_p$

    `end for`

`end for`

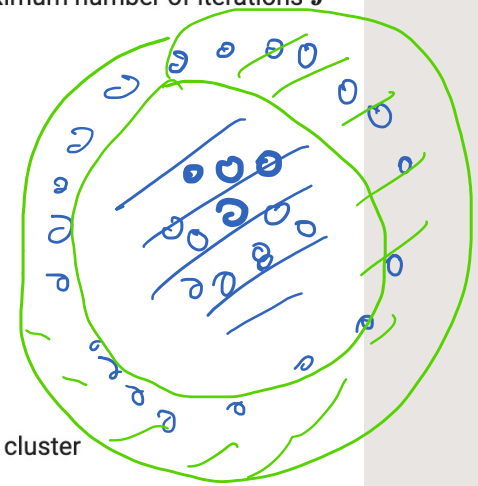
`# Update cluster assignments using final centroids`

`for` $p = 1, \ldots, P$

    $a_p = \underset{k=1,\ldots,K}{\operatorname{argmin}} \|\mathbf{c}_k - \mathbf{x}_p\|_2$

`end for`

output: optimal centroids and assignments



Issues with k-means

- Sensitive to initialization
- Sensitive to the choice of k

Chapter 9 Feature engineering and selection

- Histogram features (one-hot encoding)
- Feature scaling via standard normalization
- Feature scaling via PCA sphering
- Feature selection via Boosting
- Feature selection via Regularization

Standard normalization of classification data

$$\frac{x_{p,n} - \mu_n}{\sigma_n}$$

$$\mu_n = \frac{1}{P}\sum_{p=1}^{P} x_{p,n}$$

$$\sigma_n = \sqrt{\frac{1}{P}\sum_{p=1}^{P}\left(x_{p,n} - \mu_n\right)^2}.$$

Python implementation:
https://colab.research.google.com/github/jermwatt/machine_learning_refined/blob/main/notes/9_Feature_engineer_select/9_3_Scaling.ipynb

- First and foremost, they is indicative of a more general truth regarding machine learning cost functions: ***normalizing the input features of a dataset by mean centering and scaling by the standard deviation of each input will results in a cost function with less elliptical and more 'circular' contours.***
- This includes regression, (two-class / multi-class) classification, as well as unsupervised learning cost functions.

PCA-sphering scheme:

$$\mu_n = \frac{1}{P}\sum_{p=1}^{P} x_{p,n}$$

1. **(mean center)** for each n replace $x_{p,n} \longleftarrow (x_{p,n} - \mu_n)$ where

2. **(PCA rotation)** transform $\mathbf{w}_p = \mathbf{V}^T\mathbf{x}_p$ where $\mathbf{V}$ is the full set of eigenvectors of the regularized covariance matrix

3. **(divide off std)** for each n replace $w_{p,n} \longleftarrow \frac{w_{p,n}}{d_n^{1/2}}$ where $d_n^{1/2}$ is the square root of the n^{th} eigenvalue of the regularized covariance matrix

$$\mathbf{X} \longleftarrow \mathbf{D}^{-\frac{1}{2}}\mathbf{V}^T\mathbf{X}.$$

Boosting and Regularization

- Boosting:
  - Add feature one at a time (can generalize to a subset at a time)
- Regularization
  - Add a sparsity-inducing regularizer to cost functions

Chapter 10 Nonlinear Feature Engineering

- Nonlinear Learning: modeling principles
- Nonlinear multi-output regression
- Nonlinear two-class classification
- Nonlinear multi-class classification
- Nonlinear unsupervised learning

Nonlinear Feature Engineering

- In general we could create a nonlinear model that is the weighted sum of B nonlinear functions of our input as

$$\text{model}\left(\mathbf{x}, \Theta\right) = w_0 + f_1\left(\mathbf{x}\right) w_1 + f_2\left(\mathbf{x}\right) w_2 + \cdots + f_B\left(\mathbf{x}\right) w_B$$

- Here $f_1, f_2, \ldots f_B$ are nonlinear parameterized or unparameterized functions - or *feature transformations*.
- Once again w_0 through w_B (along with any additional weights internal to the nonlinear functions) are represented in the weight set Θ and must be tuned properly.

Chapter 11 Feature Learning

- What is Feature Learning
- Universal approximator
- Naïve Cross-Validation
- Cross-Validation via Boosting
- Cross-Validation via Regularization
- Testing data
- Which Universal Approximator works the Best
- Bagging
- When Feature Learning Fails

Chapter 12 Kernel Methods

- Fixed-Shape Universal Approximators
- The Kernel Trick
- Kernels as Similarity Measures
- Optimization of Kernelized Models
- Cross-Validating Kernelized Learners