CSE 5523:
Machine Learning Setup

The Ohio State University

Questions?

Today

Machine learning setup

- Review
- Course overview
- More formal definitions

What is machine learning?

Learning from Data

Algorithm, model **Evaluation, loss** **Training data**

“Supervised” ML pipeline

Training data → Learning algorithms → **Learned models & patterns**

“Supervised” ML pipeline

Training data

Learning algorithms

Learned models & patterns

Test data

How to know if the model works well or not?

Training data

Learning algorithms

Learned models & patterns

Test data

oracle

Difference (loss)

25

Ground truth

Different flavors of ML problems

- Several paradigms (e.g., dependent on the types of *"training"* data):
  - **Supervised learning (w/ labels)**
    - Training data: $\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$
    - $\boldsymbol{x}_n$ (or just $\boldsymbol{x}$) is the input instance/example/feature vector $\in \mathbb{R}^d$ (or $\mathcal{X}$: feature space)
    - y_n (or just y) is the label, e.g., a real number $\in \mathbb{R}$ or a category index $\in \mathcal{C}$ (or $\mathcal{Y}$: label space)
    - Aim to predict (as in many of the previous list of applications)
  - **Unsupervised learning (w/o labels)**
    - Training data: $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$
    - Aim to discover hidden and latent patterns and explore data
  - Reinforcement learning
  - Many others ……

Unsupervised learning

- Clustering:

- Distribution/Density estimation
  - Can sample from the distribution to generate new data

Classification vs. regression

- Classification
  - Form: Feature vector → Desired category (integer number index)
  - Ex: 1: Cat, 2: Dog, 3: Horse, ……, 1000: Car

- Regression (curve Fitting)
  - Form: Feature vector → Desired real number (e.g., price, chance, etc.)

Machine learning: capture patterns from training data that can be generalized to future unseen data

Questions to ML

- What assumptions do we need for learning to be possible?
  - Data and Models

- Given a fixed model type (e.g., linear boundaries), how can the machine output the “right” model?

- How many “training” data samples are needed to ensure that the output model will generalize well to the unseen test data?
- What is the practice of solving an ML problem?

Today

Machine learning setup

- Review
- Course overview
- More formal definitions

Topics

- **Supervised learning**
  - Regression
  - Classification (nearest neighbors, linear models, SVM)
  - Probabilistic approaches
  - Ensemble approaches
  - Kernel methods
  - Optimization
- **Machine learning foundation, theories, practice**
  - Empirical risk minimization and regularization
  - PAC learning and VC dimension
  - Bias and variance, over-fitting vs under-fitting
  - Debugging

Topics

- **Unsupervised learning**
  - K-means clustering
  - GMM and EM
  - Dimensionality reduction and metric learning (supervised or unsupervised)
- **Neural networks and deep learning**
  - Backpropagation and optimization
  - MLP
  - CNN
  - Transformer
- **Optional: Advanced ML paradigms or approaches**
  - Generative models: VAE, GAN, diffusion models
  - Domain adaptation, semi-supervised learning, self-supervised learning

Today

Machine learning setup

- Review
- Course overview
- More formal definitions

More formal definition of supervised learning

- An "unknown" true distribution $P(X \in \mathcal{X}, Y \in \mathcal{Y})$

- A loss function: $\ell(y' \in \mathcal{Y}, y \in \mathcal{Y}) \in \mathbb{R}$

- Goal: find a function $h: \mathcal{X} \mapsto \mathcal{Y}$ to minimize
  - $E_{(x,y)\sim P}[\ell(h(x), y)] = E_{(x,y)\sim P}[\ell(\hat{y}, y)]$, where $\hat{y} = h(x)$ is the predicted label

Data pair with true/target label: $(\ x\ ,\ y\ )$

h

$\hat{y} = h(x)$

More formal definition of supervised learning

- An "unknown" true distribution $P(X \in \mathcal{X}, Y \in \mathcal{Y})$

- A loss function: $\ell(y' \in \mathcal{Y}, y \in \mathcal{Y}) \in \mathbb{R}$

- Goal: find a function $h: \mathcal{X} \mapsto \mathcal{Y}$ to minimize
  - $E_{(x,y)\sim P}[\ell(h(x), y)] = E_{(x,y)\sim P}[\ell(\hat{y}, y)]$, where $\hat{y} = h(x)$ is the predicted label

- Training data:
  - $D_{tr} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$, where $(\boldsymbol{x}_n \in \mathcal{X}, y_n \in \mathcal{Y}) \sim P$ and $n \in \{1, \dots, N\}$
  - Each pair $(\boldsymbol{x}_n, y_n)$ is IID sampled from P; $D_{tr} \sim P^N$
  - ML: Apply the learning algorithm to find $h: \mathcal{X} \mapsto \mathcal{Y}$

More formal definition of supervised learning

- An "unknown" true distribution $P(X \in \mathcal{X}, Y \in \mathcal{Y})$

- A loss function: $\ell(y' \in \mathcal{Y}, y \in \mathcal{Y}) \in \mathbb{R}$

- Goal: find a function $h: \mathcal{X} \mapsto \mathcal{Y}$ to minimize
  - $E_{(x,y)\sim P}[\ell(h(x), y)] = E_{(x,y)\sim P}[\ell(\hat{y}, y)]$, where $\hat{y} = h(x)$ is the predicted label

- Test data:
  - $D_{te} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_M, y_M)\}$, where $(\boldsymbol{x}_m \in \mathcal{X}, y_m \in \mathcal{Y}) \sim P$
  - Each pair $(\boldsymbol{x}_m, y_m)$ is IID sampled from P; $D_{te} \sim P^M$
  - Compute $\sum_{m=1}^{M} \ell(h(\boldsymbol{x}_m), y_m)$

Special cases

- Data distributions:
  - $P(X, Y) = P(X|Y)P(Y) = P(X)P(Y|X)$
  - In some cases, we may assume that there exists an oracle (or labeling or target) function $f: \mathcal{X} \mapsto \mathcal{Y}$ that deterministically generates the true label.
  - $(x, y) \sim P(X)P(Y|X)$ becomes $x \sim P(X)$ and then $y = f(x)$

- Notations:
  - Sometimes people use $\mathcal{D}$ instead of P to denote a distribution

Hypothesis class (model type)

- Instead of allowing arbitrary h, we put some restrictions such that $h \in \mathcal{H}$
- $\mathcal{H}$: hypothesis class (a set of hypotheses h)
  - Linear classifiers/boundaries
  - Nonlinear classifiers/boundaries
  - Neural networks

- A hypothesis class encodes important *assumptions/prior knowledge/inductive bias* about the *problem (the distribution P)* we are trying to learn
- The No free lunch theorem states that every successful ML algorithm must make assumptions
  - There is no single ML algorithm that works for every problem (data distributions)

Hypothesis class (model type)

- Linear classifiers $\mathcal{H}$

Loss functions ℓ

- For classification
  - 0-1 loss: $\ell(y, y') = 1[\mathrm{y} \neq y']$; 1[True] = 1, 1[False] = 0 is called an indicator function

- For regression
  - Square loss: $\ell(y, y') = (y - y')^2$
  - Absolute loss: $\ell(y, y') = |y - y'|$
    - Grow linearly
    - More tolerance to noisy data

- There are many other types of loss functions, and you will see them soon

More formal definition of supervised learning

- Generalization loss/error/risk: $\mathcal{L}(h; P) = \epsilon(h; P) = E_{(x,y)\sim P}[\ell(h(x), y)]$
  - Aka, true error
  - Realizability (not always): there exists a $h \in \mathcal{H}$ such that $\epsilon(h; P) = 0$ for the 0-1 loss

- Test loss: $\mathcal{L}(h; D_{te}) = \hat{\epsilon}(h; D_{te}) = \frac{1}{M}\sum_{m=1}^{M} \ell(h(\boldsymbol{x}_m), y_m)$

- Training loss: $\mathcal{L}(h; D_{tr}) = \hat{\epsilon}(h; D_{tr}) = \frac{1}{N}\sum_{n=1}^{N} \ell(h(\boldsymbol{x}_n), y_n)$
  - Aka, empirical loss, empirical risk

- Since D_{tr} is random, $\hat{h}$ (the one found by the ML algorithm) is random
  - So are $\hat{\epsilon}(\hat{h}; D_{tr})$, $\hat{\epsilon}(\hat{h}; D_{te})$, and $\epsilon(\hat{h}; P)$

More in learning theories!

ML problem solving given D_{tr} (inner loop)

- Step 1: Select an ML algorithm
  - We think appropriate for the learning problem
  - Include selecting $\mathcal{H}$
  - More to be defined

- Step 2: Find the "best" $\hat{h} \in \mathcal{H}$ from D_{tr}
  - The "best" depends on the ML algorithm
  - Ideally, "best" for minimizing $\epsilon(\hat{h}; P)$
  - Usually involve an optimization problem

ML problem solving given D_{tr} (outer loop)

- How to pick an ML learning algorithm if there are several choices?
  - Goal: minimize the generalization error or test error
  - Problem: only have the training data
  - Solution: separate D_{tr} into D_{base} and D_{val}; treat D_{val} as "pseudo" test data
    - Separate "randomly" to keep D_{base}, D_{val} distributionally similar, not simply by first-vs-second half!

- For each learning algorithm (and its $\mathcal{H}$)
  - Find the "best" $\hat{h} \in \mathcal{H}$ from D_{base}
  - Compute $\hat{\epsilon}(\hat{h}; D_{val})$

- Choose the best algorithm whose $\hat{\epsilon}(\hat{h}; D_{val})$ is the smallest
  - Find the "best" $\hat{h} \in \mathcal{H}$ from D_{tr}, using the best algorithm

Empirical risk minimization (ERM)

- A widely used ML algorithm (together with different $\mathcal{H}$)

- Given D_{tr} (or D_{base}), output $\hat{h} \in \mathcal{H}$
  - $\hat{h} = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\, \hat{\epsilon}(h; D_{tr}) = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} \ell(h(\boldsymbol{x}_n), y_n)$
  - "Hope" that it will work well on $\epsilon(\hat{h}; P)$

- **When will the "hope" have some theoretical guarantee?**
  - $D_{tr} \sim P^N$ (or from a similar distribution: e.g., train on synthetic data, test on real data)
  - $\mathcal{H}$ is with appropriate assumptions and an appropriate set size
  - N is large enough

Generalization

- Can we simply choose the ML algorithm whose $\hat{\epsilon}(\hat{h}; D_{tr})$ is minimized?
  - Example: $\hat{h}(\boldsymbol{x}) = \begin{cases} y_n, & \text{if } \exists(\boldsymbol{x}_n, y_n) \in D_{tr} \quad \text{s.t.}\, \boldsymbol{x} = \boldsymbol{x}_n \\ 0, & \text{otherwise} \end{cases}$
  - $\hat{\epsilon}(\hat{h}; D_{tr}) = 0$, but know nothing about data beyond D_{tr}

- How to resolve this?
  - We can detect this problem by finding $\hat{h}(\boldsymbol{x})$ from D_{base} and evaluating $\hat{\epsilon}(\hat{h}; D_{val})$
  - This is why we separate D_{tr} into D_{base} and D_{val} to choose an ML algorithm

Why does data size matter?

Why does data size matter?

D_{tr} not representative of P to find a good $\hat{h}$

Why does data size matter?

D_{tr} not representative of P to find a good $\hat{h}$

D_{te} not representative of P to evaluate $\hat{h}$

Why do the similar data distributions matter?

D_{tr} not representative of P to find a good $\hat{h}$

Why does the hypothesis class matter?

Why does the hypothesis class matter?

- D_{tr} is "random"
- A "simpler" $\mathcal{H}$ is less sensitive to the randomness!

More

- Learning theories
- Concentration theory, Hoeffding's inequality, VC dimension, …

Summary

Learning from Data

Algorithm, model

Hypothesis class
+
Learning algorithm

Evaluation, loss

Training data

Why do we need linear algebra?

- Machine learning:
  - Learning patterns from data
  - A data instance can usually be represented by a vector
  - A dataset can usually be represented by a matrix
  - The "pattern" can usually be represented by a vector or a matrix!
    - A coin with weight * 0.2 + diameter * 0.04 < 2 $\Rightarrow$ 10 cent
  - To learn the right patterns, we need computation between possible patterns and data!

weight
diameter $\begin{bmatrix} 5.6 \\ 23.5 \end{bmatrix}$

$$2 - \begin{bmatrix} 0.2 \\ 0.04 \end{bmatrix}^T \times \begin{bmatrix} 5.6 \\ 23.5 \end{bmatrix}$$

- Linear algebra:
  - Powerful for computation on vectors and matrices
  - More important to get the meaning of the computation!
  - Please review the linear algebra slides!

Caution! Python indices start with 0!

TODO for math/programming review

- See the beginning slides and the course website for suggested reading

CSE 5523:
Nearest Neighbor

The Ohio State University

Important for this week

- **Register:** If you were on the waitlist but have not been enrolled, please talk to me after the class
- **Register** for the class on piazza: our platform for discussion and communication
  - https://piazza.com/osu/autumn2024/cse5523_chao (please use name.#@osu.edu)
  - Access code: osu-cse-5523-AU24-chao
- **Math review/self-diagnostic**: do Homework #0 and suggested materials on the website --- extremely important to check your readiness for the course
- **Decision:** stay or drop
- **Office hours:** start this Thursday
  - Mine: Tuesday 11 am - 12 pm, Thursday 6 pm - 7 pm (DL 587)
  - TA (Tai-Yu Pan): Monday 11 am - 12 pm, Friday 4 pm - 5 pm (BE406, #12 and #13)

Interest in the Buckeye AutoDrive Team?

Interested in the growing field of Autonomous Vehicles (AV)? Want to transform a Chevy Bolt into a fully autonomous vehicle?

Website

We have opportunities for CSE, MAE and ECE Students!

Gain hands-on technical skills while also networking with industry professionals at SAE & GM!

Interest Survey

Today

Review

- ML problem solving
- Training, validation, testing

Nearest neighbor

- Basic and variants
- Properties
- Curse of dimensionality
- Bayes optimal classifier

ML problem solving given D_{tr} (inner loop)

- Step 1: Select an ML algorithm
  - We think appropriate for the learning problem
  - Include selecting hypothesis class $\mathcal{H}$
  - More to be defined

- Step 2: Find the "best" $\hat{h} \in \mathcal{H}$ based on D_{tr}
  - The "best" depends on the ML algorithm
  - Ideally, "best" for minimizing $\epsilon(\hat{h}; P)$
  - Usually involve an optimization problem

ML problem solving given D_{tr} (outer loop)

- How to pick an ML learning algorithm if there are several choices?
  - Goal: minimize the generalization error or test error
  - Problem: only have the training data
  - Solution: separate D_{tr} into D_{base} and D_{val}; treat D_{val} as "pseudo" test data
    - Separate "randomly" to keep D_{base}, D_{val} distributionally similar, not simply by first-vs-second half!

- For each learning algorithm (and its $\mathcal{H}$)
  - Find the "best" $\hat{h} \in \mathcal{H}$ based on D_{base}
  - Compute $\hat{\epsilon}(\hat{h}; D_{val})$

- Choose the best algorithm whose $\hat{\epsilon}(\hat{h}; D_{val})$ is the smallest
  - Find the "best" $\hat{h} \in \mathcal{H}$ based on D_{tr}

Empirical risk minimization (ERM)

- A widely used ML algorithm (together with different $\mathcal{H}$)

- Given D_{tr} (or D_{base}), output $\hat{h} \in \mathcal{H}$
  - $\hat{h} = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\, \hat{\epsilon}(h; D_{tr}) = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} \ell(h(\boldsymbol{x}_n), y_n)$
  - "Hope" that it will work well on $\epsilon(\hat{h}; P)$

- **When will the "hope" have some theoretical guarantee?**
  - $D_{tr} \sim P^n$ (or from a similar distribution: e.g., train on synthetic data, test on real data)
  - $\mathcal{H}$ is with appropriate **"assumptions"** and an appropriate set size
  - N is large enough

Nearest neighbor

Which COLOR?
Why?

Nearest neighbor

Which COLOR?
Why?

Nearest neighbor

Which COLOR?
Why?

Nearest neighbor

Which COLOR?
Why?

Nearest neighbor

Which COLOR?
Why?

Nearest neighbor

- What is the assumption?
  - Similar Inputs (feature vectors) have similar outputs (labels)

- What does it mean by similar?
  - Smaller distance, larger similarity
  - Larger distance, smaller similarity

- Classification rule:
  - For a test data input $\boldsymbol{x}$, assign the label of its closet training data instance in D_{tr}

Nearest neighbor

Nearest neighbor

- **Given:**
  - A distance measure dist(. , .)
  - $D_{tr} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$, $t = \infty$ (min distance)
- **Input:** a test data input $\boldsymbol{x}$
- **NN classification rule (implementation):**

$t = \infty$

$n = 1 : N$

  if dist$(\boldsymbol{x}, \boldsymbol{x}_n) < t$

    $\hat{y} = y_n$

    $t = \text{dis}(\boldsymbol{x}, \boldsymbol{x}_n)$

Return: $\hat{y}$

Distance

- Nearest neighbor classifiers rely on a distance metric
- The better that metric reflects label similarity, the better the classifier is.

$\boldsymbol{x}$: 2-dimensional vectors

- L_2 distance (Euclidean): $$\|\boldsymbol{x} - \boldsymbol{x_n}\|_2 = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^2\right)^{1/2}$$
- L_1 distance: $$\|\boldsymbol{x} - \boldsymbol{x_n}\|_1 = \sum_{d=1}^{D} |x[d] - x_n[d]|$$
- L_p norm: $$\|\boldsymbol{x} - \boldsymbol{x_n}\|_p = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^p\right)^{1/p}$$

Green line is Euclidean distance. Red, Blue, and Yellow lines are L_1 distance

Nearest neighbor (Euclidean distance)

$$\|\boldsymbol{x} - \boldsymbol{x_n}\|_2 = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^2\right)^{1/2}$$

Nearest neighbor (Manhattan distance)

$$\|\boldsymbol{x} - \boldsymbol{x_n}\|_1 = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^1\right)^{1/1}$$

Nearest neighbor

$$\|\boldsymbol{x} - \boldsymbol{x_n}\|_\infty = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^\infty\right)^{1/\infty} \approx \max_d |x[d] - x_n[d]|$$

Nearest neighbor (Euclidean distance)

- Given $\boldsymbol{x}$
- if $\exists(\boldsymbol{x}_n, y_n) \in D_{tr}$ s.t. $\boldsymbol{x} = \boldsymbol{x}_n$
- $\hat{y} = y_n$

$$\|\boldsymbol{x} - \boldsymbol{x}_n\|_2 = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^2\right)^{1/2}$$

K-Nearest neighbors (*K*NN)

- What is the assumption?
  - Similar Inputs (feature vectors) have similar outputs (labels)

- Classification rule:
  - For a test data input $\boldsymbol{x}$, assign the most common label amongst its K most similar training data instances in D_{tr}
  - aka, majority vote

*K*NN (Euclidean distance)

$$\|\boldsymbol{x} - \boldsymbol{x_n}\|_2 = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^2\right)^{1/2}$$

*K*NN (Euclidean distance)

$$\|\boldsymbol{x} - \boldsymbol{x_n}\|_2 = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^2\right)^{1/2}$$

*K*NN (Euclidean distance)

$$\|\boldsymbol{x} - \boldsymbol{x_n}\|_2 = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^2\right)^{1/2}$$

*K*NN (Euclidean distance)

K=*N*

- Just the major class
- Independent of where $\boldsymbol{x}$ is!

$$\|\boldsymbol{x} - \boldsymbol{x_n}\|_2 = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^2\right)^{1/2}$$

*K*NN

- **Given:**
  - A distance measure dist(. , .)
  - $D_{tr} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$, $t = \infty$ (min distance)
- **Input:** a test data input $\boldsymbol{x}$
- ***K*NN classification rule (implementation):**
  - Denote by KNN($\boldsymbol{x}$) the set of the K nearest neighbors of $\boldsymbol{x}$ in D_{tr}
    - KNN($\boldsymbol{x}$) $\subset D_{tr}$, |KNN($\boldsymbol{x}$)| $= K$
    - $\forall(\boldsymbol{x}', y') \in D_{tr} \backslash$KNN($\boldsymbol{x}$), dist($\boldsymbol{x}, \boldsymbol{x}'$) $\geq \max_{(\boldsymbol{x}'', y'') \in \text{KNN}(\boldsymbol{x})}$ dist($\boldsymbol{x}, \boldsymbol{x}''$)
    - Every point in D_{tr} but not in KNN($\boldsymbol{x}$) is at least as far away from $\boldsymbol{x}$ as the furthest point in KNN($\boldsymbol{x}$)
  - **Return:** $\hat{y} = \hat{h}(\boldsymbol{x}) = \text{mode}(\{y''; \ (\boldsymbol{x}'', y'') \in \text{KNN}(\boldsymbol{x})\})$: most frequent label in KNN($\boldsymbol{x}$)

Question

- How to resolve the case of a tie?

- Applicable to binary classification only or not?

*K*NN (Euclidean distance) for multiple classes

*K*NN (Euclidean distance) for multiple classes

Variants of *K*NN: Rules

Weighted KNN

- for class c = 1: C
  - Vote(c) = $\sum_{(\boldsymbol{x}',y')\in KNN(\boldsymbol{x})} \mathbf{1}[y' == c] \times \text{weight}(\boldsymbol{x}, \boldsymbol{x}')$
- $\hat{y} = \underset{c}{\text{argmax}}\, \text{Vote}(c)$
- For example
  - $\text{weight}(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}}$
  - Radial basis function (RBF) kernel

Variants of *K*NN: Distances

- Cosine distance
  - $\cos(\boldsymbol{x}, \boldsymbol{x}') = \frac{\boldsymbol{x}^T \boldsymbol{x}'}{\|\boldsymbol{x}\|_2 \|\boldsymbol{x}'\|_2}$
  - $1 - \cos(\boldsymbol{x}, \boldsymbol{x}') = 1 - \frac{\boldsymbol{x}^T \boldsymbol{x}'}{\|\boldsymbol{x}\|_2 \|\boldsymbol{x}'\|_2}$

- Mahalanobis distance (less strict form)
  - $(\boldsymbol{x} - \boldsymbol{x}')^T A (\boldsymbol{x} - \boldsymbol{x}')$, where $A \in \mathbb{R}^{D \times D}$ is symmetric positive semi-definite (PSD)
  - PSD: $\forall \boldsymbol{x} \neq \boldsymbol{0}, \boldsymbol{x}^T A \boldsymbol{x} \geq \boldsymbol{0}$
  - $(\boldsymbol{x} - \boldsymbol{x}')^T A (\boldsymbol{x} - \boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{x}')^T M^T M (\boldsymbol{x} - \boldsymbol{x}') = \left(M(\boldsymbol{x} - \boldsymbol{x}')\right)^T M (\boldsymbol{x} - \boldsymbol{x}')$
  - $M \in \mathbb{R}^{D' \times D}$ and has no other constraint

Variants of *K*NN: Distances

- What does Mahalanobis distance mean? How to set *M*?

$$M = \begin{bmatrix} \frac{1}{a} & 0 \\ 0 & \frac{1}{b} \end{bmatrix}$$

$$\begin{bmatrix} a \\ 0 \end{bmatrix}^T \begin{bmatrix} \frac{1}{a} & 0 \\ 0 & \frac{1}{b} \end{bmatrix}^T \begin{bmatrix} \frac{1}{a} & 0 \\ 0 & \frac{1}{b} \end{bmatrix} \begin{bmatrix} a \\ 0 \end{bmatrix} = 1,$$

$$\begin{bmatrix} 0 \\ b \end{bmatrix}^T \begin{bmatrix} \frac{1}{a} & 0 \\ 0 & \frac{1}{b} \end{bmatrix}^T \begin{bmatrix} \frac{1}{a} & 0 \\ 0 & \frac{1}{b} \end{bmatrix} \begin{bmatrix} 0 \\ b \end{bmatrix} = 1$$

- Long-standing statistical problem. Can estimate *M* or *A* from covariance
- Distance metric learning to learn *M* or *A*

Commonly-used tricks for pre-processing

- Z-score
  - Compute training mean: $\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n$
  - Compute training STD per dimension: $\rho[d] = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(\boldsymbol{x}_n[d] - \boldsymbol{\mu})^2}$
  - For every data instances $\boldsymbol{x}$, training or test, do $\boldsymbol{x}[d] \leftarrow \frac{\boldsymbol{x}[d]-\boldsymbol{\mu}[d]}{\rho[d]}$
  - Make the training data "zero"-mean, and equal variance per dimension

- In practice, many other pre-processing methods like L2-normalization

Properties

- Does learning EVER happen?
  - Seems NO before seeing the test data instance $\boldsymbol{x}$
  - Lazy learning: generalization of the training data is delayed until a test data comes.
  - Opposite: eager learning, which generalizes the training data before test data.

- Is *K*NN merely memorization?
  - Recall: $\hat{h}(\boldsymbol{x}) = \begin{cases} y_n, & \text{if } \exists(\boldsymbol{x}_n, y_n) \in D_{tr} \quad \text{s.t.}\, \boldsymbol{x} = \boldsymbol{x}_n \\ 0, & \text{otherwise} \end{cases}$
  - NN: Recall: $\hat{h}(\boldsymbol{x}) = \begin{cases} y_n, & \text{if } \exists(\boldsymbol{x}_n, y_n) \in D_{tr} \quad \text{s.t.}\, \boldsymbol{x} = \boldsymbol{x}_n \\ y_m & \text{s.t.}\, m = \operatorname{argmin}_n \operatorname{dist}(\boldsymbol{x}, \boldsymbol{x}_n) \end{cases}$
  - *K*NN: Recall: $\hat{h}(\boldsymbol{x}) = \operatorname{mode}(\{y''; \ (\boldsymbol{x}'', y'') \in \text{KNN}(\boldsymbol{x})\})$

Assuming no duplicated data input!

Properties

- Why does *K*NN not just memorize?
  - Similar Inputs (feature vectors) have similar outputs (labels)
  - Every successful ML algorithms MUST make assumptions
- What is the hypothesis class $\mathcal{H}$?
  - Linear?
- What are the controlling factors of $\mathcal{H}$?
  - *K*
  - Distance metric
  - They are called "hyper-parameters"

Properties

- What is the training error of NN?
  - $\hat{\epsilon}(\hat{h}; D_{tr}) = \frac{1}{N}\sum_{n=1}^{N} \ell\left(y_n, \hat{h}(\boldsymbol{x}_n; D_{tr})\right)$
  - Assume there are no pair of training data share the same input
    - $\hat{\epsilon}(\hat{h}; D_{tr}) = \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, y_n) = 0$

- What is the training error of *K*NN?
  - May NOT be 0

What is the effect of K?

What is the effect of *K*?

How to choose K?

Leave-one-out Cross-validation

Each training instance takes turns to temporally be the "single" Val instance

- $\hat{\epsilon}(\hat{h}; D_{val}) = 0$
- for n = 1 : N
  - temporally remove $(\boldsymbol{x}_n, y_n)$ from D_{tr}
  - $\hat{\epsilon}(\hat{h}; D_{val}) \mathrel{+}= \ell(y_n, \hat{h}(\boldsymbol{x}_n; D_{tr} \backslash \{(\boldsymbol{x}_n, y_n)\}))$
- $\hat{\epsilon}(\hat{h}; D_{val}) = \frac{\hat{\epsilon}(\hat{h}; D_{val})}{N}$

Pros and Cons

- Pros
  - Easy to implement (and have theoretical guarantee)
  - No training, suitable for “dynamically” changing training data
  - Suitable for “unknown” number of classes
  - Suitable for not well-defined classes (e.g., image retrieval, Google image search)

Okapi un animal misterios Okapia johnstoni - zooland.ro

- Cons
  - Need to carry all the training data to testing
  - O(NxD) test time (speed-up by KD-tree or hashing)
  - Become less effective in high-dimensional space: Curse of dimensionality
  - Popular distance metrics may not be good enough

Curse of dimensionality

- In high-dimensional space
  - Hard to find training data that are close to the test data

Higher dimension, hard to find neighbors!

Most uniformly distributed data are outside the circle

$$V(D) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)} r^D \quad \text{v.s.} \quad (2r)^D$$

Curse of dimensionality

- In high-dimensional space
  - Every training data seem to be equally far away from the test data

$$V(A) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)}(r+\varepsilon)^D$$

$$V(B) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)}r^D$$

$\frac{V(A)-V(B)}{V(B)} \uparrow$, as $D \uparrow$

Feature learning, dimensionality reduction

$$\boldsymbol{x} = g(I)$$
$$g: \mathcal{I} \mapsto \mathcal{X}$$

Theoretical guarantee: Loss lower bound

- Assume our true data distribution is $P(X \in \mathcal{X}, Y \in \mathcal{Y}) = P(X)P(Y|X)$
- Assume you know the distribution (impossible mostly, thus an ideal case):
  - What will you predict given $\boldsymbol{x}$? Consider the 0-1 loss!
  - Predict $y^* = \underset{y}{argmax}\, P(Y = y|\boldsymbol{x})$, what is the "expected" loss?
    - $(1 - P(Y = y^*|\boldsymbol{x}))$
    - Predicting any other label will lead to a higher expected loss
- **Bayes optimal classifier:** $y^* = \underset{y}{argmax}\, P(Y = y|\boldsymbol{x})$
  - "Generalization" loss: $E_{\boldsymbol{x}\sim P(\boldsymbol{x})}[1 - P(Y = y^*|\boldsymbol{x})]$
  - Cannot go even lower without other extra information

Theoretical guarantee: NN vs. Bayes optimal

- Claim: As $N \to \infty$, the NN classification error is no more than TWICE of Bayes optimal
  - When $N \to \infty$, the nearest neighbor $\boldsymbol{x}'$ of $\boldsymbol{x}$ has dist($\boldsymbol{x}$, $\boldsymbol{x}'$) →0, which means $\boldsymbol{x}'$ is almost identical to $\boldsymbol{x}$
  - Assign the label of $\boldsymbol{x}'$ to $\boldsymbol{x}$
  - When will it go wrong? (let's try binary)
    - $\boldsymbol{x}'$: $+1$, $\boldsymbol{x}$: -1
    - $\boldsymbol{x}'$: -1, $\boldsymbol{x}$: $+1$
    - $2 \times P(Y = y^* | \boldsymbol{x})\big(1 - P(Y = y^* | \boldsymbol{x})\big)$
    - $\leq 2 \times \big(1 - P(Y = y^* | \boldsymbol{x})\big)$

  - Loss(KNN) $\leq 2 \times \big(1 - P(Y = y^* | \boldsymbol{x})\big) = 2 \times$ Loss(Bayes optimal)

Summary

- *K*NN classifiers are lazy learning methods
- The same algorithm works for binary and multi-class classification
- Assumptions: similar inputs, similar outputs

- Different hypothesis classes are governed by (1) K, (2) distance metric
- “Leave-One-Out” (LOO) cross-validation

- Pros: Easy to implement, “no training”, theoretical guarantee
- Cons: test time, (training) data memory, curse of dimensionality

CSE 5523:
Nearest Neighbor + Linear Regression

The Ohio State University

Important for this week

- **Register** for the class on Piazza: our platform for discussion and communication
  - https://piazza.com/osu/autumn2024/cse5523_chao (please use name.#@osu.edu)
  - Access code: osu-cse-5523-AU24-chao

- **Math review/self-diagnostic**: do Homework #0 and suggested materials on the website --- extremely important to check your readiness for the course

- **Office hours:** start this Thursday
  - Mine: Tuesday 11 am - 12 pm, Thursday 6 pm - 7 pm (DL 587)
  - TA (Tai-Yu Pan): Monday 11 am - 12 pm, Friday 4 pm - 5 pm (BE406, #12 and #13)

Homework

- HW # 1:
  - Release date: 9/3
  - Due date: 9/17 (11:59 pm)
  - Problem set + Programming set

Today

Nearest neighbor

- Review
- Properties (continued)
- Curse of dimensionality
- Bayes optimal classifier

Linear regression

K-Nearest neighbors (*K*NN)

- What is the assumption?
  - Similar Inputs (feature vectors) have similar outputs (labels)
  - This makes *K*NN not merely memorization!

- Classification rule:
  - For a test data input $\boldsymbol{x}$, assign the most common label amongst its K most similar training data instances in D_{tr}
  - aka, majority vote

*K*NN (Euclidean distance)

$$\|\boldsymbol{x} - \boldsymbol{x_n}\|_2 = \left(\sum_{d=1}^{D} |x[d] - x_n[d]|^2\right)^{1/2}$$

Distance

- The better the distance metric reflects label similarity, the better the classifier is.
  - L_2 distance (Euclidean): $$\|\boldsymbol{x}-\boldsymbol{x_n}\|_2 = \left(\sum_{d=1}^{D} |x[d]-x_n[d]|^2\right)^{1/2}$$
  - L_1 distance: $$\|\boldsymbol{x}-\boldsymbol{x_n}\|_1 = \sum_{d=1}^{D} |x[d]-x_n[d]|$$
  - L_p norm: $$\|\boldsymbol{x}-\boldsymbol{x_n}\|_p = \left(\sum_{d=1}^{D} |x[d]-x_n[d]|^p\right)^{1/p}$$
  - Cosine distance, Mahalanobis distance, ……
  - Data pre-processing (e.g., z-score)

$\boldsymbol{x}$: 2-dimensional vectors

Green line is Euclidean distance. Red, Blue, and Yellow lines are L_1 distance

Commonly-used tricks for pre-processing

- Z-score
  - Compute training mean: $\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n$
  - Compute training STD per dimension: $\rho[d] = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(\boldsymbol{x}_n[d] - \boldsymbol{\mu})^2}$
  - For every data instances $\boldsymbol{x}$, training or test, do $\boldsymbol{x}[d] \leftarrow \frac{\boldsymbol{x}[d]-\boldsymbol{\mu}[d]}{\rho[d]}$
  - Make the training data "zero"-mean, and equal variance per dimension

- In practice, many other pre-processing methods like L2-normalization

Properties

- Lazy learning methods
- Work for binary and multi-class classification
- Different hypothesis classes:
  - K + distance metric
  - They are called "hyper-parameters"

Properties

- What is the training error of NN?
  - $\hat{\epsilon}(\hat{h}; D_{tr}) = \frac{1}{N}\sum_{n=1}^{N} \ell\left(y_n, \hat{h}(\boldsymbol{x}_n; D_{tr})\right)$
  - Assume there are no pair of training data share the same input
    - $\hat{\epsilon}(\hat{h}; D_{tr}) = \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, y_n) = 0$

- What is the training error of *K*NN?
  - May NOT be 0

What is the effect of K?

What is the effect of *K*?

How to choose K?

Leave-one-out Cross-validation

Each training instance takes turns to temporally be the "single" Val instance

- $\hat{\epsilon}(\hat{h}; D_{val}) = 0$
- for n = 1 : N
  - temporally remove $(\boldsymbol{x}_n, y_n)$ from D_{tr}
  - $\hat{\epsilon}(\hat{h}; D_{val}) \mathrel{+}= \ell(y_n, \hat{h}(\boldsymbol{x}_n; D_{tr} \backslash \{(\boldsymbol{x}_n, y_n)\}))$
- $\hat{\epsilon}(\hat{h}; D_{val}) = \frac{\hat{\epsilon}(\hat{h}; D_{val})}{N}$

Pros and Cons

- Pros
  - Easy to implement (and have theoretical guarantee)
  - No training, suitable for “dynamically” changing training data
  - Suitable for “unknown” number of classes
  - Suitable for not well-defined classes (e.g., image retrieval, Google image search)

Okapi un animal misterios Okapia johnstoni - zooland.ro

- Cons
  - Need to carry all the training data to testing
  - O(NxD) test time (speed-up by KD-tree or hashing)
  - Become less effective in high-dimensional space: Curse of dimensionality
  - Popular distance metrics may not be good enough

Curse of dimensionality

- In high-dimensional space
  - Hard to find training data that are close to the test data

Higher dimension, hard to find neighbors!

Most uniformly distributed data are outside the circle

$$V(D) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)} r^D \quad \text{v.s.} \quad (2r)^D$$

Curse of dimensionality

- In high-dimensional space
  - Every training data seem to be equally far away from the test data

$$V(A) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)}(r+\varepsilon)^D$$

$$V(B) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)}r^D$$

$\frac{V(A)-V(B)}{V(B)} \uparrow$, as $D \uparrow$

Feature learning, dimensionality reduction

$$\boldsymbol{x} = g(I)$$
$$g: \mathcal{I} \mapsto \mathcal{X}$$

Theoretical guarantee: Loss lower bound

- Assume our true data distribution is $P(X \in \mathcal{X}, Y \in \mathcal{Y}) = P(X)P(Y|X)$
- Assume you know the distribution (impossible mostly, thus an ideal case):
  - What will you predict given $\boldsymbol{x}$? Consider the 0-1 loss!
  - Predict $y^* = \underset{y}{argmax}\, P(Y = y|\boldsymbol{x})$, what is the "expected" loss?
    - $(1 - P(Y = y^*|\boldsymbol{x}))$
    - Predicting any other label will lead to a higher expected loss
- **Bayes optimal classifier:** $y^* = \underset{y}{argmax}\, P(Y = y|\boldsymbol{x})$
  - "Generalization" loss: $E_{\boldsymbol{x}\sim P(\boldsymbol{x})}[1 - P(Y = y^*|\boldsymbol{x})]$
  - Cannot go even lower without other extra information

Theoretical guarantee: NN vs. Bayes optimal

- Claim: As $N \to \infty$, the NN classification error is no more than TWICE of Bayes optimal
  - When $N \to \infty$, the nearest neighbor $\boldsymbol{x}'$ of $\boldsymbol{x}$ has dist($\boldsymbol{x}$, $\boldsymbol{x}'$) →0, which means $\boldsymbol{x}'$ is almost identical to $\boldsymbol{x}$
  - Assign the label of $\boldsymbol{x}'$ to $\boldsymbol{x}$
  - When will it go wrong? (let's try binary)
    - $\boldsymbol{x}'$: $+1$, $\boldsymbol{x}$: -1
    - $\boldsymbol{x}'$: -1, $\boldsymbol{x}$: $+1$
    - $2 \times P(Y = y^* | \boldsymbol{x})\big(1 - P(Y = y^* | \boldsymbol{x})\big)$
    - $\leq 2 \times \big(1 - P(Y = y^* | \boldsymbol{x})\big)$

  - Loss(KNN) $\leq 2 \times \big(1 - P(Y = y^* | \boldsymbol{x})\big) = 2 \times$ Loss(Bayes optimal)

Summary

- *K*NN classifiers are lazy learning methods
- The same algorithm works for binary and multi-class classification
- Assumptions: similar inputs, similar outputs

- Different hypothesis classes are governed by (1) K, (2) distance metric
- “Leave-One-Out” (LOO) cross-validation

- Pros: Easy to implement, “no training”, theoretical guarantee
- Cons: test time, (training) data memory, curse of dimensionality

(Linear) regression

- Predicting a continuous output
  - Price, distance, location, ...

- Setup
  - $D_{tr} = \{(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \mathbb{R})\}_{n=1}^{N}$

- Key different from classification
  - The loss function is different!
  - Leading to different models and algorithms

Linear regression

- Data visualization and task illustration

- Meaning:
  - Sale price = price-per-sqft * sqft + fixed_expense

- How to measure the loss?
  - Square loss: $\ell(y, \hat{y}) = (y - \hat{y})^2$
  - Absolute loss: $\ell(y, \hat{y}) = |y - \hat{y}|$

Regression (house price):
From x (area), predict y (price)

Linear regression: beyond 1-D input

Linear regression

- Training data: $D_{tr} = \{(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \mathbb{R})\}_{n=1}^{N}$

- Models (hypothesis class):
  - $\hat{y} = h(\boldsymbol{x}) = w[1]x[1] + \cdots + w[D]x[D] + b = \sum_{d=1}^{D} w[d]x[d] + b = \boldsymbol{w}^T\boldsymbol{x} + b$

  - $\boldsymbol{w} \in \mathbb{R}^D$: weights, parameters, parameter weights
  - b (or usually denoted by $w[0]$): bias, intercept

  - Concatenation: $\widetilde{\boldsymbol{w}} = [w[0], w[1], \ldots, w[D]]^T \in \mathbb{R}^{D+1}$; $\widetilde{\boldsymbol{x}} = [1, x[1], \ldots, x[D]]^T \in \mathbb{R}^{D+1}$
  - $\boldsymbol{w}^T\boldsymbol{x} + w[0] = \widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}}$

  - Pay attention to the context: People may use notations differently!

Linear regression

- Models (hypothesis class):
  - $\hat{y} = h(\boldsymbol{x}) = w[1]x[1] + \cdots + w[D]x[D] + b = \sum_{d=1}^{D} w[d]x[d] + b = \boldsymbol{w}^T\boldsymbol{x} + b$

- Assumptions:
  - Linear relationship between input features and output label
  - Linear combinations of input features
  - Parameters are coefficients of each input feature dimension $x[d]$
  - Thus, only works when we expect the input-output relationship is a line/plane/hyperplane

Learning goal

- Ideally, minimizing the generalization error as much as possible
  - But we only have training data
  - So, let's try empirical risk minimization

- **Sum of Squared Error (SSE) or Residual sum of square (RSS)**
  - RSS($\widetilde{\boldsymbol{w}}$) = $\sum_i [y_i - h(\boldsymbol{x}_i)]^2 = \sum_i [y_i - (\boldsymbol{w}^T \boldsymbol{x}_i + b)]^2 = \sum_i [y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i]^2$
    - Simplicity
    - Gaussian noise assumption (later)

- Others:
  - **Sum of Absolute Error (SAE):** $\sum_i |y_i - h(\boldsymbol{x}_i)|$
  - **Maximum error:** $\max_i |y_i - h(\boldsymbol{x}_i)|$

Least mean square (LMS) solution

- $\widetilde{\boldsymbol{w}}^{\mathrm{LMS}} = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}}\, \mathrm{RSS}(\widetilde{\boldsymbol{w}}) = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_i [y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i]^2$

- Let's try to expand it!

$$
\begin{aligned}
&\mathrm{RSS}(\widetilde{\boldsymbol{w}}) \\
&= \sum_i \left(y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)^2 \\
&= \sum_i \left(y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)\left(y_i - \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}}\right) \\
&= \sum_i y_i{}^2 + \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} - 2 y_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} = \sum_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} - 2 y_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} + \text{constant}
\end{aligned}
$$

Least mean square (LMS) solution

- Data notation

$$\boldsymbol{X} \in \mathbb{R}^{D\times N} = [\boldsymbol{x}_1, \quad \boldsymbol{x}_2, \quad \boldsymbol{x}_3, \quad \dots, \quad \boldsymbol{x}_N] = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \boldsymbol{x}_3 & \dots & \boldsymbol{x}_N \\ | & | & | & & | \end{bmatrix}$$

$$\widetilde{\boldsymbol{X}} \in \mathbb{R}^{(D+1)\times N} = [\widetilde{\boldsymbol{x}}_1, \quad \widetilde{\boldsymbol{x}}_2, \quad \widetilde{\boldsymbol{x}}_3, \quad \dots, \quad \widetilde{\boldsymbol{x}}_N] = \begin{bmatrix} | & | & | & & | \\ \widetilde{\boldsymbol{x}}_1 & \widetilde{\boldsymbol{x}}_2 & \widetilde{\boldsymbol{x}}_3 & \dots & \widetilde{\boldsymbol{x}}_N \\ | & | & | & & | \end{bmatrix}$$

$$\boldsymbol{y} \in \mathbb{R}^N = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Least mean square (LMS) solution

- Continued

$$= \sum_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} - 2y_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} + \text{constant}$$

$$= \widetilde{\boldsymbol{w}}^T \left(\sum_i \widetilde{\boldsymbol{x}}_i \widetilde{\boldsymbol{x}}_i{}^T \right) \widetilde{\boldsymbol{w}} - 2 \left(\sum_i y_i \widetilde{\boldsymbol{x}}_i{}^T \right) \widetilde{\boldsymbol{w}} + \text{constant}$$

$$= \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{X}} \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\left(\widetilde{\boldsymbol{X}} \boldsymbol{y}\right)^T \widetilde{\boldsymbol{w}} + \text{constant}$$

$$\boxed{\boldsymbol{X}\,\boldsymbol{X}^T = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \boldsymbol{x}_3 & \dots & \boldsymbol{x}_N \\ | & | & | & & | \end{bmatrix} \begin{bmatrix} -\boldsymbol{x}_1^T - \\ \vdots \\ -\boldsymbol{x}_N^T - \end{bmatrix} = \sum_{i=1}^N \begin{bmatrix} | \\ \boldsymbol{x}_i \\ | \end{bmatrix} \begin{bmatrix} -\boldsymbol{x}_i^T - \end{bmatrix}}$$

Least mean square (LMS) solution

- Normal equations
  - Taking derivative of RSS($\widetilde{\boldsymbol{w}}$) w.r.t. $\widetilde{\boldsymbol{w}}$, and setting it to zeros

$$\frac{\partial \mathrm{RSS}(\widetilde{\boldsymbol{w}})}{\partial \widetilde{\boldsymbol{w}}} = \frac{\partial\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\left(\widetilde{\boldsymbol{X}}\boldsymbol{y}\right)^T \widetilde{\boldsymbol{w}}\right)}{\partial \widetilde{\boldsymbol{w}}} = 2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\widetilde{\boldsymbol{X}}\boldsymbol{y} = \boldsymbol{0} \in \mathbb{R}^{(D+1)}$$

Closed-form $\widetilde{\boldsymbol{w}}^{\mathrm{LMS}} = \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-1}\widetilde{\boldsymbol{X}}\boldsymbol{y}$, since $2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} = 2\widetilde{\boldsymbol{X}}\boldsymbol{y}$ leads to $\widetilde{\boldsymbol{w}} = \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-1}\widetilde{\boldsymbol{X}}\boldsymbol{y}$

$$E(\boldsymbol{w}) = \boldsymbol{x}^T\boldsymbol{w}: \quad \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial E(\boldsymbol{w})}{\partial w[1]} \\ \vdots \\ \frac{\partial E(\boldsymbol{w})}{\partial w[D]} \end{bmatrix} = \boldsymbol{x}$$

$$E(\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{A}\boldsymbol{w}: \quad \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = \boldsymbol{A}\boldsymbol{w} + \boldsymbol{A}^T\boldsymbol{w} = \left(\boldsymbol{A} + \boldsymbol{A}^T\right)\boldsymbol{w}$$

More useful tools in the Matrix Cookbook!

Least mean square (LMS) solution: another way

$$
\begin{aligned}
&\mathrm{RSS}(\widetilde{\boldsymbol{w}}) \\
&= \sum_i \left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i - y_i\right)^2 \\
&= \sum_i \left(\widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} - y_i\right)^2 \\
&= \left[\left(\widetilde{\boldsymbol{x}}_1{}^T \widetilde{\boldsymbol{w}} - y_1\right), \dots\dots, \left(\widetilde{\boldsymbol{x}}_N{}^T \widetilde{\boldsymbol{w}} - y_N\right)\right] \begin{bmatrix} \widetilde{\boldsymbol{x}}_1{}^T \widetilde{\boldsymbol{w}} - y_1 \\ \vdots \\ \widetilde{\boldsymbol{x}}_N{}^T \widetilde{\boldsymbol{w}} - y_N \end{bmatrix} \\
&= \left\|\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - \mathbf{y}\right\|_2^2 \\
&= \left(\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - \mathbf{y}\right)^T \left(\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - \mathbf{y}\right) \\
&= \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{X}} \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\left(\widetilde{\boldsymbol{X}} \boldsymbol{y}\right)^T \widetilde{\boldsymbol{w}} + \text{constant}
\end{aligned}
$$

Mini summary

- Least mean square (LMS) has a closed-form solution
- Works for any model with **<u>linear</u>** <u>combination of parameters</u> to minimize RSS

- Bottleneck
  - $\left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-\mathbf{1}}$: computationally $O\left((D+1)^3\right)$
  - $\left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-\mathbf{1}}$: exists only if $\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T$ is of full rank: $N \geq D+1$. In other words, we need at least D+1 training data
  - Often, we need more than that to overcome noise and over-fitting

Recap: Eigen-decomposition (ED)

- Given a square matrix $\boldsymbol{A} \in R^{D \times D}$
- If $\boldsymbol{A}$ has D linearly independent eigenvectors $\{\boldsymbol{q}_1, \dots, \boldsymbol{q}_D\}$ and the corresponding eigenvalues $\{\lambda_1, \dots, \lambda_D\}$

$$\boldsymbol{A}\begin{bmatrix} | & & | \\ \boldsymbol{q}_1 & \dots & \boldsymbol{q}_D \\ | & & | \end{bmatrix} = \boldsymbol{A}\boldsymbol{Q} = \begin{bmatrix} | & & | \\ \lambda_1\boldsymbol{q}_1 & \dots & \lambda_D\boldsymbol{q}_D \\ | & & | \end{bmatrix} = \begin{bmatrix} | & & | \\ \boldsymbol{q}_1 & \dots & \boldsymbol{q}_D \\ | & & | \end{bmatrix}\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_D \end{bmatrix} = \boldsymbol{Q}\boldsymbol{\Lambda}$$

- Eigen-decomposition: $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$
  - $(\boldsymbol{A}\boldsymbol{Q})\boldsymbol{Q}^{-1} = (\boldsymbol{Q}\boldsymbol{\Lambda})\,\boldsymbol{Q}^{-1}$ then $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$
- Not all the square matrices have eigen-decomposition

Recap: Singular value decomposition (SVD)

- Singular value decomposition (SVD)

Non-negative

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Gamma}\boldsymbol{V}^T = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 & \boldsymbol{u}_3 & \dots & \boldsymbol{u}_D \\ | & | & | & & | \end{bmatrix} \begin{bmatrix} \gamma_1 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & \gamma_D & 0 \end{bmatrix} \begin{bmatrix} -\boldsymbol{v}_1^T- \\ \vdots \\ -\boldsymbol{v}_N^T- \end{bmatrix}$$

(D x N) (D x D) (D x N) (N x N)

$$\boldsymbol{U}^T\boldsymbol{U} = \begin{bmatrix} -\boldsymbol{u}_1^T- \\ \vdots \\ -\boldsymbol{u}_N^T- \end{bmatrix} \begin{bmatrix} | & | & | & & | \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 & \boldsymbol{u}_3 & \dots & \boldsymbol{u}_D \\ | & | & | & & | \end{bmatrix} = \boldsymbol{I} = \boldsymbol{U}\boldsymbol{U}^T$$

Orthonormal, unitary

$$\boldsymbol{V}^T\boldsymbol{V} = \begin{bmatrix} -\boldsymbol{v}_1^T- \\ \vdots \\ -\boldsymbol{v}_N^T- \end{bmatrix} \begin{bmatrix} | & | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 & \dots & \boldsymbol{v}_D \\ | & | & | & & | \end{bmatrix} = \boldsymbol{I} = \boldsymbol{V}\boldsymbol{V}^T$$

Orthonormal, unitary

Recap: SVD + ED

- Combination (suppose $\boldsymbol{X} \in R^{D\times N}$ and $\boldsymbol{X} = \boldsymbol{U\Gamma V}^T$)

$$\boldsymbol{XX}^T = \boldsymbol{U\Gamma V}^T\left(\boldsymbol{U\Gamma V}^T\right)^T = \boldsymbol{U\Gamma V}^T\left(\boldsymbol{V\Gamma}^T\boldsymbol{U}^T\right) = \boldsymbol{U\Gamma V}^T\boldsymbol{V\Gamma}^T\boldsymbol{U}^T = \boldsymbol{U\Gamma\Gamma}^T\boldsymbol{U}^T$$
$$= \boldsymbol{U}\left(\boldsymbol{\Gamma\Gamma}^T\right)\boldsymbol{U}^T = \boldsymbol{U}\left(\boldsymbol{\Gamma\Gamma}^T\right)\boldsymbol{U}^{-1}$$

  - $\boldsymbol{XX}^T$ has an eigen-decomposition $\boldsymbol{U}\left(\boldsymbol{\Gamma\Gamma}^T\right)\boldsymbol{U}^{-1}$

Definition of SVD

- Inversion

$$\left(\boldsymbol{XX}^T\right)^{-1} = \left(\boldsymbol{U}\left(\boldsymbol{\Gamma\Gamma}^T\right)\boldsymbol{U}^{-1}\right)^{-1} = \boldsymbol{U}\left(\boldsymbol{\Gamma\Gamma}^T\right)^{-1}\boldsymbol{U}^{-1}$$

  - $\boldsymbol{XX}^T$ is invertible iff $\boldsymbol{\Gamma\Gamma}^T$ is invertible

Why not invertible?

- Look deeper!

$$\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T = \boldsymbol{U}\begin{bmatrix} {\gamma_1}^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & {\gamma_D}^2 \end{bmatrix}\boldsymbol{U}^T \qquad \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-\mathbf{1}} = \boldsymbol{U}\begin{bmatrix} \frac{1}{{\gamma_1}^2} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{{\gamma_D}^2} \end{bmatrix}\boldsymbol{U}^T$$

  - Not invertible if there are zeros along the diagonal

How to make it invertible?

- Adding positive numbers to the diagonal!

$$\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T + \lambda \boldsymbol{I} = \boldsymbol{U}\begin{bmatrix} {\gamma_1}^2 + \lambda & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & {\gamma_D}^2 + \lambda \end{bmatrix}\boldsymbol{U}^T \qquad \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T + \lambda \boldsymbol{I}\right)^{-\mathbf{1}} = \boldsymbol{U}\begin{bmatrix} \frac{1}{{\gamma_1}^2 + \lambda} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{{\gamma_D}^2 + \lambda} \end{bmatrix}\boldsymbol{U}^T$$

- Closed-form solution $= \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T + \lambda \boldsymbol{I}\right)^{-\mathbf{1}}\widetilde{\boldsymbol{X}}\boldsymbol{y}$
  - Named "ridge regression"
  - Not the same $\widetilde{\boldsymbol{w}}^{\mathrm{LMS}}$ when $\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T$ is invertible

- We will justify why doing this makes sense later!

Practice!

- Show $\left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T + \lambda \boldsymbol{I}\right)^{-\mathbf{1}}\widetilde{\boldsymbol{X}}\boldsymbol{y}$ is the solution of $\underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_i [y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i]^2 + \lambda \|\widetilde{\boldsymbol{w}}\|_2^2$
  - That is, we introduce the prior knowledge that the solution should be close to **0**

- How to select λ?

Summary

- Linear regression (linear curve fitting)
  - Model input-output relationships via linear functions
- To learn regression model parameters
  - Given training data examples
  - Choose an error/loss function
- Linear least mean square (LMS)
  - Minimize sum of residual sum of square (RSS)
    - Calculus and Linear Algebra
  - Ridge regression for noninvertible cases

CSE 5523: Regression

The Ohio State University

Hw-1

- Will be released tonight (midnight)
  - You will receive Carmen's announcement
- Will be **due on 9/17** (midnight)
- Questions in both the programming set and the problem set
- Please use Piazza or come to the office hours for discussion
- Please start working on it ASAP

Today

Linear regression

- Review
- Linear regression (continued)
- Ridge regression
- A little more math

Nonlinear regression

Linear regression

- Training data: $D_{tr} = \left\{\left(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \mathbb{R}\right)\right\}_{n=1}^{N}$
- Models (hypothesis class):
  - $\hat{y} = h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w[0] = \widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}}$
  - $\widetilde{\boldsymbol{w}} = [w[0], w[1], \dots, w[D]]^T \in \mathbb{R}^{D+1}$; $\widetilde{\boldsymbol{x}} = [1, x[1], \dots, x[D]]^T \in \mathbb{R}^{D+1}$
  - Assumption: Linear relationship between input features and output label

$$\boldsymbol{X} \in \mathbb{R}^{D\times N} = [\boldsymbol{x}_1,\quad \boldsymbol{x}_2,\quad \boldsymbol{x}_3,\quad \dots,\quad \boldsymbol{x}_N] = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \boldsymbol{x}_3 & \dots & \boldsymbol{x}_N \\ | & | & | & & | \end{bmatrix} \qquad \boldsymbol{y} \in \mathbb{R}^N = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

$$\widetilde{\boldsymbol{X}} \in \mathbb{R}^{(D+1)\times N} = [\widetilde{\boldsymbol{x}}_1,\quad \widetilde{\boldsymbol{x}}_2,\quad \widetilde{\boldsymbol{x}}_3,\quad \dots,\quad \widetilde{\boldsymbol{x}}_N] = \begin{bmatrix} | & | & | & & | \\ \widetilde{\boldsymbol{x}}_1 & \widetilde{\boldsymbol{x}}_2 & \widetilde{\boldsymbol{x}}_3 & \dots & \widetilde{\boldsymbol{x}}_N \\ | & | & | & & | \end{bmatrix}$$

Linear regression

- Data visualization and task illustration

- Meaning:
  - Sale price = price-per-sqft * sqft + fixed_expense

- Learning goal?
  - Estimate parameters by minimizing the loss
  - **Residual sum of square (RSS):**
    - RSS($\widetilde{\boldsymbol{w}}$) = $\sum_i [y_i - h(\boldsymbol{x}_i)]^2$

Regression (house price):
From x (area), predict y (price)

Least mean square (LMS) solution

- $\widetilde{\boldsymbol{w}}^{\mathrm{LMS}} = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}}\, \mathrm{RSS}(\widetilde{\boldsymbol{w}}) = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_i [y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i]^2$

- Let's try to expand it!

$$
\begin{aligned}
&\mathrm{RSS}(\widetilde{\boldsymbol{w}}) \\
&= \sum_i \left(y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)^2 \\
&= \sum_i \left(y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)\left(y_i - \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}}\right) \\
&= \sum_i y_i{}^2 + \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} - 2 y_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} = \sum_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} - 2 y_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} + \text{constant}
\end{aligned}
$$

Least mean square (LMS) solution

- Continued

$$\text{RSS}(\widetilde{\boldsymbol{w}}) = \sum_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} - 2y_i \widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} + \text{constant}$$

$$= \widetilde{\boldsymbol{w}}^T \left(\sum_i \widetilde{\boldsymbol{x}}_i \widetilde{\boldsymbol{x}}_i{}^T \right) \widetilde{\boldsymbol{w}} - 2 \left(\sum_i y_i \widetilde{\boldsymbol{x}}_i{}^T \right) \widetilde{\boldsymbol{w}} + \text{constant}$$

$$= \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{X}} \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\left(\widetilde{\boldsymbol{X}}\boldsymbol{y}\right)^T \widetilde{\boldsymbol{w}} + \text{constant}$$

$$\boxed{\boldsymbol{X}\,\boldsymbol{X}^T = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \boldsymbol{x}_3 & \dots & \boldsymbol{x}_N \\ | & | & | & & | \end{bmatrix} \begin{bmatrix} -\boldsymbol{x}_1^T - \\ \vdots \\ -\boldsymbol{x}_N^T - \end{bmatrix} = \sum_{i=1}^N \begin{bmatrix} | \\ \boldsymbol{x}_i \\ | \end{bmatrix} \begin{bmatrix} -\boldsymbol{x}_i^T - \end{bmatrix}}$$

Least mean square (LMS) solution

- Normal equations
  - Taking derivative of RSS($\widetilde{\boldsymbol{w}}$) w.r.t. $\widetilde{\boldsymbol{w}}$, and setting it to zeros

$$\frac{\partial \mathrm{RSS}(\widetilde{\boldsymbol{w}})}{\partial \widetilde{\boldsymbol{w}}} = \frac{\partial\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\left(\widetilde{\boldsymbol{X}}\boldsymbol{y}\right)^T \widetilde{\boldsymbol{w}}\right)}{\partial \widetilde{\boldsymbol{w}}} = 2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\widetilde{\boldsymbol{X}}\boldsymbol{y} = \boldsymbol{0} \in \mathbb{R}^{(D+1)}$$

Closed-form $\widetilde{\boldsymbol{w}}^{\mathrm{LMS}} = \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-1}\widetilde{\boldsymbol{X}}\boldsymbol{y}$, since $2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} = 2\widetilde{\boldsymbol{X}}\boldsymbol{y}$ leads to $\widetilde{\boldsymbol{w}} = \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-1}\widetilde{\boldsymbol{X}}\boldsymbol{y}$

$$E(\boldsymbol{w}) = \boldsymbol{x}^T\boldsymbol{w}: \quad \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial E(\boldsymbol{w})}{\partial w[1]} \\ \vdots \\ \frac{\partial E(\boldsymbol{w})}{\partial w[D]} \end{bmatrix} = \boldsymbol{x}$$

$$E(\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{A}\boldsymbol{w}: \quad \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = \boldsymbol{A}\boldsymbol{w} + \boldsymbol{A}^T\boldsymbol{w} = \left(\boldsymbol{A} + \boldsymbol{A}^T\right)\boldsymbol{w}$$

More useful tools in the Matrix Cookbook!

Least mean square (LMS) solution: another way

$$
\begin{aligned}
&\mathrm{RSS}(\widetilde{\boldsymbol{w}}) \\
&= \sum_i \left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i - y_i\right)^2 \\
&= \sum_i \left(\widetilde{\boldsymbol{x}}_i{}^T \widetilde{\boldsymbol{w}} - y_i\right)^2 \\
&= \left[\left(\widetilde{\boldsymbol{x}}_1{}^T \widetilde{\boldsymbol{w}} - y_1\right), \ldots\ldots, \left(\widetilde{\boldsymbol{x}}_N{}^T \widetilde{\boldsymbol{w}} - y_N\right)\right] \begin{bmatrix} \widetilde{\boldsymbol{x}}_1{}^T \widetilde{\boldsymbol{w}} - y_1 \\ \vdots \\ \widetilde{\boldsymbol{x}}_N{}^T \widetilde{\boldsymbol{w}} - y_N \end{bmatrix} \\
&= \left\|\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - \mathbf{y}\right\|_2^2 \\
&= \left(\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - \mathbf{y}\right)^T \left(\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - \mathbf{y}\right) \\
&= \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{X}} \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\left(\widetilde{\boldsymbol{X}} \boldsymbol{y}\right)^T \widetilde{\boldsymbol{w}} + \text{constant}
\end{aligned}
$$

Mini summary

- Least mean square (LMS) has a closed-form solution
- Works for any model with **<u>linear</u>** <u>combination of parameters</u> to minimize RSS

- Bottleneck
  - $\left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-\mathbf{1}}$: computationally $O\left((D+1)^3\right)$
  - $\left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-\mathbf{1}}$: exists only if $\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T$ is of full rank: $N \geq D+1$. In other words, we need at least D+1 training data
  - Often, we need more than that to overcome noise and over-fitting

Recap: Eigen-decomposition (ED)

- Given a square matrix $\boldsymbol{A} \in R^{D \times D}$
- If $\boldsymbol{A}$ has D linearly independent eigenvectors $\{\boldsymbol{q}_1, \dots, \boldsymbol{q}_D\}$ and the corresponding eigenvalues $\{\lambda_1, \dots, \lambda_D\}$

$$\boldsymbol{A}\begin{bmatrix} | & & | \\ \boldsymbol{q}_1 & \dots & \boldsymbol{q}_D \\ | & & | \end{bmatrix} = \boldsymbol{A}\boldsymbol{Q} = \begin{bmatrix} | & & | \\ \lambda_1\boldsymbol{q}_1 & \dots & \lambda_D\boldsymbol{q}_D \\ | & & | \end{bmatrix} = \begin{bmatrix} | & & | \\ \boldsymbol{q}_1 & \dots & \boldsymbol{q}_D \\ | & & | \end{bmatrix}\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_D \end{bmatrix} = \boldsymbol{Q}\boldsymbol{\Lambda}$$

- Eigen-decomposition: $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$
  - $(\boldsymbol{A}\boldsymbol{Q})\boldsymbol{Q}^{-1} = (\boldsymbol{Q}\boldsymbol{\Lambda})\,\boldsymbol{Q}^{-1}$ then $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$
- Not all the square matrices have eigen-decomposition

Recap: Singular value decomposition (SVD)

- Singular value decomposition (SVD)

Non-negative

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Gamma}\boldsymbol{V}^T = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 & \boldsymbol{u}_3 & \dots & \boldsymbol{u}_D \\ | & | & | & & | \end{bmatrix} \begin{bmatrix} \gamma_1 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & \gamma_D & 0 \end{bmatrix} \begin{bmatrix} -\boldsymbol{v}_1^T- \\ \vdots \\ -\boldsymbol{v}_N^T- \end{bmatrix}$$

(D x N) (D x D) (D x N) (N x N)

$$\boldsymbol{U}^T\boldsymbol{U} = \begin{bmatrix} -\boldsymbol{u}_1^T- \\ \vdots \\ -\boldsymbol{u}_N^T- \end{bmatrix} \begin{bmatrix} | & | & | & & | \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 & \boldsymbol{u}_3 & \dots & \boldsymbol{u}_D \\ | & | & | & & | \end{bmatrix} = \boldsymbol{I} = \boldsymbol{U}\boldsymbol{U}^T$$

Orthonormal, unitary

$$\boldsymbol{V}^T\boldsymbol{V} = \begin{bmatrix} -\boldsymbol{v}_1^T- \\ \vdots \\ -\boldsymbol{v}_N^T- \end{bmatrix} \begin{bmatrix} | & | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 & \dots & \boldsymbol{v}_D \\ | & | & | & & | \end{bmatrix} = \boldsymbol{I} = \boldsymbol{V}\boldsymbol{V}^T$$

Orthonormal, unitary

Recap: SVD + ED

- Combination (suppose $\boldsymbol{X} \in R^{D\times N}$ and $\boldsymbol{X} = \boldsymbol{U\Gamma V}^T$)

$$\boldsymbol{XX}^T = \boldsymbol{U\Gamma V}^T\left(\boldsymbol{U\Gamma V}^T\right)^T = \boldsymbol{U\Gamma V}^T\left(\boldsymbol{V\Gamma}^T\boldsymbol{U}^T\right) = \boldsymbol{U\Gamma V}^T\boldsymbol{V\Gamma}^T\boldsymbol{U}^T = \boldsymbol{U\Gamma\Gamma}^T\boldsymbol{U}^T$$
$$= \boldsymbol{U}\left(\boldsymbol{\Gamma\Gamma}^T\right)\boldsymbol{U}^T = \boldsymbol{U}\left(\boldsymbol{\Gamma\Gamma}^T\right)\boldsymbol{U}^{-1}$$

  - $\boldsymbol{XX}^T$ has an eigen-decomposition $\boldsymbol{U}\left(\boldsymbol{\Gamma\Gamma}^T\right)\boldsymbol{U}^{-1}$

Definition of SVD

- Inversion

$$\left(\boldsymbol{XX}^T\right)^{-1} = \left(\boldsymbol{U}\left(\boldsymbol{\Gamma\Gamma}^T\right)\boldsymbol{U}^{-1}\right)^{-1} = \boldsymbol{U}\left(\boldsymbol{\Gamma\Gamma}^T\right)^{-1}\boldsymbol{U}^{-1}$$

  - $\boldsymbol{XX}^T$ is invertible iff $\boldsymbol{\Gamma\Gamma}^T$ is invertible

Why not invertible?

- Look deeper!

$$\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T = \boldsymbol{U}\begin{bmatrix} {\gamma_1}^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & {\gamma_D}^2 \end{bmatrix}\boldsymbol{U}^T \qquad \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-\mathbf{1}} = \boldsymbol{U}\begin{bmatrix} \frac{1}{{\gamma_1}^2} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{{\gamma_D}^2} \end{bmatrix}\boldsymbol{U}^T$$

  - Not invertible if there are zeros along the diagonal

How to make it invertible?

- Adding positive numbers to the diagonal!

$$\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T + \lambda \boldsymbol{I} = \boldsymbol{U}\begin{bmatrix} {\gamma_1}^2 + \lambda & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & {\gamma_D}^2 + \lambda \end{bmatrix}\boldsymbol{U}^T \qquad \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T + \lambda \boldsymbol{I}\right)^{-\mathbf{1}} = \boldsymbol{U}\begin{bmatrix} \frac{1}{{\gamma_1}^2 + \lambda} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{{\gamma_D}^2 + \lambda} \end{bmatrix}\boldsymbol{U}^T$$

- Closed-form solution $= \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T + \lambda \boldsymbol{I}\right)^{-\mathbf{1}}\widetilde{\boldsymbol{X}}\boldsymbol{y}$
  - Named "ridge regression"
  - Not the same $\widetilde{\boldsymbol{w}}^{\mathrm{LMS}}$ when $\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T$ is invertible

- We will justify why doing this makes sense later!

Practice!

- Show $\left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T + \lambda \boldsymbol{I}\right)^{-\mathbf{1}}\widetilde{\boldsymbol{X}}\boldsymbol{y}$ is the solution of $\underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_i [y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i]^2 + \lambda \|\widetilde{\boldsymbol{w}}\|_2^2$
  - That is, we introduce the prior knowledge that the solution should be close to **0**

- How to select λ?

Summary

- Linear regression (linear curve fitting)
  - Model input-output relationships via linear functions
- To learn regression model parameters
  - Given training data examples
  - Choose an error/loss function
- Linear least mean square (LMS)
  - Minimize sum of residual sum of square (RSS)
    - Calculus and Linear Algebra
  - Ridge regression for noninvertible cases

A little more math

- Goal:
  - We want to minimize RSS($\widetilde{\boldsymbol{w}}$) = $\sum_i [y_i - h(\boldsymbol{x}_i)]^2$
  - Normal equations tell us $\frac{\partial \text{RSS}(\widetilde{\boldsymbol{w}})}{\partial \widetilde{\boldsymbol{w}}} = 2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{w}} - 2\widetilde{\boldsymbol{X}}\boldsymbol{y} = \mathbf{0} \in \mathbb{R}^{(D+1)}$
  - Closed-form solution: $\widetilde{\boldsymbol{w}}^{\text{LMS}} = \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-\mathbf{1}}\widetilde{\boldsymbol{X}}\boldsymbol{y}$

- What does it mean by $\left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-\mathbf{1}}$ not invertible?
  - NO solutions for $\frac{\partial \text{RSS}(\widetilde{\boldsymbol{w}})}{\partial \widetilde{\boldsymbol{w}}} = \mathbf{0}$?
  - TOO MANY solutions for $\frac{\partial \text{RSS}(\widetilde{\boldsymbol{w}})}{\partial \widetilde{\boldsymbol{w}}} = \mathbf{0}$?

A little more math

- Answer:
  - D+1 unknown (degree of freedom), but < D+1 linear independent training data in "$\widetilde{\boldsymbol{X}}$"
  - TOO MANY choices of unknown to fit the training data

- Examples:
  - Learn $\hat{y} = h(\boldsymbol{x}) = w[1]x + w[0]$ from $(\boldsymbol{x}_1 = -1,\ y_1 = 1)$
    - Any pair of $(w[1], w[0])$ such that $w[1] - w[0] = -1$
  - Learn $\hat{y} = h(\boldsymbol{x}) = w[1]x + w[0]$ from $(\boldsymbol{x}_1 = -1,\ y_1 = 1)$ and $(\boldsymbol{x}_2 = -1,\ y_2 = 1)$
    - Any pair of $(w[1], w[0])$ such that $w[1] - w[0] = -1$
  - Learn $\hat{y} = h(\boldsymbol{x}) = w[1]x + w[0]$ from $(\boldsymbol{x}_1 = -1,\ y_1 = 1)$ and $(\boldsymbol{x}_2 = 1,\ y_2 = 1)$
    - $w[1] - w[0] = -1$ AND $w[1] + w[0] = 1$, we get $w[1] = 0, w[0] = 1$

What does ridge regression do?

- Within many solutions, identify one!

- Geometric illustration: $w[1] - w[0] = -1$

A little more math: convex functions

- A function $g(\boldsymbol{w})$ is convex iff $g(t\boldsymbol{w}_1 + (1-t)\boldsymbol{w}_2) \leq tg(\boldsymbol{w}_1) + (1-t)g(\boldsymbol{w}_2)$
  - $\forall\, \boldsymbol{w}_1, \boldsymbol{w}_2;\ \forall t \in [0,1]$

- If $g(\boldsymbol{w})$ is differentiable, then
  - $g(\boldsymbol{w}_2) \geq g(\boldsymbol{w}_1) + \langle \nabla g(\boldsymbol{w}_1), \boldsymbol{w}_2 - \boldsymbol{w}_1 \rangle$
  - $\forall\, \boldsymbol{w}_1, \boldsymbol{w}_2$

- Gradient $\nabla g(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial g(\boldsymbol{w})}{\partial w[1]} \\ \vdots \\ \frac{\partial g(\boldsymbol{w})}{\partial w[D]} \end{bmatrix}$

A little more math: convex functions

- A function $g(\boldsymbol{w})$ is convex iff $g(t\boldsymbol{w}_1 + (1-t)\boldsymbol{w}_2) \leq tg(\boldsymbol{w}_1) + (1-t)g(\boldsymbol{w}_2)$
  - $\forall\, \boldsymbol{w}_1, \boldsymbol{w}_2;\ \forall t \in [0,1]$

- If $g(\boldsymbol{w})$ is twice differentiable, then
  - $\mathbf{H}g(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial^2 g(\boldsymbol{w})}{\partial w[1]\partial w[1]} & \cdots & \frac{\partial^2 g(\boldsymbol{w})}{\partial w[1]\partial w[D]} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 g(\boldsymbol{w})}{\partial w[D]\partial w[1]} & \cdots & \frac{\partial^2 g(\boldsymbol{w})}{\partial w[D]\partial w[D]} \end{bmatrix} \succcurlyeq \mathbf{0}, \forall \boldsymbol{w}$
  - Hessian matrix is positive semi-definite (PSD), $\forall \boldsymbol{w}$
  - Physical meaning?

A little more math: convex functions

- Properties: Any local optimum is a global optimum
  - Gradients (partial derivatives) = 0 means local optimums

- Is RSS($\widetilde{\boldsymbol{w}}$) convex?
  - $\text{RSS}(\widetilde{\boldsymbol{w}}) = \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{X}} \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\left(\widetilde{\boldsymbol{X}}\boldsymbol{y}\right)^T \widetilde{\boldsymbol{w}} + \text{constant}$
  - $\nabla\text{RSS}(\widetilde{\boldsymbol{w}}) = 2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\,\widetilde{\boldsymbol{X}}\boldsymbol{y}$
  - $\mathbf{H}\text{RSS}(\widetilde{\boldsymbol{w}}) = 2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T$
    - PSD because $\boldsymbol{v}^T\left(2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)\boldsymbol{v} = 2\left(\widetilde{\boldsymbol{X}}^T\boldsymbol{v}\right)^T \widetilde{\boldsymbol{X}}^T\boldsymbol{v} = 2\left\|\widetilde{\boldsymbol{X}}^T\boldsymbol{v}\right\|_2^2 \geq \mathbf{0}, \forall \boldsymbol{v} \in \mathbb{R}^{D+1} \neq \mathbf{0}$

Caution

- When the function to be minimized is convex:
  - Any local optimum is a global optimum
  - However, it does not imply there exists closed-form solutions
  - Indeed, most of the (convex) optimization problems do not have closed-form solutions

Nonlinear regression

- Definition: fit a nonlinear curve to data
  - $h(\boldsymbol{x})$ is linear iff $h(a\boldsymbol{x}_1 + b\boldsymbol{x}_2) = a \times h(\boldsymbol{x}_1) + b \times h(\boldsymbol{x}_2)$
  - $\forall\ \boldsymbol{x}_1, \boldsymbol{x}_2; \forall\ a, b \in \mathbb{R}$
  - Otherwise, it is nonlinear
  - Example: $h(\boldsymbol{x}) = \ \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}$ is linear

- Questions:
  - How to represent a nonlinear function?
  - What are the parameters?
  - How to estimate the parameters?

Nonlinear regression: representation

- Method 1: directly use nonlinear functions
  - $h(x; w) = e^{wx}$
  - $h(x; w) = \sin(wx)$
  - …

Explicitly indicate the parameters; alternatively, $h_w(x)$

- Method 2: use nonlinear feature transforms or mappings
  - $\boldsymbol{\phi}(\boldsymbol{x}): \boldsymbol{x} \in \mathbb{R}^D \mapsto \boldsymbol{z} \in \mathbb{R}^M$
  - M is the dimensionality of the transformed feature $\boldsymbol{z}$ (or $\boldsymbol{\phi}(\boldsymbol{x})$); can be larger, smaller than, or the same as D
  - Notation $z[m] = \phi_m(\boldsymbol{x})$
  - ϕ_m is called a nonlinear basis function
  - $\boldsymbol{\phi}(\boldsymbol{x})$ can be used for other ML algorithms (e.g., nearest neighbor) as well

Nonlinear regression: representation

- Examples of $\phi_m(\boldsymbol{x})$:
  - $z[m] = (x[m])^2$
  - $z[m] = sin(x[m])$
  - $z[m] = log(x[m])$
  - $z[m] = e^{x[m]}$

  - When $x \in \mathbb{R}$
    - $z[m] = (x)^{m-1}$
    - $\boldsymbol{z} \in \mathbb{R}^M$ can already encode the dummy variable 1

Nonlinear regression: representation

- Previously we use $h(\boldsymbol{x}) = \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}$

- Now we use $h(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) = \sum_m w[m]\phi_m(\boldsymbol{x})$, where $\boldsymbol{w} \in \mathbb{R}^M$
  - In nonlinear regression, the number of parameters is usually different from D or D+1

- $\mathrm{RSS}(\boldsymbol{w}) = \sum_i [y_i - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)]^2$

- $\boldsymbol{w}^{\mathrm{LMS}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \mathrm{RSS}(\boldsymbol{w}) = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_i [y_i - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)]^2$

Nonlinear regression: parameter estimation

- $\boldsymbol{w}^{\mathrm{LMS}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \mathrm{RSS}(\boldsymbol{w}) = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_i [y_i - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)]^2$

$$\boldsymbol{\Phi} \in \mathbb{R}^{M \times N} = [\boldsymbol{\phi}(\boldsymbol{x}_1) \quad \dots \quad \boldsymbol{\phi}(\boldsymbol{x}_N)] = \begin{bmatrix} | & & | \\ \boldsymbol{\phi}(\boldsymbol{x}_1) & \dots & \boldsymbol{\phi}(\boldsymbol{x}_N) \\ | & & | \end{bmatrix}$$

- Closed form solution $\boldsymbol{w}^{\mathrm{LMS}}$: $\left(\widetilde{\boldsymbol{\Phi}}\widetilde{\boldsymbol{\Phi}}^T\right)^{-1}\widetilde{\boldsymbol{\Phi}}\boldsymbol{y}$
- Question:
  - How many data are needed?
  - Is $\mathrm{RSS}(\boldsymbol{w})$ still convex w.r.t. $\boldsymbol{w}$?

Nonlinear regression: example

- Polynomial regression for 1-D x, i.e., $h(x) = \sum_{m=0}^{M} w[m]x^m = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)$

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ \vdots \\ x^M \end{bmatrix}$$

- Special case: $D_{tr} = \left\{\left(x_n \in \mathbb{R},\ y_n = {x_n}^2\right)\right\}_{n=1}^{N}$

Nonlinear regression: example

- Training data are generated from a sin function + noise
  - Under-fitting

    h too simple

  - Over-fitting

    h too complicated

    training error = 0

[Slides: from USC CSCI567]

Nonlinear regression: example

- Training data are generated from a sin function + noise
  - Just good enough

[Slides: from USC CSCI567]

Training vs. testing

- Models learned from the training data should be applicable to test data

$$D_{train} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$$

$$D_{test} = \{(\boldsymbol{x}_1, y_{N+1}), \dots, (\boldsymbol{x}_N, y_{N+M})\}$$

In training, we only see training data!

Choosing a more complicated hypothesis class does not necessarily lead to lower test errors!

Over-fitting vs. hypothesis class (brief)

- Over-fitting:
  - Small training error
  - Large test error (even larger than some other “simpler” models)

- How to quantify hypothesis class’ complexity?
  - *K*NN: smaller *K*, larger complexity
  - Polynomial regression: larger *M*, larger complexity

- How to choose?

Nonlinear regression: example

- 2nd-order polynomial regression for 2-D $\boldsymbol{x}$:

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x[1] \\ x[2] \\ x[1]^2 \\ x[2]^2 \\ x[1]x[2] \end{bmatrix}$$

CSE 5523: Optimization

The Ohio State University

Hw-1

- Released on 9/3 midnight
- Due on 9/17 midnight
- Two questions in the programming set; five questions in the problem set
- Please use Piazza or come to the office hours for discussion
- Please start working on it ASAP

Next Tuesday (9/10)

- I will be traveling
- The class will be through Zoom, pre-recorded, or canceled.
  - Will announce it soon.

- Midterm
  - Will announce the date soon, by this weekend.

Today

Review

Nonlinear regression

Optimization

- Gradient descent
- Newton's method

Least mean square (LMS) solution

- $\widetilde{\boldsymbol{w}}^{\mathrm{LMS}} = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}}\, \mathrm{RSS}(\widetilde{\boldsymbol{w}}) = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_i [y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i]^2$

- Into a matrix representation

$$\mathrm{RSS}(\widetilde{\boldsymbol{w}}) = \sum_i \left(y_i - \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)^2 = \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{X}} \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\left(\widetilde{\boldsymbol{X}}\boldsymbol{y}\right)^T \widetilde{\boldsymbol{w}} + \text{constant}$$

- Normal equations

$$\frac{\partial \mathrm{RSS}(\widetilde{\boldsymbol{w}})}{\partial \widetilde{\boldsymbol{w}}} = 2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\widetilde{\boldsymbol{X}}\boldsymbol{y} = \boldsymbol{0} \in \mathbb{R}^{(D+1)}$$

Closed-form $\widetilde{\boldsymbol{w}}^{\mathrm{LMS}} = \left(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)^{-1} \widetilde{\boldsymbol{X}}\boldsymbol{y}$

A little more math: convex functions

- A function $g(\boldsymbol{w})$ is convex iff $g(t\boldsymbol{w}_1 + (1-t)\boldsymbol{w}_2) \leq tg(\boldsymbol{w}_1) + (1-t)g(\boldsymbol{w}_2)$
  - $\forall\, \boldsymbol{w}_1, \boldsymbol{w}_2;\ \forall t \in [0,1]$

- If $g(\boldsymbol{w})$ is twice differentiable, then
  - $\mathbf{H}g(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial^2 g(\boldsymbol{w})}{\partial w[1]\partial w[1]} & \cdots & \frac{\partial^2 g(\boldsymbol{w})}{\partial w[1]\partial w[D]} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 g(\boldsymbol{w})}{\partial w[D]\partial w[1]} & \cdots & \frac{\partial^2 g(\boldsymbol{w})}{\partial w[D]\partial w[D]} \end{bmatrix} \succcurlyeq \mathbf{0}, \forall \boldsymbol{w}$
  - Hessian matrix is positive semi-definite (PSD), $\forall \boldsymbol{w}$

A little more math: convex functions

- Properties: Any local optimum is a global optimum
  - Gradients (partial derivatives) = 0 means local optimums

- Is RSS($\widetilde{\boldsymbol{w}}$) convex?
  - $\text{RSS}(\widetilde{\boldsymbol{w}}) = \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{X}} \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\left(\widetilde{\boldsymbol{X}}\boldsymbol{y}\right)^T \widetilde{\boldsymbol{w}} + \text{constant}$
  - $\nabla\text{RSS}(\widetilde{\boldsymbol{w}}) = 2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{w}} - 2\,\widetilde{\boldsymbol{X}}\boldsymbol{y}$
  - $\mathbf{H}\text{RSS}(\widetilde{\boldsymbol{w}}) = 2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T$
    - PSD because $\boldsymbol{v}^T\left(2\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^T\right)\boldsymbol{v} = 2\left(\widetilde{\boldsymbol{X}}^T\boldsymbol{v}\right)^T \widetilde{\boldsymbol{X}}^T\boldsymbol{v} = 2\left\|\widetilde{\boldsymbol{X}}^T\boldsymbol{v}\right\|_2^2 \geq \mathbf{0}, \forall \boldsymbol{v} \in \mathbb{R}^{D+1} \neq \mathbf{0}$

Convex functions

- When the function to be minimized is convex:
  - Any local optimum is a global optimum
  - However, it does not imply there exists closed-form solutions
  - Indeed, most of the (convex) optimization problems do not have closed-form solutions

Nonlinear regression

- Definition: fit a nonlinear curve to data
  - $h(\boldsymbol{x})$ is linear iff $h(a\boldsymbol{x}_1 + b\boldsymbol{x}_2) = a \times h(\boldsymbol{x}_1) + b \times h(\boldsymbol{x}_2)$
  - $\forall\ \boldsymbol{x}_1, \boldsymbol{x}_2; \forall\ a, b \in \mathbb{R}$
  - Otherwise, it is nonlinear
  - Example: $h(\boldsymbol{x}) = \ \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}$ is linear

- Questions:
  - How to represent a nonlinear function?
  - What are the parameters?
  - How to estimate the parameters?

Nonlinear regression: representation

- Method 1: directly use nonlinear functions
  - $h(x; w) = e^{wx}$
  - $h(x; w) = \sin(wx)$
  - …

Explicitly indicate the parameters; alternatively, $h_w(x)$

- Method 2: use nonlinear feature transforms or mappings
  - $\boldsymbol{\phi}(\boldsymbol{x}): \boldsymbol{x} \in \mathbb{R}^D \mapsto \boldsymbol{z} \in \mathbb{R}^M$
  - M is the dimensionality of the transformed feature $\boldsymbol{z}$ (or $\boldsymbol{\phi}(\boldsymbol{x})$); can be larger, smaller than, or the same as D
  - Notation $z[m] = \phi_m(\boldsymbol{x})$
  - ϕ_m is called a nonlinear basis function
  - $\boldsymbol{\phi}(\boldsymbol{x})$ can be used for other ML algorithms (e.g., nearest neighbor) as well

Nonlinear regression: representation

- Examples of $\phi_m(\boldsymbol{x})$:
  - $z[m] = (x[m])^2$
  - $z[m] = sin(x[m])$
  - $z[m] = log(x[m])$
  - $z[m] = e^{x[m]}$

  - When $x \in \mathbb{R}$
    - $z[m] = (x)^{m-1}$
    - $\boldsymbol{z} \in \mathbb{R}^M$ can already encode the dummy variable 1

Nonlinear regression: representation

- Previously we use $h(\boldsymbol{x}) = \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}$

- Now we use $h(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) = \sum_m w[m]\phi_m(\boldsymbol{x})$, where $\boldsymbol{w} \in \mathbb{R}^M$
  - In nonlinear regression, the number of parameters is usually different from D or D+1

- $\mathrm{RSS}(\boldsymbol{w}) = \sum_i [y_i - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)]^2$

- $\boldsymbol{w}^{\mathrm{LMS}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \mathrm{RSS}(\boldsymbol{w}) = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_i [y_i - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)]^2$

Nonlinear regression: parameter estimation

- $\boldsymbol{w}^{\mathrm{LMS}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \mathrm{RSS}(\boldsymbol{w}) = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_i [y_i - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)]^2$

$$\boldsymbol{\Phi} \in \mathbb{R}^{M \times N} = [\boldsymbol{\phi}(\boldsymbol{x}_1) \quad \dots \quad \boldsymbol{\phi}(\boldsymbol{x}_N)] = \begin{bmatrix} | & & | \\ \boldsymbol{\phi}(\boldsymbol{x}_1) & \dots & \boldsymbol{\phi}(\boldsymbol{x}_N) \\ | & & | \end{bmatrix}$$

- Closed form solution $\boldsymbol{w}^{\mathrm{LMS}}$: $\left(\widetilde{\boldsymbol{\Phi}}\widetilde{\boldsymbol{\Phi}}^T\right)^{-1}\widetilde{\boldsymbol{\Phi}}\boldsymbol{y}$
- Question:
  - How many data are needed?
  - Is $\mathrm{RSS}(\boldsymbol{w})$ still convex?

Nonlinear regression: example

- Polynomial regression for 1-D x, i.e., $h(x) = \sum_{m=0}^{M} w[m]x^m = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)$

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ \vdots \\ x^M \end{bmatrix}$$

- Special case: $D_{tr} = \left\{\left(x_n \in \mathbb{R},\ y_n = {x_n}^2\right)\right\}_{n=1}^{N}$

Nonlinear regression: example

- Training data are generated from a sin function + noise
  - Under-fitting

    h too simple

  - Over-fitting

    h too complicated

    training error = 0

[Slides: from USC CSCI567]

Nonlinear regression: example

- Training data are generated from a sin function + noise
  - Just good enough

[Slides: from USC CSCI567]

Training vs. testing

- Models learned from the training data should be applicable to test data

$$D_{train} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$$

$$D_{test} = \{(\boldsymbol{x}_1, y_{N+1}), \dots, (\boldsymbol{x}_N, y_{N+M})\}$$

In training, we only see training data!

Choosing a more complicated hypothesis class does not necessarily lead to lower test errors!

Over-fitting vs. hypothesis class (brief)

- Over-fitting:
  - Small training error
  - Large test error (even larger than some other “simpler” models)

- How to quantify hypothesis class’ complexity?
  - *K*NN: smaller *K*, larger complexity
  - Polynomial regression: larger *M*, larger complexity

- How to choose?

Nonlinear regression: example

- 2nd-order polynomial regression for 2-D $\boldsymbol{x}$:

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x[1] \\ x[2] \\ x[1]^2 \\ x[2]^2 \\ x[1]x[2] \end{bmatrix}$$

In finding the solution (estimating parameters) …

- The training data are fixed
- The variables are the parameters $\boldsymbol{w}$ to estimate
- A "machine learning" algorithm finds $\boldsymbol{w}$ to minimize the training loss
  - **Question: how to do this?**
- For the "users", we care about given a future x, what the prediction will be

$$E(\boldsymbol{w}) = \sum_i [y_i - h(\boldsymbol{x}_i; \boldsymbol{w})]^2$$

Different figures

$h(x) = wx$ for a certain x

$E(w) = \sum_i (wx_i - y_i)^2$

Convex optimizations require the error function, not the prediction function, to be convex!

Linear w.r.t. w!

Convex

Different figures

Nonlinear regression since $h(x) = wx^2$ is not linear w.r.t. x.

Still linear w.r.t. w!

Convex

Different figures

Linear regression since
$h(x) = w^2 x$ is linear w.r.t. x.

Not linear w.r.t. w!

Maybe not convex

When $h(x; w)$ is nonlinear w.r.t. w

- $\mathrm{RSS}(\boldsymbol{w}) = \sum_i \left(y_i - h(\boldsymbol{x}_i; \boldsymbol{w})\right)^2$ is no longer parabolic w.r.t. the parameters $\boldsymbol{w}$
  - **May be nonconvex:** local minimums are not always global minimums
  - **May have no closed-form solution:** need numerical approximation

$$E(w) = \sum_i (wx_i - y_i)^2$$

- $\frac{\partial E}{\partial w} = 0$ means global minimums
- Closed-forms to find w for $\frac{\partial E}{\partial w} = 0$

When $h(x; w)$ is nonlinear w.r.t. w

- $\mathrm{RSS}(\boldsymbol{w}) = \sum_i \left(y_i - h(\boldsymbol{x}_i; \boldsymbol{w})\right)^2$ is no longer parabolic w.r.t. the parameters $\boldsymbol{w}$
  - **May be nonconvex:** local minimums are not always global minimums
  - **May have no closed-form solution:** need numerical approximation

$$E(w) = \sum_i (h(x_i; w) - y_i)^2$$

- $\frac{\partial E}{\partial w} = 0$ does not mean global minimums
- Maybe no closed-forms to find w for $\frac{\partial E}{\partial w} = 0$

When other loss functions are used

- The training loss $E(\boldsymbol{w})$ may not be parabolic
  - May have no closed-form solution for $\frac{\partial E}{\partial \boldsymbol{w}} = 0$
- The training loss $E(\boldsymbol{w})$ may not be convex
  - $\frac{\partial E}{\partial \boldsymbol{w}} = 0$ does not mean global minimums

- How are we going to do parameter estimation?

Gradient descent (GD)

- A simple (and fast?) method for optimization
  - $h(\boldsymbol{x}; \boldsymbol{w})$ linear or nonlinear w.r.t. parameters $\boldsymbol{w}$
  - Sum or square losses or other losses

- A general method to find local minimum for a multi-parameter error function
  - Search through "parameter space" to find the minimum error

- Main idea
  - Evaluate the error function for current guess of parameters
  - Determine change in parameters $\Delta\boldsymbol{w}$ that decreases error
    - From gradient of error function
  - Update parameters with this new change
  - Repeat process until converge (or hit max iterations)

Problem visualization

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \times (\text{derivative direction})$$

Problem visualization

Local approximation (let $\boldsymbol{w}$ be M-dim)

- Taylor series expansion

Constant term

Linear (first-order) term

Quadratic (second-order) term

$$E(\boldsymbol{w}) = \boxed{E(\boldsymbol{w}')} + \boxed{\nabla E(\boldsymbol{w}')^T\,(\boldsymbol{w} - \boldsymbol{w}')} + \boxed{\frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}')^T \mathbf{H}_E(\boldsymbol{w}')(\boldsymbol{w} - \boldsymbol{w}')} + \textit{H.O.T.}$$

Variable to solve

Current guess

Variation "around" w'

- Rewrite $\boldsymbol{w} = (\boldsymbol{w}' + \Delta\boldsymbol{w})$

$$E(\boldsymbol{w}' + \Delta\boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T \Delta\boldsymbol{w} + \frac{1}{2}\Delta\boldsymbol{w}^T \mathbf{H}_E(\boldsymbol{w}')\Delta\boldsymbol{w} + \text{ H.O.T.}$$

$\Delta\boldsymbol{w}$: the movement from $\boldsymbol{w}'$

Local approximation (let $\boldsymbol{w}$ be M-dim)

- Visualization
  - First order: $E'(\boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T (\boldsymbol{w} - \boldsymbol{w}')$

  - Second order: $E'(\boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T (\boldsymbol{w} - \boldsymbol{w}')$
    $+ \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}')^T \mathbf{H}_E(\boldsymbol{w}')(\boldsymbol{w} - \boldsymbol{w}')$
    - The approximation $E'(\boldsymbol{w})$ is quadratic!
    - If $E(\boldsymbol{w})$ is strictly convex, its Hessian matrix $\mathbf{H}_E(\boldsymbol{w}') \succ 0$: closed forms to minimize the approximation $E'(\boldsymbol{w})$

Why does gradient descent work?

- Recall the Taylor expansion

$$E(\boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T (\boldsymbol{w} - \boldsymbol{w}') + \text{ H.O.T. } = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T \Delta \boldsymbol{w} + \text{ H.O.T.}$$

- Approximate $E(\boldsymbol{w})$ by the linear terms: $E'(\boldsymbol{w}' + \Delta \boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T \Delta \boldsymbol{w}$

- Which direction, i.e., $\Delta \boldsymbol{w}$ with $\|\Delta \boldsymbol{w}\|_2^2 = \mathbf{1}$, can minimize $E'(\boldsymbol{w}' + \Delta \boldsymbol{w})$?
  - Answer: direction of $\Delta \boldsymbol{w} = -\dfrac{\nabla E(\boldsymbol{w}')}{\|\nabla E(\boldsymbol{w}')\|_2}$
  - Interpretation: move in the direction of $-\nabla E(\boldsymbol{w}')$ is the most efficient for minimization
  - Only works for small $\Delta \boldsymbol{w}$ (so a scale η) due to the nature of linear Taylor expansion

The GD Algorithm

Initialize parameters: $\hat{\boldsymbol{w}}^{(0)} = [w[1], w[2], \ldots, w[M]]^T$

Error function: $E(\boldsymbol{w})$

Compute gradients:

$$\nabla E\left(\hat{\boldsymbol{w}}^{(t)}\right) = \begin{bmatrix} \frac{\partial E\left(\hat{\boldsymbol{w}}^{(t)}\right)}{\partial w[1]} \\ \vdots \\ \frac{\partial E\left(\hat{\boldsymbol{w}}^{(t)}\right)}{\partial w[M]} \end{bmatrix}$$

Update parameters: $\hat{\boldsymbol{w}}^{(t+1)} = \hat{\boldsymbol{w}}^{(t)} - \eta^{(t)} \cdot \nabla E\left(\hat{\boldsymbol{w}}^{(t)}\right)$

Learning rate

[Repeat until converges]

2-order Taylor expansion approximation

- Taylor series expansion ($\Delta\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{w}'$)

$$E(\boldsymbol{w}' + \Delta\boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T \Delta\boldsymbol{w} + \frac{1}{2}\Delta\boldsymbol{w}^T \mathbf{H}_E(\boldsymbol{w}')\Delta\boldsymbol{w} + \text{H.O.T.}$$

- Hessian matrix $\mathbf{H}_E(\boldsymbol{w}')$

$$\mathbf{H}_E(\boldsymbol{w}') = \begin{bmatrix} \frac{\partial^2 E(\boldsymbol{w}')}{\partial w[1]\partial w[1]} & \cdots & \frac{\partial^2 E(\boldsymbol{w}')}{\partial w[1]\partial w[M]} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\boldsymbol{w}')}{\partial w[M]\partial w[1]} & \cdots & \frac{\partial^2 E(\boldsymbol{w}')}{\partial w[M]\partial w[M]} \end{bmatrix}$$

- Newton's method
  - For any kind of "convex" loss $E(\boldsymbol{w})$
  - Quadratic approximation of $E(\boldsymbol{w})$ at $\boldsymbol{w}'$: ignore H.O.T.
  - Minimize the quadratic approximation to get $\Delta\boldsymbol{w} = -\mathbf{H}_E{}^{-1}(\boldsymbol{w}')\nabla E(\boldsymbol{w}')$
  - Update $\boldsymbol{w}'$ by $\boldsymbol{w}' + \eta^{(t)}\Delta\boldsymbol{w}$
- Pros: faster convergence; Cons: $\mathbf{H}_E(\boldsymbol{w}')$ and $\mathbf{H}_E{}^{-1}(\boldsymbol{w}')$ are time consuming

The Newton's method

Initialize parameters: $\widehat{\boldsymbol{w}}^{(0)} = [w[1], w[2], \dots, w[M]]^T$

Error function: $E(\boldsymbol{w})$

Compute gradients and Hessian:

$$\nabla E\left(\widehat{\boldsymbol{w}}^{(t)}\right) = \begin{bmatrix} \frac{\partial E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[1]} \\ \vdots \\ \frac{\partial E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[M]} \end{bmatrix}, \quad \mathbf{H}_E\left(\widehat{\boldsymbol{w}}^{(t)}\right) = \begin{bmatrix} \frac{\partial^2 E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[1]\partial w[1]} & \cdots & \frac{\partial^2 E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[1]\partial w[M]} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[M]\partial w[1]} & \cdots & \frac{\partial^2 E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[M]\partial w[M]} \end{bmatrix}$$

Update parameters:

$$\widehat{\boldsymbol{w}}^{(t+1)} = \widehat{\boldsymbol{w}}^{(t)} - \eta^{(t)} \mathbf{H}_E{}^{-1}(\boldsymbol{w}')\nabla E(\boldsymbol{w}')$$

Learning rate

[Repeat until converges]

Summary

- Pros of GD:
  - No expensive matrix operations (i.e., inversion)
  - Non-RSS error functions (if gradient is well formed)
  - Easy to parallelize: $E(\boldsymbol{w}) = \sum_i \ell(h(\boldsymbol{x}_i; \boldsymbol{w}), y_i) \implies \nabla E(\boldsymbol{w}) = \sum_i \nabla \ell(h(\boldsymbol{x}_i; \boldsymbol{w}), y_i)$

- Cons of GD:
  - Can be misled by local minima and stuck at larger-error valleys
  - Can get stuck on a flat plain/flat region of the error function
  - Can overshoot back and forth (oscillate from side to side)

- Helpers
  - Use multiple random-restarts with random initial parameters
  - Methods exist for selecting appropriate (best) learning rate at each iteration

Summary

- Pros of Newton's method:
  - Non-RSS error functions (if gradient and Hessian are well formed)
  - If $\boldsymbol{w}'$ is well-chosen, it can converge faster, since 2nd-order Taylor has more information.

- Cons of Newton's method:
  - Hessian and its inverse are time consuming
  - Divergence often happens if the function is flat or almost flat with respect to some dimension. In that case the second derivatives are close to zero, and their inverse becomes very large, resulting in gigantic steps.

- Helpers
  - Compute only the diagonal Hessian: $\Delta\boldsymbol{w} = -H^{-1}(\boldsymbol{w}')\nabla E(\boldsymbol{w}')$
  - GD and then Newton's method

Question

- How many steps do you need to optimize $E(w) = \sum_i (wx_i - y_i)^2$?
  - GD?
  - Newton's method?

CSE 5523: Perceptron

The Ohio State University

Hw-1

- Due on 9/17 midnight
- Two questions in the programming set; five questions in the problem set
- Please use Piazza or come to the office hours for discussion
- Please start working on the homework ASAP

Other reminders

- Midterm
  - 10/17 in class
- Office hour this Thursday
  - 6:00 pm – 7:30 pm
- Piazza
  - Still many of you not in Piazza
  - Please find the link and access code in Carmen Syllabus or Carmen Announcement

Today

Optimization

- Gradient descent (continued)
- Newton's method

Linear classifier

- Definition
- First algorithm: perceptron
- Reading: pocket algorithm

Probability refresher (self reading)

Gradient descent (GD)

- A simple (and fast?) method for optimization
  - $h(\boldsymbol{x}; \boldsymbol{w})$ linear or nonlinear w.r.t. parameters $\boldsymbol{w}$
  - Sum or square losses or other losses

$$E(\boldsymbol{w}) = \sum_i \ell(h(\boldsymbol{x}_i; \boldsymbol{w}), \quad y_i)$$

- A general method to find <u>local minimum</u> for a multi-parameter error function
  - Search through "parameter space" to find the minimum error

- Main idea
  - Evaluate the error function for <u>current guess</u> of parameters
  - Determine <u>change in parameters</u> $\Delta\boldsymbol{w}$ that <u>decreases error</u>
    - From <u>gradient</u> of error function
  - Update parameters with this new change
  - Repeat process until converge (or hit max iterations)

Problem visualization

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \times (\text{derivative direction})$$

Local approximation (let $\boldsymbol{w}$ be M-dim)

- Taylor series expansion

Constant term

Linear (first-order) term

Quadratic (second-order) term

$$E(\boldsymbol{w}) = \boxed{E(\boldsymbol{w}')} + \boxed{\nabla E(\boldsymbol{w}')^T\,(\boldsymbol{w} - \boldsymbol{w}')} + \boxed{\frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}')^T \mathbf{H}_E(\boldsymbol{w}')(\boldsymbol{w} - \boldsymbol{w}')} + \textit{H.O.T.}$$

Variable to solve

Current guess

Variation "around" w'

- Rewrite $\boldsymbol{w} = (\boldsymbol{w}' + \Delta\boldsymbol{w})$

$$E(\boldsymbol{w}' + \Delta\boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T \Delta\boldsymbol{w} + \frac{1}{2}\Delta\boldsymbol{w}^T \mathbf{H}_E(\boldsymbol{w}')\Delta\boldsymbol{w} + \text{ H.O.T.}$$

$\Delta\boldsymbol{w}$: the movement from $\boldsymbol{w}'$

Local approximation (let $\boldsymbol{w}$ be M-dim)

- Visualization
  - First order: $E'(\boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T (\boldsymbol{w} - \boldsymbol{w}')$

  - Second order: $E'(\boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T (\boldsymbol{w} - \boldsymbol{w}')$
    $+ \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}')^T \mathbf{H}_E(\boldsymbol{w}')(\boldsymbol{w} - \boldsymbol{w}')$
    - The approximation $E'(\boldsymbol{w})$ is quadratic!
    - If $E(\boldsymbol{w})$ is strictly convex, its Hessian matrix $\mathbf{H}_E(\boldsymbol{w}') \succ 0$: closed forms to minimize the approximation $E'(\boldsymbol{w})$

Why does gradient descent work?

- Recall the Taylor expansion

$$E(\boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T (\boldsymbol{w} - \boldsymbol{w}') + \text{ H.O.T. } = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T \Delta \boldsymbol{w} + \text{ H.O.T.}$$

- Approximate $E(\boldsymbol{w})$ by the linear terms: $E'(\boldsymbol{w}' + \Delta \boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T \Delta \boldsymbol{w}$

- Which direction, i.e., $\Delta \boldsymbol{w}$ with $\|\Delta \boldsymbol{w}\|_2^2 = \mathbf{1}$, can minimize $E'(\boldsymbol{w}' + \Delta \boldsymbol{w})$?
  - Answer: direction of $\Delta \boldsymbol{w} = -\dfrac{\nabla E(\boldsymbol{w}')}{\|\nabla E(\boldsymbol{w}')\|_2}$
  - Interpretation: move in the direction of $-\nabla E(\boldsymbol{w}')$ is the most efficient for minimization
  - Only works for small $\Delta \boldsymbol{w}$ (so a scale η) due to the nature of linear Taylor expansion

The GD Algorithm

Initialize parameters: $\hat{\boldsymbol{w}}^{(0)} = [w[1], w[2], \ldots, w[M]]^T$

Error function: $E(\boldsymbol{w})$

Compute gradients:

$$\nabla E\left(\hat{\boldsymbol{w}}^{(t)}\right) = \begin{bmatrix} \frac{\partial E\left(\hat{\boldsymbol{w}}^{(t)}\right)}{\partial w[1]} \\ \vdots \\ \frac{\partial E\left(\hat{\boldsymbol{w}}^{(t)}\right)}{\partial w[M]} \end{bmatrix}$$

Update parameters: $\hat{\boldsymbol{w}}^{(t+1)} = \hat{\boldsymbol{w}}^{(t)} - \eta^{(t)} \cdot \nabla E\left(\hat{\boldsymbol{w}}^{(t)}\right)$

Learning rate

[Repeat until converges]

2-order Taylor expansion approximation

- Taylor series expansion ($\Delta \boldsymbol{w} = \boldsymbol{w} - \boldsymbol{w}'$)

$$E(\boldsymbol{w}' + \Delta \boldsymbol{w}) = E(\boldsymbol{w}') + \nabla E(\boldsymbol{w}')^T \Delta \boldsymbol{w} + \frac{1}{2} \Delta \boldsymbol{w}^T \mathbf{H}_E(\boldsymbol{w}') \Delta \boldsymbol{w} + \text{H.O.T.}$$

- Hessian matrix $\mathbf{H}_E(\boldsymbol{w}')$

$$\mathbf{H}_E(\boldsymbol{w}') = \begin{bmatrix} \frac{\partial^2 E(\boldsymbol{w}')}{\partial w[1] \partial w[1]} & \cdots & \frac{\partial^2 E(\boldsymbol{w}')}{\partial w[1] \partial w[M]} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\boldsymbol{w}')}{\partial w[M] \partial w[1]} & \cdots & \frac{\partial^2 E(\boldsymbol{w}')}{\partial w[M] \partial w[M]} \end{bmatrix}$$

- Newton's method
  - For any kind of "convex" loss $E(\boldsymbol{w})$
  - Quadratic approximation of $E(\boldsymbol{w})$ at $\boldsymbol{w}'$: ignore H.O.T.
  - Minimize the quadratic approximation to get $\Delta \boldsymbol{w} = -\mathbf{H}_E{}^{-1}(\boldsymbol{w}') \nabla E(\boldsymbol{w}')$
  - Update $\boldsymbol{w}'$ by $\boldsymbol{w}' + \eta^{(t)} \Delta \boldsymbol{w}$
- Pros: faster convergence; Cons: $\mathbf{H}_E(\boldsymbol{w}')$ and $\mathbf{H}_E{}^{-1}(\boldsymbol{w}')$ are time consuming

The Newton's method

Initialize parameters: $\widehat{\boldsymbol{w}}^{(0)} = [w[1], w[2], \dots, w[M]]^T$

Error function: $E(\boldsymbol{w})$

Compute gradients and Hessian:

$$\nabla E\left(\widehat{\boldsymbol{w}}^{(t)}\right) = \begin{bmatrix} \frac{\partial E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[1]} \\ \vdots \\ \frac{\partial E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[M]} \end{bmatrix}, \quad \mathbf{H}_E\left(\widehat{\boldsymbol{w}}^{(t)}\right) = \begin{bmatrix} \frac{\partial^2 E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[1]\partial w[1]} & \cdots & \frac{\partial^2 E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[1]\partial w[M]} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[M]\partial w[1]} & \cdots & \frac{\partial^2 E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[M]\partial w[M]} \end{bmatrix}$$

Update parameters:

$$\widehat{\boldsymbol{w}}^{(t+1)} = \widehat{\boldsymbol{w}}^{(t)} - \eta^{(t)} \mathbf{H}_E{}^{-1}(\boldsymbol{w}')\nabla E(\boldsymbol{w}')$$

Learning rate

[Repeat until converges]

Summary

- Pros of GD:
  - No expensive matrix operations (i.e., inversion)
  - Non-RSS error functions (if gradient is well formed)
  - Easy to parallelize: $E(\boldsymbol{w}) = \sum_i \ell(h(\boldsymbol{x}_i; \boldsymbol{w}), y_i) \implies \nabla E(\boldsymbol{w}) = \sum_i \nabla \ell(h(\boldsymbol{x}_i; \boldsymbol{w}), y_i)$

- Cons of GD:
  - Can be misled by local minima and stuck at larger-error valleys
  - Can get stuck on a flat plain/flat region of the error function
  - Can overshoot back and forth (oscillate from side to side)

- Helpers
  - Use multiple random-restarts with random initial parameters
  - Methods exist for selecting appropriate (best) learning rate at each iteration

Summary

- Pros of Newton's method:
  - Non-RSS error functions (if gradient and Hessian are well formed)
  - If $\boldsymbol{w}'$ is well-chosen, it can converge faster, since 2nd-order Taylor has more information.

- Cons of Newton's method:
  - Hessian and its inverse are time consuming
  - Divergence often happens if the function is flat or almost flat with respect to some dimension. In that case the second derivatives are close to zero, and their inverse becomes very large, resulting in gigantic steps.

- Helpers
  - Compute only the diagonal Hessian: $\Delta\boldsymbol{w} = -\mathbf{H}_E{}^{-1}(\boldsymbol{w}')\nabla E(\boldsymbol{w}')$
  - GD and then Newton's method

Question

- How many steps do you need to optimize $E(w) = \sum_i (wx_i - y_i)^2$?
  - GD?
  - Newton's method?

“Binary Linear” classification

- Predicting a binary class label
  - Yes/No, 1/0, +1/-1

- Setup
  - $D_{tr} = \{(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \{-1, +1\})\}_{n=1}^{N}$

- Models (hypothesis class = hyperplanes):
  - $\hat{y} = h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + w[0]) = \text{sign}(\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}})$
  - $\widetilde{\boldsymbol{w}} = [w[0], w[1], \dots, w[D]]^T \in \mathbb{R}^{D+1}$; $\widetilde{\boldsymbol{x}} = [1, x[1], \dots, x[D]]^T \in \mathbb{R}^{D+1}$
  - $\boldsymbol{w}$ (or $\widetilde{\boldsymbol{w}}$): weight vector; b: bias
  - Assumption: Linear separation between input features of different classes

Linear classification

- Data visualization and task illustration
  - What if b = 0?

- Meaning:
  - If w[1] * year + w[2] * miles > -b, do …

- How to measure the loss?
  - 0-1 loss: $\ell(y, y') = 1[\mathrm{y} \neq y']$
  - 1[True] = 1, 1[False] = 0 is the indicator function

Linear classification: beyond 2-D input

Learning goal

- Ideally, minimizing the generalized classification error as much as possible
  - But we only have training data
  - So, let's try empirical risk minimization

- **0-1 classification error**
  - $E(\widetilde{\boldsymbol{w}}) = \sum_i \mathbf{1}[y_i \neq h(\boldsymbol{x}_i; \widetilde{\boldsymbol{w}})] = \sum_i \mathbf{1}\left[y_i \neq \operatorname{sign}\left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right)\right] = \sum_i \mathbf{1}\left[y_i \neq \operatorname{sign}\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)\right]$
  - $E(\widetilde{\boldsymbol{w}}) = \sum_i \mathbf{1}\left[y_i\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right) \leq 0\right]$
  - Is this easy to minimize?

- Others:
  - **Surrogate losses as "differentiable" upper bounds for optimization**

Visualization of the concatenation

$\boldsymbol{x}_n \in \mathbb{R}^1$

$\widetilde{\boldsymbol{x}}_n \in \mathbb{R}^2$

$\boldsymbol{x}_n \in \mathbb{R}^2$

$\widetilde{\boldsymbol{x}}_n \in \mathbb{R}^3$

[From Cornell CS 4780]

Terminology

- **Classification accuracy**
  - $\frac{1}{N}\sum_i \mathbf{1}[y_i = h(\boldsymbol{x}_i; \widetilde{\boldsymbol{w}})]$
  - If it is on training/test data, then called training/test accuracy.

- **Note:**
  - Later when we introduce other ML algorithms, we may not use 0-1 loss!
  - Nevertheless, when we calculate the accuracy, we use the formula above.

Perceptron Algorithm

- An algorithm to learn a binary classification by minimizing the 0-1 loss

$\text{sign}(\boldsymbol{w}^T \boldsymbol{x})$

Perceptron Algorithm

- Concept:

Perceptron Algorithm

- Let $\boldsymbol{x_n} \in \mathbb{R}^D$ and $\boldsymbol{w} \in \mathbb{R}^D$, $y \in \{-1,1\}$
  - Assume that a dummy variable 1 is included in $\boldsymbol{x_n}$, and b is included in $\boldsymbol{w}$

- Initialize weight vector $\boldsymbol{w} = \boldsymbol{0}$
- Loop for T iterations (or till converge; i.e., no misclassification)
  - Loop for all training examples $\boldsymbol{x_n}$ (random order!)
    - Predict $\hat{y}_n = \text{sign}(\boldsymbol{w}^T \boldsymbol{x_n})$
    - If $\hat{y}_n \neq y_n$
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y_n \boldsymbol{x_n})$

- Note: η is the learning rate (step size) and $\eta \geq 0$

Perceptron Algorithm

- Example:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y_n \boldsymbol{x_n})$$

Perceptron Algorithm

- Example:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y_n \boldsymbol{x}_n)$$

Perceptron Algorithm

- Why does it work?
  - If $\hat{y}_n = \text{sign}(\boldsymbol{w}^T \boldsymbol{x}_n) = -1$ but $y_n = +1$
  - $(\boldsymbol{w} + \eta(y_n \boldsymbol{x}_n))^T \boldsymbol{x}_n = \boldsymbol{w}^T \boldsymbol{x}_n + \eta y_n \boldsymbol{x}_n{}^T \boldsymbol{x}_n \geq \boldsymbol{w}^T \boldsymbol{x}_n$

  - If $\hat{y}_n = \text{sign}(\boldsymbol{w}^T \boldsymbol{x}_n) = +1$ but $y_n = -1$
  - $(\boldsymbol{w} + \eta(y_n \boldsymbol{x}_n))^T \boldsymbol{x}_n = \boldsymbol{w}^T \boldsymbol{x}_n + \eta y_n \boldsymbol{x}_n{}^T \boldsymbol{x}_n \leq \boldsymbol{w}^T \boldsymbol{x}_n$

Perceptron Algorithm: online/streaming version

- Let $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{w} \in \mathbb{R}^D$, $y \in \{-1,1\}$
  - Assume that a dummy variable 1 is included in $\boldsymbol{x_n}$, and b is included in $\boldsymbol{w}$

- Initialize weight vector $\boldsymbol{w} = \boldsymbol{0}$
- Online version: Get one training data $(\boldsymbol{x}, y)$
  - Predict $\hat{y} = \text{sign}(\boldsymbol{w}^T\boldsymbol{x})$
  - If $\hat{y} \neq y$
    - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y\boldsymbol{x})$

Perceptron Algorithm

- Convergence (in terms of the training error):
  - If the data is linear separable

- Not convergence:
  - If the data is not linear separable
  - Not good … any idea to resolve it?

- **The pocket algorithm:** keep the best solution so far "in the pocket"
  - "Best" in terms of the training error

Proof of convergence

- If the training set $D_{tr} = \{(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \{-1, +1\})\}_{n=1}^{N}$ is linearly separable
  - $R = \max_n \|\boldsymbol{x}_n\|$
  - $B = \min_n \|\boldsymbol{w}\|,\ s.t.\ \forall n, y_n \boldsymbol{w}^T \boldsymbol{x}_n \geq 1$
  - Stop after performing at most $(RB)^2$ updated

- Intuition:
  - The more separable the data are
  - The faster the convergence

Proof of convergence

- Proof sketch
  - Let $\boldsymbol{w}^*$ be a solution of $\min_n \|\boldsymbol{w}\|, \ s.t. \ \forall n, y_n \boldsymbol{w}^T \boldsymbol{x}_n \geq 1$ (perfect prediction)
  - Let $\boldsymbol{w}^{(t)}$ be the $\boldsymbol{w}$ after t updates, and $\boldsymbol{w}^{(0)} = \boldsymbol{0}$
  - Let $\eta = 1$
  - Show that $\frac{\boldsymbol{w}^{(t)^T} \boldsymbol{w}^*}{\|\boldsymbol{w}^{(t)}\|_2 \|\boldsymbol{w}^*\|_2} \geq \frac{\sqrt{t}}{RB}$
    - The left-hand side cannot go over 1
    - $t \leq (RB)^2$
  - Intuition:
    - every update moves $\boldsymbol{w}^{(t)}$ closer to $\boldsymbol{w}^*$
  - Details:
    - See UML: 9-9.1.2

Reading: Pocket algorithm

- **Training data:** $D_{tr} = \{ (\boldsymbol{x}_i \in \mathbb{R}^D,\ \mathrm{y}_i \in \{+1, -1\}) \}_{i=1}^{N}$

- **Model:** $\mathrm{sign}(\boldsymbol{w}\boldsymbol{x} + b) = \mathrm{sign}(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}})$

- **Goal:** Make Perceptron stable when the data is not linearly separable
- **How?**
  - After every epoch (every iteration through all the training data), you compare the current solution $\widetilde{\boldsymbol{w}}$ to $\widetilde{\boldsymbol{w}}^{\mathrm{best}}$ in terms of their accuracy on the training data
  - If $\widetilde{\boldsymbol{w}}$ is better than $\widetilde{\boldsymbol{w}}^{\mathrm{best}}$, update $\widetilde{\boldsymbol{w}}^{\mathrm{best}}$ by $\widetilde{\boldsymbol{w}}$. That is, $\widetilde{\boldsymbol{w}}^{\mathrm{best}}$ keeps the best solution in terms of the training accuracy so far.

Reading: Pocket algorithm

- Initialize $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{w}}^{\text{best}} = \mathbf{0}$
- For $t = 1{:}T$
  - Loop for all training examples $\widetilde{\boldsymbol{x}}_{\boldsymbol{n}}$ (random order!)
    - Predict $\hat{y}_n = \text{sign}\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_{\boldsymbol{n}}\right)$
    - If $\hat{y}_n \neq y_n$
      - Update: $\widetilde{\boldsymbol{w}} \leftarrow \widetilde{\boldsymbol{w}} + \eta(y_n \widetilde{\boldsymbol{x}}_{\boldsymbol{n}})$

This is an epoch: seeing all the training data once

  - Evaluate $\widetilde{\boldsymbol{w}}$ on the "training data" and calculate the training accuracy
    - If training accuracy by $\widetilde{\boldsymbol{w}}$ is "higher" than the training accuracy by $\widetilde{\boldsymbol{w}}^{\text{best}}$
    - $\widetilde{\boldsymbol{w}}^{\text{best}} \leftarrow \widetilde{\boldsymbol{w}}$
- Output $\widetilde{\boldsymbol{w}}^{\text{best}}$

Reading: Pocket algorithm

- Initialize $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{w}}^{\text{best}} = \boldsymbol{0}$
- For $t = 1{:}T$
  - Loop for all training examples $\widetilde{\boldsymbol{x}}_{\boldsymbol{n}}$ (random order!)
    - Predict $\hat{y}_n = \text{sign}\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_{\boldsymbol{n}}\right)$
    - If $\hat{y}_n \neq y_n$
      - Update: $\widetilde{\boldsymbol{w}} \leftarrow \widetilde{\boldsymbol{w}} + \eta(y_n \widetilde{\boldsymbol{x}}_{\boldsymbol{n}})$
  - Evaluate $\widetilde{\boldsymbol{w}}$ on the "training data" and calculate the training accuracy
    - If training accuracy by $\widetilde{\boldsymbol{w}}$ is "higher" than the training accuracy by $\widetilde{\boldsymbol{w}}^{\text{best}}$
    - $\widetilde{\boldsymbol{w}}^{\text{best}} \leftarrow \widetilde{\boldsymbol{w}}$
- Output $\widetilde{\boldsymbol{w}}^{\text{best}}$

Here you compare $\widetilde{\boldsymbol{w}}$ to $\widetilde{\boldsymbol{w}}^{\text{best}}$ and make an update on $\widetilde{\boldsymbol{w}}^{\text{best}}$ if needed

Reading: Pocket algorithm

- Initialize $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{w}}^{\text{best}} = \mathbf{0}$
- For $t = 1{:}T$
  - Loop for all training examples $\widetilde{\boldsymbol{x}}_{\boldsymbol{n}}$ (random order!)
    - Predict $\hat{y}_n = \text{sign}\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_{\boldsymbol{n}}\right)$
    - If $\hat{y}_n \neq y_n$
      - Update: $\widetilde{\boldsymbol{w}} \leftarrow \widetilde{\boldsymbol{w}} + \eta(y_n \widetilde{\boldsymbol{x}}_{\boldsymbol{n}})$
  - Evaluate $\widetilde{\boldsymbol{w}}$ on the "training data" and calculate the training accuracy
    - If training accuracy by $\widetilde{\boldsymbol{w}}$ is "higher" than the training accuracy by $\widetilde{\boldsymbol{w}}^{\text{best}}$
    - $\widetilde{\boldsymbol{w}}^{\text{best}} \leftarrow \widetilde{\boldsymbol{w}}$
- Output $\widetilde{\boldsymbol{w}}^{\text{best}}$

Note that, this step does not update $\widetilde{\boldsymbol{w}}$

Reading: Pocket algorithm

- Initialize $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{w}}^{\text{best}} = \boldsymbol{0}$
- For $t = 1{:}T$
  - Loop for all training examples $\widetilde{\boldsymbol{x}}_{\boldsymbol{n}}$ (random order!)
    - Predict $\hat{y}_n = \text{sign}\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_{\boldsymbol{n}}\right)$
    - If $\hat{y}_n \neq y_n$
      - Update: $\widetilde{\boldsymbol{w}} \leftarrow \widetilde{\boldsymbol{w}} + \eta(y_n \widetilde{\boldsymbol{x}}_{\boldsymbol{n}})$
  - Evaluate $\widetilde{\boldsymbol{w}}$ on the "training data" and calculate the training accuracy
    - If training accuracy by $\widetilde{\boldsymbol{w}}$ is "higher" than the training accuracy by $\widetilde{\boldsymbol{w}}^{\text{best}}$
    - $\widetilde{\boldsymbol{w}}^{\text{best}} \leftarrow \widetilde{\boldsymbol{w}}$
- Output $\widetilde{\boldsymbol{w}}^{\text{best}}$

Finally, you output $\widetilde{\boldsymbol{w}}^{\text{best}}$

Probability (very high-level)

- Let X be a random variable and x be a particular outcome value

- If the sample space $\mathcal{X}$ (where x can take values from) is discrete
  - $P(X = x)$ is the probability mass function: the probability of $X = x$
  - $\sum_{x \in \mathcal{X}} P(X = x) = 1, 1 \geq P(X = x) \geq 0$

- If the sample space $\mathcal{X}$ (where x can take values from) is continuous
  - $p(X = x)$ is the probability density function: $P(b \geq X \geq a) = \int_a^b p(X = x)\, dx$
  - $\int p(X = x)\, dx = 1$, $p(X = x) \geq 0$

- When there is no confusion, X will be ignored; P and p are interchangeable

Gaussian density

- $p(X = x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2\sigma^2}$

- $p(X = x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$

Probability (very high-level)

- Expectation/mean
  - $\mu = E_P[X] = \sum_{x \in \mathcal{X}} P(X = x) x$; $\quad E_P[X] = E_{x \sim P}[x]$
  - $\mu = E_p[X] = \int p(X = x)\, x\, dx$; $\quad E_p[X] = E_{x \sim p}[x]$

- Variance
  - $E_P[(X - \mu)^2] = \sum_{x \in \mathcal{X}} P(X = x)(x - \mu)^2$
  - $E_p[(X - \mu)^2] = \int p(X = x)\, (x - \mu)^2 dx$

Probability (very high-level)

- Joint probability (the starting point):
  - $P(X = x, Y = y)$; $\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} P(X = x, Y = y) = 1$
  - $p(X = x, Y = y)$; $\int p(X = x, Y = y)\, dxdy = 1$

- Marginal probability:
  - $P(X = x) = \sum_{y \in \mathcal{Y}} P(X = x, Y = y)$
  - $p(X = x) = \int p(X = x, Y = y)\, dy$

- Conditional probability:
  - $P(X = x | Y = y) = \frac{P(X=x,Y=y)}{P(Y=y)} = \frac{P(X=x,Y=y)}{\sum_{x \in \mathcal{X}} P(X=x,Y=y)}$
  - $p(X = x | Y = y) = \frac{p(X=x,Y=y)}{p(Y=y)} = \frac{p(X=x,Y=y)}{\int p(X=x,Y=y) dx}$

Probability (very high-level)

- Chain rule:
  - $P(X = x, Y = y) = P(X = x)P(Y = y|X = x) = P(Y = y)P(X = x|Y = y)$
  - $P(X_1 = x_1, \dots, X_N = x_N) = P(x_1)P(x_2|x_1) \dots P(x_N|x_1, \dots, x_{N-1}) = \prod_i P(x_i|x_1, \dots., x_{i-1})$

- Bayes' rule
  - $P(X = x|Y = y) = \frac{P(X=x,Y=y)}{P(Y=y)} = \frac{P(X=x)P(Y=y|X=x)}{P(Y=y)}$
  - $P(Y = y|X = x) = \frac{P(X=x,Y=y)}{P(X=x)} = \frac{P(Y=y)P(X=x|Y=y)}{P(X=x)}$
  - Usage: reverse the cause and effect

Probability (very high-level)

- Independence (not always)
  - $P(X = x, Y = y) = P(X = x)P(Y = y)$

- Conditional independence (not always)
  - $P(X = x, Y = y|Z = z) = P(X = x|Z = z)P(Y = y|Z = z)$
  - Imply $P(X = x|Z = z, Y = y) = P(X = x|Z = z)$
  - Imply $P(Y = y|Z = z, X = x) = P(Y = y|Z = z)$

- In machine learning, inferring if random variables are independent, from data, is very challenging
- Most of the time, we “assume” variables are (conditional) independent to simplify the model and reduce the number of parameters to learn

Some terminology

- **Probabilistic inference:**
  - Derive the (conditional or marginal) probability of one or more random variables <u>taking a specific value or set of values</u>

  - **Example:**
    - Suppose $P(X = x, Y = y)$ is known
    - Derive $P(X = x|Y = y)$, $P(X = x)$, $\underset{x}{\operatorname{argmax}}\, P(X = x|Y = y)$

- **Learning/estimation:**
  - How to know the parameters within $P(X = x, Y = y)$?
  - How to know what probabilistic models to use for $P(X = x, Y = y)$?

What are probability models?

- Mathematical representation of probabilities
  - Bernoulli distribution (2 discrete assignments)
  - Categorical distribution (multiple discrete assignments)
  - Gaussian distribution (continuous assignments)

The probability of win is 40%; loss is 60%

- Probability models "often" have parameters
  - To set by domain experts
  - To be estimated from data: MLE, MAP, and Bayesian estimates

CSE 5523:
Estimating Probability from Data

The Ohio State University

HW

- HW-1 is due next Tuesday at midnight (23:59 ET).
- HW-2 will be released next Tuesday by midnight, and due in two weeks (10/1).

Today – Part 1

Linear classifier

- Review
- Perceptron (continued)

“Binary Linear” classification

- Predicting a binary class label
  - Yes/No, 1/0, +1/-1

- Setup
  - $D_{tr} = \{(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \{-1, +1\})\}_{n=1}^{N}$

- Models (hypothesis class = hyperplanes):
  - $\hat{y} = h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + w[0]) = \text{sign}(\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}})$
  - $\widetilde{\boldsymbol{w}} = [w[0], w[1], \dots, w[D]]^T \in \mathbb{R}^{D+1}$; $\widetilde{\boldsymbol{x}} = [1, x[1], \dots, x[D]]^T \in \mathbb{R}^{D+1}$

Perceptron Algorithm

- An algorithm to learn a binary classification by minimizing the 0-1 loss

$\text{sign}(\boldsymbol{w}^T\boldsymbol{x})$

0-1 classification error

$$E(\widetilde{\boldsymbol{w}}) = \sum_i \mathbf{1}[y_i \neq h(\boldsymbol{x}_i; \widetilde{\boldsymbol{w}})]$$

$$= \sum_i \mathbf{1}\left[y_i \neq \text{sign}\left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right)\right]$$

$$= \sum_i \mathbf{1}\left[y_i \neq \text{sign}\left(\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}}_i\right)\right]$$

$$E(\widetilde{\boldsymbol{w}}) = \textstyle\sum_i \mathbf{1}\left[y_i\left(\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}}_i\right) \leq 0\right]$$

Perceptron Algorithm

- Concept:

Perceptron Algorithm

- Let $\boldsymbol{x_n} \in \mathbb{R}^D$ and $\boldsymbol{w} \in \mathbb{R}^D$, $y \in \{-1,1\}$
  - Assume that a dummy variable 1 is included in $\boldsymbol{x_n}$, and b is included in $\boldsymbol{w}$

- Initialize weight vector $\boldsymbol{w} = \boldsymbol{0}$
- Loop for T iterations (or till converge; i.e., no misclassification)
  - Loop for all training examples $\boldsymbol{x_n}$ (random order!)
    - Predict $\hat{y}_n = \text{sign}(\boldsymbol{w}^T \boldsymbol{x_n})$
    - If $\hat{y}_n \neq y_n$
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y_n \boldsymbol{x_n})$

- Note: η is the learning rate (step size) and $\eta \geq 0$

Perceptron Algorithm

- Example:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y_n \boldsymbol{x_n})$$

Perceptron Algorithm

- Why does it work?
  - If $\hat{y}_n = \text{sign}(\boldsymbol{w}^T \boldsymbol{x}_n) = -1$ but $y_n = +1$
  - $(\boldsymbol{w} + \eta(y_n \boldsymbol{x}_n))^T \boldsymbol{x}_n = \boldsymbol{w}^T \boldsymbol{x}_n + \eta y_n \boldsymbol{x}_n{}^T \boldsymbol{x}_n \geq \boldsymbol{w}^T \boldsymbol{x}_n$

  - If $\hat{y}_n = \text{sign}(\boldsymbol{w}^T \boldsymbol{x}_n) = +1$ but $y_n = -1$
  - $(\boldsymbol{w} + \eta(y_n \boldsymbol{x}_n))^T \boldsymbol{x}_n = \boldsymbol{w}^T \boldsymbol{x}_n + \eta y_n \boldsymbol{x}_n{}^T \boldsymbol{x}_n \leq \boldsymbol{w}^T \boldsymbol{x}_n$

Perceptron Algorithm: online/streaming version

- Let $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{w} \in \mathbb{R}^D$, $y \in \{-1,1\}$
  - Assume that a dummy variable 1 is included in $\boldsymbol{x_n}$, and b is included in $\boldsymbol{w}$

- Initialize weight vector $\boldsymbol{w} = \boldsymbol{0}$
- Online version: Get one training data $(\boldsymbol{x}, y)$
  - Predict $\hat{y} = \text{sign}(\boldsymbol{w}^T\boldsymbol{x})$
  - If $\hat{y} \neq y$
    - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y\boldsymbol{x})$

Perceptron Algorithm

- Convergence (in terms of the training error):
  - If the data is linear separable

- Not convergence:
  - If the data is not linear separable
  - Not good … any idea to resolve it?

- **The pocket algorithm:** keep the best solution so far "in the pocket"
  - "Best" in terms of the training error

Proof of convergence

- If the training set $D_{tr} = \{(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \{-1, +1\})\}_{n=1}^{N}$ is linearly separable
  - $R = \max_n \|\boldsymbol{x}_n\|$
  - $B = \min_n \|\boldsymbol{w}\|,\ s.t.\ \forall n, y_n \boldsymbol{w}^T \boldsymbol{x}_n \geq 1$
  - Stop after performing at most $(RB)^2$ updated

- Intuition:
  - The more separable the data are
  - The faster the convergence

Proof of convergence

- Proof sketch
  - Let $\boldsymbol{w}^*$ be a solution of $\min_n \|\boldsymbol{w}\|\,,\ s.t.\ \forall n, y_n \boldsymbol{w}^T \boldsymbol{x}_n \geq 1$ (perfect prediction)
  - Let $\boldsymbol{w}^{(t)}$ be the $\boldsymbol{w}$ after t updates, and $\boldsymbol{w}^{(0)} = \boldsymbol{0}$
  - Let $\eta = 1$
  - Show that $\frac{\boldsymbol{w}^{(t)^T} \boldsymbol{w}^*}{\|\boldsymbol{w}^{(t)}\|_2 \|\boldsymbol{w}^*\|_2} \geq \frac{\sqrt{t}}{RB}$
    - The left-hand side cannot go over 1
    - $t \leq (RB)^2$
  - Intuition:
    - every update moves $\boldsymbol{w}^{(t)}$ closer to $\boldsymbol{w}^*$
  - Details:
    - See UML: 9-9.1.2

Today – Part 2

Estimating Probability from Data

- Overview
- Case study: Bernoulli distribution
  - MLE
  - MAP
  - Bayesian estimate
- *Reading: categorical and Gaussian*
- Multi-dimensional data
  - *Reading: discrete variables*

Estimating Probability from Data

- Goal: Fit a <u>probability model</u> p (interchangeable with P here) to data
  - $p(X, Y)$
    - $p(Y)p(\mathrm{X}|Y)$: generative learning
    - $p(X)p(Y|X)$: discriminative learning
  - or just $p(X)$ without labels

- Why to do so?
  - We can then use the Bayes optimal classifier $\hat{y} = \underset{y}{\operatorname{argmax}}\, P(Y = y|\boldsymbol{x})$
  - We can then perform probabilistic inference
  - We can then sample/generate new data instances
  - We can then evaluate the probability of seeing a particular data instance

Estimating Probability from Data

- Data $\mathcal{D}$ (observation)
  - $D_{tr} = \{(\boldsymbol{x}_n,\ y_n)\}_{n=1}^{N}$
  - $D_{tr} = \{\boldsymbol{x}_n\}_{n=1}^{N}$

- Assumption:
  - IID (or i.i.d.): Independent and Identically Distributed
  - $p\left((\boldsymbol{x}_i,\ y_i), (\boldsymbol{x}_j,\ y_j)\right) = p_i\left((\boldsymbol{x}_i,\ y_i)\right) \times p_j\left((\boldsymbol{x}_j,\ y_j)\right) = p\left((\boldsymbol{x}_i,\ y_i)\right) \times p\left((\boldsymbol{x}_j,\ y_j)\right)$
  - p_i and p_j are the same probability distribution

- Fitting to $D_{tr} = \{(\boldsymbol{x}_n,\ y_n)\}_{n=1}^{N}$ or $D_{tr} = \{\boldsymbol{x}_n\}_{n=1}^{N}$ is conceptually similar

Estimating "Parameters of Probability" from Data

- **Maximum likelihood estimation (MLE)**
  - Let $p(\boldsymbol{x}; \boldsymbol{\theta})$ be the probability model (for one data instance) with parameters $\boldsymbol{\theta}$
    - For example, a Gaussian distribution's mean and variance
  - Want to find with which $\boldsymbol{\theta}$, the data D_{tr} are most likely to be observed
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{x}_1, \boldsymbol{x}_2 \ \ldots, \boldsymbol{x}_N; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{x}_1; \boldsymbol{\theta}) \times \cdots p(\boldsymbol{x}_N; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(\boldsymbol{x}_n; \boldsymbol{\theta})$

- **Maximum a posteriori estimation (MAP)**
  - Assume that $\boldsymbol{\theta}$ is a random variable with a distribution $p(\boldsymbol{\theta})$; $p(\boldsymbol{x}; \boldsymbol{\theta})$ becomes $p(\boldsymbol{x}|\boldsymbol{\theta})$
    - For example, a Gaussian's mean is around 0 and variance is around 1
  - Want to find $\boldsymbol{\theta}$ with a high likelihood of observing D_{tr} & a high probability $p(\boldsymbol{\theta})$
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) \times p(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N|\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) \times \prod_n p(\boldsymbol{x}_n|\boldsymbol{\theta})$

Today

Estimating Probability from Data

- Overview
- Case study: Bernoulli distribution
  - MLE
  - MAP
  - Bayesian estimate
- Reading: categorical and Gaussian
- Multi-dimensional data
  - Reading: discrete variables

Bernoulli distribution (for a binary random variable)

- Parameters: $0 \leq \lambda \leq 1$
  - P("head") $= \lambda$
  - P("tail") $= 1 - \lambda$

λ 1- λ

- Examples (coin flipping):
  - Fair coins: $\lambda = 0.5$
  - Unfair coins: $\lambda \neq 0.5$

- Usages (given λ):
  - You can tell that seeing a "head" is P("head") $= \lambda$
  - You can simulate the coin flipping and get a "head" with a probability λ

Bernoulli distribution: parameter estimation

- Don't know λ, but have "N" independent instances/samples
  - Ex: have 9 coin flipping samples
  - 3 heads, 6 tails

- What should λ (i.e., P("head")) be?

- Why?

Bernoulli distribution: parameter estimation

- Don't know λ, but have "N" independent instances/samples
  - Ex: have 9 coin flipping samples
  - 3 heads, 6 tails

- What should λ (i.e., P("head")) be?
  - Assume the first three samples are "head"
  - Assume the last six samples are "tail" … this removes a constant $\binom{9}{3}$
  - Joint probability: $P(X_1 = \text{head}, X_2 = \text{head}, X_3 = \text{head}, X_4 = \text{tail}, \dots, X_9 = \text{tail}) = \boxed{P(X_1 = \text{head})P(X_2 = \text{head})P(X_3 = \text{head})P(X_4 = \text{tail}) \dots P(X_9 = \text{tail}) =}$ independent $\lambda^3(1-\lambda)^6$
  - The "likelihood function" of λ to see these assignments/observations is $\lambda^3(1-\lambda)^6$

Likelihood function

- In statistics, the likelihood function (simply, the likelihood) measures the goodness of fit of a model to a set of data instances for a given value of the unknown parameter λ.

- It is formed from the joint probability distribution of the samples
- It is viewed and used as a function of the parameters only, thus treating the random variables X as fixed at the observed values.

- In other words, the random variables get their assignments from the observations, and the likelihood is the joint probability of seeing these observations, given a value of the unknown parameter λ.

Assumptions on the observations

- Every coin flipping is based on the same probability (same parameter λ)
- Each flip is independent
  - Doesn't affect the outcomes of other flips
- IID (or i.i.d.): Independent and Identically Distributed

Maximum likelihood estimation (MLE)

- Likelihood function: p(observations; parameters)
  - When IID, p(observations ; parameters) = $\prod_i$ p(observation$_i$; parameters)

- **MLE:** choose the parameters that maximize the probability of observed data!

- Recipe:
  - Compute the log-likelihood
  - Take derivatives with respect to the parameters
  - Equate to zero and solve
    - You should check if you find a minimum or a maximum, by checking the second-order derivative (i.e., Hessian)

Maximum likelihood estimation (MLE)

- What $0 \leq \lambda \leq 1$ will maximize $\lambda^3(1-\lambda)^6$?
  - Log: monotonically increasing
  - Turn a multiplication into sum

$$\log \lambda^3(1-\lambda)^6 = \log \lambda^3 + \log(1-\lambda)^6$$
$$= 3\log\lambda + 6\log(1-\lambda)$$

  - Derivative = 0:

$$\frac{d\log\lambda^3(1-\lambda)^6}{d\lambda} = 3\frac{d\log\lambda}{d\lambda} + 6\frac{d\log(1-\lambda)}{d\lambda} = 3\frac{1}{\lambda} - 6\frac{1}{1-\lambda} = \frac{3-9\lambda}{\lambda(1-\lambda)} = 0$$

  - Answer: $\lambda = \frac{3}{9} = \frac{\#\text{ of heads}}{\#\text{ of samples}}$

Maximum likelihood estimation (MLE)

- Let $X_i(\text{“head”}) = 1, \;\; X_i(\text{“tail”}) = 0$
  - $P(X_1 = x_1, \dots, X_N = x_N) = P(X_1 = x_1) \dots P(X_N = x_N) = \prod_{i=1}^{N} P(X_i = x_i)$
  - $\prod_{i=1}^{N} P(X_i = x_i) = \prod_{i=1}^{N} \lambda^{x_i}(1-\lambda)^{1-x_i}$
  - Ex. $x_i = 1, \lambda^{x_i}(1-\lambda)^{1-x_i} = \lambda^1(1-\lambda)^{1-1} = \lambda$
  - EX. $x_i = 0, \lambda^{x_i}(1-\lambda)^{1-x_i} = \lambda^0(1-\lambda)^{1-0} = 1-\lambda$

  - $\log \prod_{i=1}^{N} \lambda^{x_i}(1-\lambda)^{1-x_i} = \sum_{i=1}^{N} \log\{\lambda^{x_i}(1-\lambda)^{1-x_i}\} = \sum_{i=1}^{N}\{\log\lambda^{x_i} + \log(1-\lambda)^{1-x_i}\}$
    $= \sum_{i=1}^{N} x_i \log\lambda + \sum_{i=1}^{N}(1-x_i)\log(1-\lambda)$
    $= (\#x_i = 1)\log\lambda + (\#x_i \neq 1)\log(1-\lambda)$

  - Maximum likelihood estimation: $\underset{\lambda}{\operatorname{argmax}}\, P(X_1 = x_1, \dots, X_N = x_N; \lambda) = \underset{\lambda}{\operatorname{argmax}}\, P(\mathcal{D}; \lambda)$
  - $\lambda = \dfrac{\#\text{ of } 1}{\#\text{ of samples}}$

Quick summary of MLE

- MLE gives the explanation of the data you observed
- MLE finds the **TRUE** parameters if
  - You choose the right probability model (hypothesis class $\mathcal{H}$)
  - N is large enough (seen from the concentration theory, Hoeffding's inequality)

- When N is small, MLE can overfit the data
- When the model is wrong, whatever you estimate is likely meaningless …

How to prevent overfitting

- Overfitting:
  - You model fits your training data too well (even the noise in the data)
    - N is small
    - The hypothesis class is too complicated
  - You model does not work well for your test data

- Here, to resolve the issue when N is small
  - Adding some prior belief "before" you see the data
    - Ex: λ for P("head") =λ is close 0.5

Maximum a posteriori estimation (MAP)

- Let λ be a random variable as well, with a probability P(λ)
  - Ex. You have some prior knowledge about the coin!

- $\underset{\lambda}{\operatorname{argmax}}\, P(\lambda|\mathcal{D}) = \underset{\lambda}{\operatorname{argmax}} \frac{P(\mathcal{D}|\lambda)p(\lambda)}{P(\mathcal{D})} = \underset{\lambda}{\operatorname{argmax}}\, P(\mathcal{D}|\lambda)p(\lambda)$ Bayes' rule!
  - $p(\lambda)$: prior distribution over λ, before seeing $\mathcal{D}$
  - $P(\mathcal{D}|\lambda)$: likelihood of the data given λ
  - $P(\lambda|\mathcal{D})$: posterior distribution over λ, after seeing $\mathcal{D}$

- How to set P(λ), the prior probability of λ, depends on:
  - Domain knowledge; or learned (e.g., empirical Bayes)
  - "Specific" probability models (guess what it will be?)

Maximum a posteriori estimation (MAP)

- Let $\mathrm{P}(\lambda) = \dfrac{\lambda^{\alpha-1}\,(1-\lambda)^{\beta-1}}{Z(\alpha,\beta)}$
  - This is called the Beta distribution of λ
  - α and β are the parameters of the distribution
  - $\mathrm{Z}(\alpha,\beta)$ is a constant (given α and β) for normalization

- Why do we use the Beta distribution?
  - It models the probability of λ, where λ has a range = [0, 1]
  - Mathematically beautiful
    - $P(\lambda|\mathcal{D}) \propto P(\mathcal{D}|\lambda)p(\lambda) \propto \left(\lambda^{\#\text{ of head}}\,(1-\lambda)^{\#\text{ of tail}}\right)\left(\lambda^{\alpha-1}\,(1-\lambda)^{\beta-1}\right) \propto \lambda^{\#\text{ of head}+\alpha-1}\,(1-\lambda)^{\#\text{ of tail}+\beta-1}$
    - Together with Bernoulli (or precisely binomial) as the likelihood, the posterior is again Beta
    - Such a prior is called the conjugate prior (of a certain likelihood function)!

Maximum a posteriori estimation (MAP)

Beta(1,1) (λ)

λ

Beta(5,5) (λ)

λ

Beta(100,100) (λ)

λ

$$\mathrm{P}(\lambda) = \frac{\lambda^{\alpha-1}\,(1-\lambda)^{\beta-1}}{Z(\alpha,\beta)}$$

$$P(\lambda|\mathcal{D}) \propto P(\mathcal{D}|\lambda)p(\lambda)$$

Maximum a posteriori estimation (MAP)

- $p(\mathcal{D}|\lambda)p(\lambda) \propto \lambda^{\#\text{ of heads}} \times (1-\lambda)^{\#\text{ of tails}} \times \lambda^{\alpha-1} \times (1-\lambda)^{\beta-1}$

$$= \lambda^{\#\text{ of heads} + \alpha - 1} \times (1-\lambda)^{\#\text{ of tails} + \beta - 1}$$

- What is $\underset{\lambda}{\operatorname{argmax}}\, P(\lambda|\mathcal{D}) = \underset{\lambda}{\operatorname{argmax}} \log p(\mathcal{D}|\lambda) + \log p(\lambda)$?
  - $\lambda = \dfrac{\#\text{ of heads} + \alpha - 1}{\#\text{ of samples} + \alpha + \beta - 2}$

- $\alpha - 1, \beta - 1$ (or α, β) are called pseudo counts
  - When N is large, MAP is similar to MLE

Comparison: MLE, MAP, and Bayesian

- Given "NH heads and NT tails"
  - MLE: $\underset{\lambda}{\operatorname{argmax}}\, P(\mathcal{D}|\lambda) = \frac{NH}{NH+NT}$
  - MAP: $\underset{\lambda}{\operatorname{argmax}}\, P(\lambda|\mathcal{D}) = \underset{\lambda}{\operatorname{argmax}}\, P(\mathcal{D}|\lambda)P(\lambda) = \frac{NH+\alpha-1}{(NH+\alpha-1)+(NT+\beta-1)}$, if $P(\lambda)$ is a Beta Dist.
  - Bayesian estimates: just keep $P(\lambda|\mathcal{D})$, with λ still a random variable
    - This keeps more information about λ (not just the mode)
  - MLE and MAP are point estimates: choosing one λ; Bayesian: keep a distribution

- Usages after seeing $\mathcal{D}$: future prediction $X =$ "head"
  - $P(X = \text{"head"}; \lambda) = \lambda$ λ is estimated by MLE or MAP
  - $P(X = \text{"head"}|\mathcal{D}) = \int P(X = \text{"head"}|\lambda)\, P(\lambda|\mathcal{D})d\lambda = \frac{NH+\alpha}{(NH+\alpha)+(NT+\beta)}$; "Bayesian inference"

More about Bayesian

- $P(X = \text{“head”}|\mathcal{D}) = \int P(X = \text{“head”}, \lambda \mid D)\, d\lambda =$
- $\int P(X = \text{“head”}|\lambda, D)\, P(\lambda|\mathcal{D}) d\lambda = \int P(X = \text{“head”}|\lambda)\, P(\lambda|\mathcal{D}) d\lambda =$
- $\int \lambda\, P(\lambda|\mathcal{D}) d\lambda = E[\lambda|\mathcal{D}] = \frac{NH+\alpha}{(NH+\alpha)+(NT+\beta)}$

- In general (for other distributions), the integration is intractable, and we must approximate it by Monte Carlo methods, variational methods, etc.
  - $\int P(X{=}x|\lambda)\, P(\lambda|\mathcal{D}) d\lambda \approx \frac{1}{M} \sum_m P(X{=}x|\lambda_m)$, where $\lambda_m \sim P(\lambda|\mathcal{D})$

- $P(\lambda|\mathcal{D}) = \frac{P(\lambda,\mathcal{D})}{P(\mathcal{D})}$ is usually intractable, due to $P(\mathcal{D}) = \int P(\mathcal{D}|\lambda)\, P(\lambda) d\lambda$

Reading: Categorical Distribution

Categorical distribution

- Parameters: $0 \leq \lambda[c] \leq 1, \sum_{c=1}^{C} \lambda[c] = 1$
  - P("1") $= \lambda[1]$
  - P("6") $= \lambda[6] = 1 - \sum_{c=1}^{5} \lambda[c]$

- Examples (dice tossing):
  - Fair dice: $\lambda[c] = 1/6$
  - Unfair dice: $\exists c, \lambda[c] \neq 1/6$

- Usages (given $\lambda[c]$):
  - The probability to see a "1" is P("1")$= \lambda[1]$
  - You can toss a dice to get a "1" with a probability $\lambda[1]$

Categorical distribution: parameter estimation

- Don't know $\lambda[c]$, but have "N" independent instances/samples
  - Ex: have 100 samples of tossing a dice
  - 30 of them are "1", 25 are "2", 20 are "3", 15 are "4", 5 are "5", and 5 are "6"

- What should $\lambda[c]$ be for each c?

- Why?

Categorical distribution: parameter estimation

- Don't know $\lambda[c]$, but have "N" independent instances/samples
  - Ex: have 100 samples of tossing a dice
  - 30 of them are "1", 25 are "2", 20 are "3", 15 are "4", 5 are "5", and 5 are "6"

- What should $\lambda[c]$ be for each c?
  - Assume the first 30 samples are "1", and so on
  - $P(X_1 = \text{"1"}, \dots, X_{100} = \text{"6"}) = P(X_1 = \text{"1"}) \dots P(X_{100} = \text{"6"}) = \lambda[1]^{30}\lambda[2]^{25}\lambda[3]^{20}\lambda[4]^{15}\lambda[5]^{5}\lambda[6]^{5}$
  - The "likelihood" to see these assignments is $\lambda[1]^{30}\lambda[2]^{25}\lambda[3]^{20}\lambda[4]^{15}\lambda[5]^{5}\lambda[6]^{5}$

Maximum likelihood estimation (MLE)

- What $0 \leq \lambda[c] \leq 1$ will maximize the likelihood?
  - Log: monotonically increasing

$$\log \lambda[1]^{30} \lambda[2]^{25} \lambda[3]^{20} \lambda[4]^{15} \lambda[5]^{5} \lambda[6]^{5}$$
$$= 30 \log \lambda[1] + 25 \log \lambda[2] + \cdots + 5 \log \lambda[6]$$

  - Derivative = 0 (with a constraint $\sum_{c=1}^{C} \lambda[c] = 1$, by Lagrangian multipliers):

$$\frac{d\, 30 \log \lambda[1] + \cdots + 5 \log \lambda[6] + \gamma\left(\sum_{c=1}^{C} \lambda[c] - 1\right)}{d\lambda[c]} = (\#\text{ of } c)\frac{1}{\lambda[c]} + \gamma = 0$$

  - Answer: $\lambda[c] = \dfrac{\#\text{ of } c}{\#\text{ of samples}}$

Maximum likelihood estimation (MLE)

- Let X ("1") $= 1, X$ ("6") $= 6$
  - $P(X_1 = x_1, \dots, X_N = x_N) = P(X_1 = x_1) \dots P(X_N = x_N) = \prod_{i=1}^{N} P(X_i = x_i)$
  - $\prod_{i=1}^{N} P(X_i = x_i) = \prod_{i=1}^{N} \{\prod_{c=1}^{C} \lambda[c]^{\mathbf{1}[x_i=c]}\}$
  - Ex. $x_i = 1, \prod_{c=1}^{C} \lambda[c]^{\mathbf{1}[x_i=c]} = \lambda[1]$
  - EX. $x_i = 6, \prod_{c=1}^{C} \lambda[c]^{\mathbf{1}[x_i=c]} = \lambda[6]$

Indicator function

Maximum likelihood estimation (MLE)

- Let X ("1") $= 1, X$ ("6") $= 6$
  - $P(X_1 = x_1, \dots, X_N = x_N) = P(X_1 = x_1) \dots P(X_N = x_N) = \prod_{i=1}^{N} P(X_i = x_i)$
  - $\prod_{i=1}^{N} P(X_i = x_i) = \prod_{i=1}^{N} \{\prod_{c=1}^{C} \lambda[c]^{\mathbf{1}[x_i=c]}\}$
  - Ex. $x_i = 1, \prod_{c=1}^{C} \lambda[c]^{\mathbf{1}[x_i=c]} = \lambda[1]$
  - EX. $x_i = 6, \prod_{c=1}^{C} \lambda[c]^{\mathbf{1}[x_i=c]} = \lambda[6]$

  - $\log \prod_{i=1}^{N} \prod_{c=1}^{C} \lambda[c]^{1[x_i=c]} = \sum_{i=1}^{N} \sum_{c=1}^{C} \mathbf{1}[x_i = c] \log \lambda[c]$
    $= \sum_{c=1}^{C} \log \lambda[c] \sum_{i=1}^{N} \mathbf{1}[x_i = c] = \sum_{c=1}^{C} \log \lambda[c] \, (\#x_i = c)$

  - Maximum likelihood estimation: $\underset{\lambda}{\operatorname{argmax}} P(X_1 = x_1, \dots, X_N = x_N | \lambda) = \underset{\lambda}{\operatorname{argmax}} P(\mathcal{D} | \lambda)$
  - $\lambda[c] = \dfrac{\#\text{ of } c}{\#\text{ of samples}}$

Maximum a posteriori estimation (MAP)

- Let $\boldsymbol{\lambda}$ be a random vector as well, with a probability P($\boldsymbol{\lambda}$)
  - Ex. You have some prior knowledge about the dice!

- $\underset{\boldsymbol{\lambda}}{\operatorname{argmax}}\, P(\boldsymbol{\lambda}|\mathcal{D}) = \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \frac{P(\mathcal{D}|\boldsymbol{\lambda})p(\boldsymbol{\lambda})}{P(\mathcal{D})} = \underset{\boldsymbol{\lambda}}{\operatorname{argmax}}\, P(\mathcal{D}|\boldsymbol{\lambda})p(\boldsymbol{\lambda})$ Bayes' rule!

- How to set P($\boldsymbol{\lambda}$), the prior probability of $\boldsymbol{\lambda}$, depends on:
  - Domain knowledge
  - "Specific" probability models (guess what it will be?)

Summary: categorical distribution

- Parameters: $0 \leq \lambda[c] \leq 1, \sum_{c=1}^{C} \lambda[c] = 1$
  - P("1") $= \lambda[1]$
  - P("6") $= \lambda[6] = 1 - \sum_{c=1}^{5} \lambda[c]$

- MLE: $\lambda[c] = \dfrac{\#\text{ of } c}{\#\text{ of samples}}$
  - Including a Lagrangian multipliers $\gamma(\sum_{c=1}^{C} \lambda[c] - 1)$ in optimization

- MAP:
  - Dirichlet distribution: $p(\boldsymbol{\lambda}) = \dfrac{\prod_c \lambda[c]^{\alpha[c]-1}}{Z(\boldsymbol{\alpha})}$
  - $\lambda[c] = \dfrac{\#\text{ of } c + (\alpha[c]-1)}{\#\text{ of samples} + \sum_{c=1}^{C} \alpha[c]-1}$

Reading: Gaussian

Gaussian distribution: continuous variables

- Let X be a random variable for coin weight, what is P($X = x$)?

- Gaussian is a popular and widely applicable distribution for continuous R.V.

$$P(X = x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$

  - Properties: uni-modal

- Parameters:
  - Mean μ
  - Variance σ^2

Gaussian distribution: continuous variables

- Don't know μ and σ, but have "N" independent instances/samples
  - Ex: have 6 coins
  - Weight: 4.0, 5.2, 5.3, 5.5, 6.0, 4.8
  - Gaussian is suitable for some "uni-modal" data distributions, but not for multi-modal

- What should μ and σ be?
  - $P(X_1 = 4.0, X_2 = 5.2, \dots, X_6 = 4.8) = P(X_1 = 4.0)P(X_2 = 5.2) \dots P(X_6 = 4.8) =$

$$\prod_{i=1}^{N} P(X_i = x_i) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x_i-\mu)^2/2\sigma^2}$$

Maximum likelihood estimation (MLE)

- What μ and σ will maximize the likelihood?
  - Log: monotonically increasing

$$\log \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x_i-\mu)^2/2\sigma^2} = \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x_i-\mu)^2/2\sigma^2}\right)$$

$$= \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(e^{-(x_i-\mu)^2/2\sigma^2}\right)$$

$$= \sum_{i=1}^{N} -\log\left(\sqrt{2\pi\sigma^2}\right) - (x_i - \mu)^2/2\sigma^2$$

Maximum likelihood estimation (MLE)

- What μ and σ will maximize the likelihood?
  - Derivative = 0 (for μ):

$$\frac{d\left[\sum_{i=1}^{N} -\log\left(\sqrt{2\pi\sigma^2}\right) - (x_i - \mu)^2/2\sigma^2\right]}{d\mu}$$

$$= \frac{d\left[\sum_{i=1}^{N} -(x_i - \mu)^2/2\sigma^2\right]}{d\mu}$$

$$= \frac{\sum_{i=1}^{N} -2(x_i - \mu)(-1)}{2\sigma^2} = \frac{\sum_{i=1}^{N}(x_i - \mu)}{\sigma^2} = \frac{\left(\sum_{i=1}^{N} x_i\right) - (N\mu)}{\sigma^2} = 0$$

$$\sum_{i=1}^{N} x_i = N\mu$$

  - Answer: $\mu = \frac{\sum_{i=1}^{N} x_i}{N} = mean$

Maximum likelihood estimation (MLE)

- What μ and σ will maximize the likelihood?
  - Derivative = 0 (for σ):

$$\frac{d\left[\sum_{i=1}^{N} -\log\left(\sqrt{2\pi\sigma^2}\right) - (x_i - \mu)^2/2\sigma^2\right]}{d\sigma}$$

$$= -\frac{d\left[\sum_{i=1}^{N} \log\left(\sqrt{2\pi\sigma^2}\right)\right]}{d\sigma} - \frac{d\left[\sum_{i=1}^{N} \frac{(x_i - \mu)^2}{2\sigma}^2\right]}{d\sigma} = \frac{dN\log\left(\sqrt{2\pi}\sigma\right)}{d\sigma} - \frac{d\left[\sum_{i=1}^{N}(x_i - \mu)^2/2\sigma^2\right]}{d\sigma}$$

$$= -\frac{N \times \sqrt{2\pi}}{\sqrt{2\pi}\sigma} - \sum_{i=1}^{N} -\frac{(x_i - \mu)^2}{\sigma^3} = -\frac{N\sigma^2}{\sigma^3} + \sum_{i=1}^{N} \frac{(x_i - \mu)^2}{\sigma^3} = 0$$

  - Answer: $\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N} = variance$

Reading: Multi-dimensional discrete

For multi-dimensional data: discrete

- Example:
  - Every time, sample one coin
  - Flip the picked coin

0 1

Instance ID: *i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Coin	0	0	1	0	0	1	0	1
Head (1)/Tail (0)	1	0	1	1	0	1	1	0

$\boldsymbol{x}_i$
$x_i[1]$
$x_i[2]$

- What parameters?
  - How many "joint outcomes"?

For multi-dimensional data: discrete

- Example:
  - Every time, sample one coin
  - Flip the picked coin

0 1

Instance ID: *i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Coin	0	0	1	0	0	1	0	1
Head (1)/Tail (0)	1	0	1	1	0	1	1	0

$\boldsymbol{x}_i$
$x_i[1]$
$x_i[2]$

- What parameters?
  - How many "joint outcomes"?
  - Need a parameter for each joint outcome (categorical): $\lambda[0, 0]$, $\lambda[0, 1]$, $\lambda[1, 0]$, $\lambda[1, 1]$
    - P("coin = 0", "tail"), P("coin = 0", "head"), P("coin = 1", "tail"), P("coin = 1", "head")
    - **MLE:** counts and normalize: P("coin = 0", "tail")= $\frac{\#\text{ of ("coin = 0", "tail")}}{\#\text{ of samples}}$

Exponentially growing!

For multi-dimensional data: discrete

- Example:
  - Every time, sample one coin
  - Flip the picked coin

0 1

Instance ID: *i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Coin	0	0	1	0	0	1	0	1
Head (1)/Tail (0)	1	0	1	1	0	1	1	0

$\boldsymbol{x}_i$
$x_i[1]$
$x_i[2]$

- MLE practice
  - P("coin = 0")?
  - P("tail" | "coin = 0")?
    - $\frac{P(\text{“tail”}, \text{“coin = 0”})}{P(\text{“coin = 0”})} = \frac{P(\text{“tail”}, \text{“coin = 0”})}{P(\text{“tail”}, \text{“coin = 0”}) + P(\text{“head”}, \text{“coin = 0”})} = \frac{\#\text{ of (“coin = 0”, “tail”)}}{\#\text{ “coin = 0”}}$
    - Look at columns of "coin = 0", among which how many of them are tails?

CSE 5523:
Generative vs. Discriminative Learning

The Ohio State University

HW

- HW-1 is due today at midnight (11:59 PM ET).
- HW-2 will be released today by midnight, and due in two weeks (10/1).

Today

Review

Multi-dimensional Gaussian

Generative vs. discriminative learning

Linear regression revisit

Regularization, cross-validation

Estimating Probability from Data

- Goal: Fit a <u>probability model</u> p (interchangeable with P here) to data
  - $p(X, Y)$
    - $p(Y)p(\mathrm{X}|Y)$: generative learning
    - $p(X)p(Y|X)$: discriminative learning
  - or just $p(X)$ without labels

- Data $\mathcal{D}$ (observation/assignments): $D_{tr} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$ or $D_{tr} = \{\boldsymbol{x}_n\}_{n=1}^{N}$
  - Treatments are conceptually the same

- Assumption: IID (or i.i.d.): Independent and Identically Distributed

Estimating "Parameters of Probability" from Data

- **Maximum likelihood estimation (MLE)**
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\mathcal{D}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{x}_1, \boldsymbol{x}_2 \ \ldots, \boldsymbol{x}_N; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{x}_1; \boldsymbol{\theta}) \times \cdots p(\boldsymbol{x}_N; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(\boldsymbol{x}_n; \boldsymbol{\theta})$

- **Maximum a posteriori estimation (MAP)**
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, P(\boldsymbol{\theta}|\mathcal{D}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) \times p(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N|\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) \times \prod_n p(\boldsymbol{x}_n|\boldsymbol{\theta})$

- **Bernoulli distribution**

λ 1- λ

$$\prod_{i=1}^{N} P(x_n) = \prod_{i=1}^{N} \lambda^{x_n}(1-\lambda)^{1-x_n}$$

Maximum a posteriori estimation (MAP): Bernoulli

Beta(1,1) (λ)

λ

Beta(5,5) (λ)

λ

Beta(100,100) (λ)

λ

$$\mathrm{P}(\lambda) = \frac{\lambda^{\alpha-1}\,(1-\lambda)^{\beta-1}}{Z(\alpha,\beta)}$$

$$P(\lambda|\mathcal{D}) \propto P(\mathcal{D}|\lambda)p(\lambda)$$

Maximum a posteriori estimation (MAP)

- $p(\mathcal{D}|\lambda)p(\lambda) \propto \lambda^{\#\text{ of heads}} \times (1-\lambda)^{\#\text{ of tails}} \times \lambda^{\alpha-1} \times (1-\lambda)^{\beta-1}$

$$= \lambda^{\#\text{ of heads} + \alpha - 1} \times (1-\lambda)^{\#\text{ of tails} + \beta - 1}$$

- What is $\underset{\lambda}{\operatorname{argmax}}\, P(\lambda|\mathcal{D}) = \underset{\lambda}{\operatorname{argmax}} \log p(\mathcal{D}|\lambda) + \log p(\lambda)$?
  - $\lambda = \dfrac{\#\text{ of heads} + \alpha - 1}{\#\text{ of samples} + \alpha + \beta - 2}$

- $\alpha - 1, \beta - 1$ (or α, β) are called pseudo counts
  - When N is large, MAP is similar to MLE

Comparison: MLE, MAP, and Bayesian

- Given "NH heads and NT tails"
  - MLE: $\underset{\lambda}{\operatorname{argmax}}\, P(\mathcal{D}|\lambda) = \frac{NH}{NH+NT}$
  - MAP: $\underset{\lambda}{\operatorname{argmax}}\, P(\lambda|\mathcal{D}) = \underset{\lambda}{\operatorname{argmax}}\, P(\mathcal{D}|\lambda)P(\lambda) = \frac{NH+\alpha-1}{(NH+\alpha-1)+(NT+\beta-1)}$, if $P(\lambda)$ is a Beta Dist.
  - Bayesian estimates: just keep $P(\lambda|\mathcal{D})$, with λ still a random variable
    - This keeps more information about λ (not just the mode)
  - MLE and MAP are point estimates: choosing one λ; Bayesian: keep a distribution

- Usages after seeing $\mathcal{D}$: future prediction $X =$ "head"
  - $P(X = \text{"head"}; \lambda) = \lambda$ λ is estimated by MLE or MAP
  - $P(X = \text{"head"}|\mathcal{D}) = \int P(X = \text{"head"}|\lambda)\, P(\lambda|\mathcal{D})d\lambda = \frac{NH+\alpha}{(NH+\alpha)+(NT+\beta)}$; **"Bayesian inference"**

More about Bayesian

- $P(X = \text{“head”}|\mathcal{D}) = \int P(X = \text{“head”}, \lambda \mid D)\, d\lambda =$
- $\int P(X = \text{“head”}|\lambda, D)\, P(\lambda|\mathcal{D}) d\lambda = \int P(X = \text{“head”}|\lambda)\, P(\lambda|\mathcal{D}) d\lambda =$
- $\int \lambda\, P(\lambda|\mathcal{D}) d\lambda = E[\lambda|\mathcal{D}] = \frac{NH+\alpha}{(NH+\alpha)+(NT+\beta)}$

- In general (for other distributions), the integration is intractable, and we must approximate it by Monte Carlo methods, variational methods, etc.
  - $\int P(X{=}x|\lambda)\, P(\lambda|\mathcal{D}) d\lambda \approx \frac{1}{M} \sum_m P(X{=}x|\lambda_m)$, where $\lambda_m \sim P(\lambda|\mathcal{D})$

- $P(\lambda|\mathcal{D}) = \frac{P(\lambda,\mathcal{D})}{P(\mathcal{D})}$ is usually intractable, due to $P(\mathcal{D}) = \int P(\mathcal{D}|\lambda)\, P(\lambda) d\lambda$

For multi-dimensional data: continuous

- Observe $\{\boldsymbol{x}_i \in \mathbb{R}^D\}_{i=1}^N$
- Fit a multi-dimensional Gaussian (by MLE):

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{D/2}\ \det(\boldsymbol{\Sigma})^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

$$\underset{\boldsymbol{\mu},\boldsymbol{\Sigma}}{\operatorname{argmax}}\, p(\boldsymbol{x}_1, \dots\dots, \boldsymbol{x}_N; \quad \boldsymbol{\mu} \in \mathbb{R}^D, \boldsymbol{\Sigma} \in \mathbb{R}^{D\times D})$$

- $\boldsymbol{\mu} = \frac{\sum_{i=1}^N \boldsymbol{x}_i}{N}$
- $\boldsymbol{\Sigma}$ = covariance matrix $= \frac{1}{N}\sum_{i=1}^N (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T$

Background: covariance matrix

$$\boldsymbol{\Sigma} = \begin{bmatrix} \mathrm{cov}_{1,1} & \dots & \mathrm{cov}_{1,D} \\ \vdots & \ddots & \vdots \\ \mathrm{cov}_{D,1} & \dots & \mathrm{cov}_{D,D} \end{bmatrix} = \begin{bmatrix} E[(x[1]-\mu[1])(x[1]-\mu[1])] & \dots & E[(x[1]-\mu[1])(x[D]-\mu[D])] \\ \vdots & \ddots & \vdots \\ E[(x[D]-\mu[D])(x[1]-\mu[1])] & \dots & E[(x[D]-\mu[D])(x[D]-\mu[D])] \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{N}\sum_{n=1}^{N}[(x_n[1]-\mu[1])(x_n[1]-\mu[1])] & \dots & \frac{1}{N}\sum_{n=1}^{N}[(x_n[1]-\mu[1])(x_n[D]-\mu[D])] \\ \vdots & \ddots & \vdots \\ \frac{1}{N}\sum_{n=1}^{N}[(x_n[D]-\mu[D])(x_n[1]-\mu[1])] & \dots & \frac{1}{N}\sum_{n=1}^{N}[(x_n[D]-\mu[D])(x_n[D]-\mu[D])] \end{bmatrix}$$

$$= \frac{1}{N}\sum_{n=1}^{N} \begin{bmatrix} (x_n[1]-\mu[1])(x_n[1]-\mu[1]) & \dots & (x_n[1]-\mu[1])(x_n[D]-\mu[D]) \\ \vdots & \ddots & \vdots \\ (x_n[D]-\mu[D])(x_n[1]-\mu[1]) & \dots & (x_n[D]-\mu[D])(x_n[D]-\mu[D]) \end{bmatrix}$$

$$= \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n-\boldsymbol{\mu})(\boldsymbol{x}_n-\boldsymbol{\mu})^T$$

$$= \frac{1}{N}\overline{\boldsymbol{X}}\overline{\boldsymbol{X}}^{\boldsymbol{T}}, \text{ where } \overline{\boldsymbol{X}} = [\boldsymbol{x}_1-\boldsymbol{\mu}, \quad \boldsymbol{x}_2-\boldsymbol{\mu}, \quad \boldsymbol{x}_3-\boldsymbol{\mu}, \quad \dots, \quad \boldsymbol{x}_N-\boldsymbol{\mu}]$$

One-dimensional vs. multi-dimensional

- One-dimensional: $p(X = x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$

- Multi-dimensional: $p(\boldsymbol{x}) = \frac{1}{(2\pi)^{D/2} \det(\boldsymbol{\Sigma})^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$

- When $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$, $p(\boldsymbol{x}) = \frac{1}{(2\pi)^{D/2}\sigma^D} e^{-(\boldsymbol{x}-\boldsymbol{\mu})^T(\boldsymbol{x}-\boldsymbol{\mu})/2\sigma^2} = \frac{1}{(2\pi)^{D/2}\sigma^D} e^{-\|\boldsymbol{x}-\boldsymbol{\mu}\|_2^2/2\sigma^2}$

For multi-dimensional data: continuous

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{D/2}\ \det(\boldsymbol{\Sigma})^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

- Properties:
  - Faraway from the mean, lower probability

Ex: $\Sigma = \begin{bmatrix} 1.5 & 1 \\ 1 & 1.1 \end{bmatrix}$

$\boldsymbol{\Sigma}^{-1} \approx \begin{bmatrix} 1.7 & -1.5 \\ -1.5 & 2.3 \end{bmatrix}$

For multi-dimensional data: continuous

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{D/2}\ \det(\boldsymbol{\Sigma})^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

- Properties:
  - Faraway from the mean, lower probability

- Think
  - What is $\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$?

Ex: $\Sigma = \begin{bmatrix} 1.5 & 1 \\ 1 & 1.1 \end{bmatrix}$

$\boldsymbol{\Sigma}^{-1} \approx \begin{bmatrix} 1.7 & -1.5 \\ -1.5 & 2.3 \end{bmatrix}$

eigenvalues, eigenvectors of $\Sigma = 2.32 \begin{bmatrix} 0.77 \\ 0.63 \end{bmatrix}, 0.28 \begin{bmatrix} 0.63 \\ -0.77 \end{bmatrix}$

For multi-dimensional data: continuous

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{D/2}\ \det(\boldsymbol{\Sigma})^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

- Properties:
  - Faraway from the mean, lower probability

- Think
  - What is $\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$?

Ex: $\Sigma = \begin{bmatrix}1.5 & 1\\1 & 1.1\end{bmatrix}$

$\boldsymbol{\Sigma}^{-1} \approx \begin{bmatrix}1.7 & -1.5\\-1.5 & 2.3\end{bmatrix}$

eigenvalues, eigenvectors of $\Sigma = 2.32\begin{bmatrix}0.77\\0.63\end{bmatrix}, 0.28\begin{bmatrix}0.63\\-0.77\end{bmatrix}$

Generative vs. Discriminative learning

- Training data: $D_{tr} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$

- Goal: fit $p(X, Y)$ to D_{tr}
  - $p(Y)p(\mathrm{X}|Y)$: generative learning (with $p(\mathrm{X}|Y)$ you can generate/sample new data)
  - $p(X)p(Y|X)$: discriminative learning
    - Usually not estimating $p(X)$

- Prediction (in the MLE or MAP scenario, mostly we do)
  - Given a test example $\boldsymbol{x}$
  - $\hat{y} = \max_{c} p(\mathrm{Y} = \mathrm{c}|\boldsymbol{x})$; the Bayes Optimal Classifier
    - In this sense, discriminative learning seems to be more straightforward (or specialized)

Generative vs. Discriminative learning

- Generative learning (MLE case)
  - $p(\boldsymbol{x}, y; \boldsymbol{\theta}) = p(\boldsymbol{x}|y; \boldsymbol{\theta})p(y; \boldsymbol{\theta})$
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(\boldsymbol{x}_n, y_n\,; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(\boldsymbol{x}_n|y_n\,; \boldsymbol{\theta})p(y_n; \boldsymbol{\theta})$
  - Need to make two modeling assumptions (conditional data distribution, prior)
  - Modeling $p(\boldsymbol{x}_n|y_n\,; \boldsymbol{\theta})$ is challenging
  - If the assumption is correct, then fewer training data are needed

Generative vs. Discriminative learning

- Generative learning (MLE case)
  - $p(\boldsymbol{x}, y; \boldsymbol{\theta}) = p(\boldsymbol{x}|y; \boldsymbol{\theta})p(y; \boldsymbol{\theta})$
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(\boldsymbol{x}_n, y_n\,; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(\boldsymbol{x}_n|y_n\,; \boldsymbol{\theta})p(y_n; \boldsymbol{\theta})$
  - Need to make two modeling assumptions (conditional data distribution, prior)
  - Modeling $p(\boldsymbol{x}_n|y_n\,; \boldsymbol{\theta})$ is challenging
  - If the assumption is correct, then fewer training data are needed

- Discriminative learning (MLE case)
  - $p(\boldsymbol{x}, y; \boldsymbol{\theta}) = p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})p(\boldsymbol{x})$
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n|\boldsymbol{x}_n; \boldsymbol{\theta})$
  - Need to make one modeling assumptions (conditional class distribution)

Don’t get confused

- Discriminative vs. generative learning:
  - Difference in modeling
- MAP vs. MLE:
  - Difference in the objective function

[Rombach et al., CVPR 2022]

Linear regression revisit (bias is absorbed)

- Given $D_{tr} = \{(\boldsymbol{x}_n, y_n \in \mathbb{R})\}_{n=1}^{N}$
- Goal: Fit a probabilistic model that can be used to predict label given $\boldsymbol{x}$

- Let us consider $p(X, Y) = p(X)p(Y|X; \boldsymbol{\theta})$ or $p(X, Y) = p(X)p(Y|X, \boldsymbol{\theta})$
  - We assume that $p(X)$ is unknown but fixed (so no need to learn it)
  - We will only estimate the parameters $\widehat{\boldsymbol{\theta}}$ for $p(Y|X; \boldsymbol{\theta})$
  - $\underset{y}{\operatorname{argmax}}\, p\left(Y = y \middle| X = \boldsymbol{x}; \widehat{\boldsymbol{\theta}}\right)$ for prediction, for example

- $p(D_{tr}; \boldsymbol{\theta}) = p\left(\{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}; \boldsymbol{\theta}\right) = \prod_n p(\boldsymbol{x}_n) \prod_n p(y_n|\boldsymbol{x}_n; \boldsymbol{\theta})$
  - That is, $p(D_{tr}; \boldsymbol{\theta}) \propto \prod_n p(y_n|\boldsymbol{x}_n; \boldsymbol{\theta})$

Linear regression revisit (bias is absorbed)

- Given $D_{tr} = \{(\boldsymbol{x}_n, y_n \in \mathbb{R})\}_{n=1}^{N}$
- Goal: We want to fit $p(Y|X; \boldsymbol{\theta})$

- What probabilistic model should we use?

- We use: $p(y|\boldsymbol{x}; \boldsymbol{w}) \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-\boldsymbol{w}^T\boldsymbol{x})^2/2\sigma^2}$
  - $Y = y$ is a normally distributed around $\boldsymbol{w}^T\boldsymbol{x}$

Linear regression revisit (bias is absorbed)

- Assume that $y = \boldsymbol{w}^T\boldsymbol{x} + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
  - $\mathcal{N}(\mu, \sigma^2)$ has a PDF $p(\varepsilon) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(\varepsilon - 0)^2/2\sigma^2}$
  - y is drawn from a line $\boldsymbol{w}^T\boldsymbol{x}$ plus some Gaussian noise

- $y|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{w}^T\boldsymbol{x}, \sigma^2)$
- $p(y|\boldsymbol{x}; \boldsymbol{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y - \boldsymbol{w}^T\boldsymbol{x})^2/2\sigma^2}$

Linear regression revisit (bias is absorbed)

- Setup
  - $D_{tr} = \{(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \mathbb{R})\}_{n=1}^{N}$
  - A fixed unknown $p(\boldsymbol{X} = \boldsymbol{x})$

- $p(D_{tr}; \boldsymbol{w}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n, y_n; \boldsymbol{w}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n) p(y_n | \boldsymbol{x}_n; \boldsymbol{w}) \propto \prod_{n=1}^{N} p(y_n | \boldsymbol{x}_n; \boldsymbol{w})$

- $\log \prod_{n=1}^{N} p(y_n | \boldsymbol{x}_n; \boldsymbol{w}) = \sum_n \log p(y_n | \boldsymbol{x}_n; \boldsymbol{w}) = \sum_n \log \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2 / 2\sigma^2} = \sum_n \log \frac{1}{\sqrt{2\pi\sigma^2}} + \log e^{-(y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2 / 2\sigma^2}$

Linear regression revisit (bias is absorbed)

- $\sum_n \log \frac{1}{\sqrt{2\pi\sigma^2}} + \log e^{-(y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2/2\sigma^2} \propto \sum_n -(y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2/2\sigma^2 \propto \sum_n -\left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2$

- MLE: $\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \sum_n -\left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2 = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_n \left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2$

Ridge regression revisit

- Assume that $p(\boldsymbol{w}) = \frac{1}{(2\pi)^{D/2}\ \det(\tau^2 \boldsymbol{I})^{1/2}} e^{-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{0})^T(\tau^2 \boldsymbol{I})^{-1}(\boldsymbol{w}-\boldsymbol{0})}$

- $p(\boldsymbol{w}) = \frac{1}{(2\pi)^{D/2}\ \det(\tau^2 \boldsymbol{I})^{1/2}} e^{-\frac{1}{2}\frac{\|\boldsymbol{w}\|_2^2}{\tau^2}}$

- $\log p(D_{tr}|\boldsymbol{w})p(\boldsymbol{w}) \propto \sum_n -\left(y_n - \boldsymbol{w}^T\boldsymbol{x}_n\right)^2 + \log p(\boldsymbol{w})$

- $\underset{\boldsymbol{w}}{\operatorname{argmax}} \sum_n -\left(y_n - \boldsymbol{w}^T\boldsymbol{x}_n\right)^2 - \frac{1}{2}\frac{\|\boldsymbol{w}\|_2^2}{\tau^2} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_n \left(y_n - \boldsymbol{w}^T\boldsymbol{x}_n\right)^2 + \frac{1}{2}\frac{\|\boldsymbol{w}\|_2^2}{\tau^2}$

- Ridge regression includes an L2 regularizer (not dependent on data)

MLE, MAP, and Bayesian for supervised learning

- MLE: $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta})$, and predict by $p\left(y \middle| \boldsymbol{x}; \widehat{\boldsymbol{\theta}}\right)$
- MAP: $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}|D_{tr}) \propto p(D_{tr}|\boldsymbol{\theta})p(\boldsymbol{\theta})$, and predict by $p\left(y \middle| \boldsymbol{x}; \widehat{\boldsymbol{\theta}}\right)$
- Difference between MAP and MLE is subtle
  - $\log p(\boldsymbol{\theta})$ can be seen as a regularizer to penalize largely deviating $\boldsymbol{\theta}$

- Bayesian: find $p(\boldsymbol{\theta}|D_{tr})$ and predict by $p(y|\boldsymbol{x}, D_{tr}) = \int p(y|\boldsymbol{x}, \boldsymbol{\theta})\, P(\boldsymbol{\theta}|D_{tr}) d\boldsymbol{\theta}$

* Bayesian (linear/nonlinear) regression *

- $p(y|\boldsymbol{x}, D_{tr}) = \int p(y|\boldsymbol{x}, \boldsymbol{w})\, p(\boldsymbol{w}|D_{tr})\, d\lambda$
- Let $p(y|\boldsymbol{x}, \boldsymbol{w}) = \mathcal{N}(w[0] + w[1]\boldsymbol{x}, \sigma^2)$, $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$

[Figure by Bishop, PRML, 2006]

* Bayesian (linear/nonlinear) regression *

$p(\boldsymbol{w}|D_{tr})$

$p(y|\boldsymbol{x},\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}^T\boldsymbol{x},\sigma^2)$

$$p(y|\boldsymbol{x},D_{tr}) = \int p(y|\boldsymbol{x},\boldsymbol{w})\,p(\boldsymbol{w}|D_{tr})d\lambda$$

Sample-dependent prediction variance

* Bayesian (linear/nonlinear) regression *

With feature transform

$$p(y|\boldsymbol{x}, D_{tr}) = \int p(y|\boldsymbol{x}, \boldsymbol{w})\, p(\boldsymbol{w}|D_{tr}) d\lambda$$

Sample-dependent prediction variance

Regularization and hyper-parameters

- Ridge regression:
  - $\underset{\boldsymbol{w}}{\operatorname{argmin}} \quad \sum_n \left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2 + \lambda \|\boldsymbol{w}\|_2^2$

Regularization and hyper-parameters

- Ridge regression:
  - $\underset{\boldsymbol{w}}{\operatorname{argmin}} \boxed{\boxed{\sum_n \left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2} + \boxed{\lambda \|\boldsymbol{w}\|_2^2}}$
  - The object function contains a training loss term and a regularization (e.g., belief) term

Regularization and hyper-parameters

- Ridge regression:
  - $\underset{\boldsymbol{w}}{\operatorname{argmin}} \boxed{\boxed{\sum_n \left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2} + \boxed{\lambda \|\boldsymbol{w}\|_2^2}}$
  - The object function contains a training loss term and a regularization (e.g., belief) term
  - Model parameters (e.g., $\boldsymbol{w}$) is learned from minimizing the training objective

Regularization and hyper-parameters

- Ridge regression:
  - $\underset{\boldsymbol{w}}{\operatorname{argmin}} \boxed{\boxed{\sum_n \left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2} + \boxed{\lambda \|\boldsymbol{w}\|_2^2}}$
  - The object function contains a training loss term and a regularization (e.g., belief) term
  - Model parameters (e.g., $\boldsymbol{w}$) is learned from minimizing the training objective
  - How about hyper-parameters (e.g., λ) of the algorithm or hypothesis class?
    - Can you determine λ by minimizing the training objective?

Regularization and hyper-parameters

- Ridge regression:
  - $\underset{\boldsymbol{w}}{\operatorname{argmin}} \boxed{\boxed{\sum_n \left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2} + \boxed{\lambda \|\boldsymbol{w}\|_2^2}}$
  - The object function contains a training loss term and a regularization (e.g., belief) term
  - Model parameters (e.g., $\boldsymbol{w}$) is learned from minimizing the training objective
  - How about hyper-parameters (e.g., λ) of the algorithm or hypothesis class?
    - Can you determine λ by minimizing the training objective?

- Caution: Don't be confused by notations; λ is not always used as parameters.

Machine learning

- A good learning algorithm + hypothesis class
  - Contain a hypothesis that can achieve low training loss and test (validation) loss
  - The algorithm can return such a hypothesis
  - We may view ridge regression as <u>regularized loss (risk) minimization + linear regressors</u>

- In practice, we choose the learning algorithm + hypothesis class by validation

(Cross) validation

- Goal: choose the hyper-parameters (ID of an algorithm + hypothesis class)
  - Ex:# of neighbors and distance in KNN, λ in ridge regression, η in iterative optimization

Algo 1 Algo 2 …… Algo M

(Cross) validation

- Goal: choose the hyper-parameters
  - Ex:# of neighbors and distance in KNN, λ in ridge regression, η in iterative optimization

  - Base + validation

(Cross) validation

- Goal: choose the hyper-parameters
  - Ex:# of neighbors and distance in KNN, λ in ridge regression, η in iterative optimization

For each algorithm

  - Base + validation

1. Find $\hat{\boldsymbol{w}}$ in the base set
2. Evaluate $\hat{\boldsymbol{w}}$ in the validation set

(Cross) validation

- Goal: choose the hyper-parameters
  - Ex:# of neighbors and distance in KNN, λ in ridge regression, η in iterative optimization

Algo 1 | Algo 2 | …… | Algo M

For each algorithm

- Base + validation

1. Find $\hat{\boldsymbol{w}}$ in the base set
2. Evaluate $\hat{\boldsymbol{w}}$ in the validation set

↓

Choose the algorithm with

1. lowest validation error/loss, or
2. highest validation accuracy

*** Regularization term is not involved**

(Cross) validation

- Goal: choose the hyper-parameters
  - Ex:# of neighbors and distance in KNN, λ in ridge regression, η in iterative optimization

  - Cross validation

(Cross) validation

- Goal: choose the hyper-parameters
  - Ex:# of neighbors and distance in KNN, λ in ridge regression, η in iterative optimization

For each algorithm

  - Cross validation

For each k

1. Find $\hat{\boldsymbol{w}}$ from D_{tr} excluding fold k
2. Evaluate $\hat{\boldsymbol{w}}$ in the fold k

(Cross) validation

- Goal: choose the hyper-parameters
  - Ex:# of neighbors and distance in KNN, λ in ridge regression, η in iterative optimization

Algo 1 | Algo 2 | …… | Algo M

For each algorithm

- Cross validation

For each k

1. Find $\hat{\boldsymbol{w}}$ from D_{tr} excluding fold k
2. Evaluate $\hat{\boldsymbol{w}}$ in the fold k

↓

Choose the algorithm with average

1. lowest validation error/loss, or
2. highest validation accuracy

CSE 5523:
Generative Learning

The Ohio State University

HW

- HW-2 was released this by morning and is due on 10/1.

Today

Review

Regularization, cross-validation

Gaussian discriminative analysis

Naïve Bayes

- Modeling
- Learning (informal)
- Learning (formal)

Estimating Probability from Data

- Data $\mathcal{D}$ (observation/assignments): $D_{tr} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$
  - Assumption: IID (or i.i.d.): Independent and Identically Distributed

- Goal: Fit a <u>probability model</u> p (interchangeable with P here) to data
  - $p(X, Y)$
    - $p(Y)p(\mathrm{X}|Y)$: generative learning
    - $p(X)p(Y|X)$: discriminative learning

- Prediction
  - Given a test example $\boldsymbol{x}$
  - $\hat{y} = \max_{c} p(\mathrm{Y} = \mathrm{c}|\boldsymbol{x})$; the Bayes Optimal Classifier

Estimating "Parameters of Probability" from Data

- **Maximum likelihood estimation (MLE)**
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\mathcal{D}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N); \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N); \boldsymbol{\theta}) \times \cdots p\big((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N); \boldsymbol{\theta}\big) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p\big((\boldsymbol{x}_n, y_n); \boldsymbol{\theta}\big)$

- **Maximum a posteriori estimation (MAP)**
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, P(\boldsymbol{\theta}|\mathcal{D}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) \times p((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)|\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) \times \prod_n p\big((\boldsymbol{x}_n, y_n)|\boldsymbol{\theta}\big)$

Linear regression revisit (bias is absorbed)

- Modeling: $y|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{w}^T\boldsymbol{x}, \sigma^2)$
  - $p(y|\boldsymbol{x};\boldsymbol{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-\boldsymbol{w}^T\boldsymbol{x})^2/2\sigma^2}$

- MLE:
  - Given $D_{tr} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$
  - $\prod_{n=1}^N p(y_n|\boldsymbol{x}_n;\boldsymbol{w})$
  - $\log \prod_{n=1}^N p(y_n|\boldsymbol{x}_n;\boldsymbol{w})$
  - $\sum_n -\left(y_n - \boldsymbol{w}^T\boldsymbol{x}_n\right)^2$
  - $\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \sum_n -\left(y_n - \boldsymbol{w}^T\boldsymbol{x}_n\right)^2 = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_n \left(y_n - \boldsymbol{w}^T\boldsymbol{x}_n\right)^2$

Ridge regression revisit

- MAP:
  - $p(\boldsymbol{w}) = \frac{1}{(2\pi)^{D/2}\ \det(\tau^2 \boldsymbol{I})^{1/2}} e^{-\frac{1}{2}\frac{\|\boldsymbol{w}\|_2^2}{\tau^2}}$
  - $\log p(D_{tr}|\boldsymbol{w})p(\boldsymbol{w}) \propto \sum_n -\left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2 + \log p(\boldsymbol{w})$
  - $\underset{\boldsymbol{w}}{\operatorname{argmax}} \sum_n -\left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2 - \frac{1}{2}\frac{\|\boldsymbol{w}\|_2^2}{\tau^2} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left\{\sum_n \left(y_n - \boldsymbol{w}^T \boldsymbol{x}_n\right)^2\right\} + \frac{1}{2}\frac{\|\boldsymbol{w}\|_2^2}{\tau^2}$

- Ridge regression includes an L2 regularizer (not dependent on data)

Gaussian discriminative analysis (GDA)

- Training data: $D_{tr} = \{(\boldsymbol{x}_n, y_n \in \{1, 2, \dots, C\})\}_{n=1}^{N}$

- To fit $p(X, Y) = p(Y)p(\mathrm{X}|Y)$: so generative learning indeed

- Assumption:
  - $p(Y; \boldsymbol{\theta})$ is Bernoulli or categorical
  - $p(X|Y = y; \boldsymbol{\theta})$ is multivariate Gaussian
    - Sometimes, assuming shared covariance

Gaussian discriminative analysis (GDA)

- Training data: $D_{tr} = \{(\boldsymbol{x}_n, y_n \in \{1, 2, \dots, C\})\}_{n=1}^{N}$

- To fit $p(X, Y) = p(Y)p(\mathrm{X}|Y)$: so generative learning indeed

- Assumption:
  - $p(Y; \boldsymbol{\theta})$ is Bernoulli or categorical
  - $p(X|Y = y; \boldsymbol{\theta})$ is multivariate Gaussian
    - Sometimes, assuming shared covariance

- Prediction: $\underset{y}{\operatorname{argmax}} \log p\left(y|\boldsymbol{x}; \widehat{\boldsymbol{\theta}}\right) = \underset{y}{\operatorname{argmax}} \log p\left(y; \widehat{\boldsymbol{\theta}}\right) + \log p\left(\boldsymbol{x}|y; \widehat{\boldsymbol{\theta}}\right)$

Given $\boldsymbol{x}$, which y is the most likely? Which y is the most likely? Which y most likely generates $\boldsymbol{x}$?

Gaussian discriminative analysis

$$p(X,Y) = p(Y)p(\mathrm{X}|Y)$$

Gaussian discriminative analysis

- Intuition:
  - $p(Y = 1; \widehat{\boldsymbol{\theta}}) =$
  - $p(Y = 0; \widehat{\boldsymbol{\theta}}) =$
  - $p(\boldsymbol{x} | Y = 1; \widehat{\boldsymbol{\theta}}) =$
  - $p(\boldsymbol{x} | Y = 0; \widehat{\boldsymbol{\theta}}) =$
  - $\boldsymbol{\theta}$ or $\widehat{\boldsymbol{\theta}}$ is the collection of all model parameters

$$p(X, Y) = p(Y)p(\mathrm{X}|Y)$$

Gaussian discriminative analysis

- Intuition:
  - $p(Y = 1; \boldsymbol{\theta}) = \lambda = \frac{N_1}{N_1 + N_0}$
  - $p(Y = 0; \boldsymbol{\theta}) = \frac{N_0}{N_1 + N_0}$
  - $p(\boldsymbol{x} | Y = 1; \boldsymbol{\theta})$ by $\boldsymbol{\mu}_1$ and $\boldsymbol{\Sigma}_1$ from all orange points
  - $p(\boldsymbol{x} | Y = 0; \boldsymbol{\theta})$ by $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ from all blue points

$$p(X, Y) = p(Y)p(\mathrm{X}|Y)$$

Gaussian discriminative analysis

- Intuition:
  - $p(Y = 1; \boldsymbol{\theta}) = \lambda = \frac{N_1}{N_1 + N_0}$
  - $p(Y = 0; \boldsymbol{\theta}) = \frac{N_0}{N_1 + N_0}$
  - $p(\boldsymbol{x} | Y = 1; \boldsymbol{\theta})$ by $\boldsymbol{\mu}_1$ and $\boldsymbol{\Sigma}_1$ from all orange points
  - $p(\boldsymbol{x} | Y = 0; \boldsymbol{\theta})$ by $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ from all blue points

- Shared $\boldsymbol{\Sigma}$ (i.e., $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_0$):

$$\boldsymbol{\Sigma} = \frac{N_1 \boldsymbol{\Sigma}_1 + N_0 \boldsymbol{\Sigma}_0}{N_1 + N_0}$$

Gaussian discriminative analysis

- Shared $\boldsymbol{\Sigma}$ (i.e., $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_0$):

$$\boldsymbol{\Sigma} = \frac{N_1\boldsymbol{\Sigma}_1 + N_0\boldsymbol{\Sigma}_0}{N_1 + N_0}$$

- Prediction
  - $\underset{y}{\operatorname{argmax}} \log p(y; \hat{\boldsymbol{\theta}}) + \log p(\boldsymbol{x}|y; \hat{\boldsymbol{\theta}})$
  - Can be rewritten as $\operatorname{sign}(\hat{\boldsymbol{w}}^T\boldsymbol{x} + \hat{b})$
    - Output 1 means class 1
    - Output -1 means class 0

Gaussian discriminative analysis

- MLE on the joint probability

$$\log \prod_{i=1}^{N} p(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i; \boldsymbol{\theta}) p(y_i; \boldsymbol{\theta})$$

$$= \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i; \boldsymbol{\theta}) + \sum_{i=1}^{N} \log p(y_i; \boldsymbol{\theta})$$

Gaussian for $y_i = 1$ or $y_i = 0$

Bernoulli, do not share parameters with Gaussian

- Each term is affected by some "separate" parameters
  - We can then estimate each parameter based on a certain set of data

Gaussian discriminative analysis

- MLE on the joint probability

$$\log \prod_{i=1}^{N} p(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i; \boldsymbol{\theta}) p(y_i; \boldsymbol{\theta})$$

$$= \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i; \boldsymbol{\theta}) + \sum_{i=1}^{N} \log p(y_i; \boldsymbol{\theta})$$

Gaussian for $y_i = 1$ or $y_i = 0$

Bernoulli, do not share parameters with Gaussian

- $\underset{\lambda}{\operatorname{argmax}} \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i) + \sum_{i=1}^{N} \log p(y_i) = \underset{\lambda}{\operatorname{argmax}} \sum_{i=1}^{N} \log p(y_i)$
  - Where did you see this?

Gaussian discriminative analysis

- MLE on the joint probability

$$\log \prod_{i=1}^{N} p(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i; \boldsymbol{\theta}) p(y_i; \boldsymbol{\theta})$$

$$= \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i; \boldsymbol{\theta}) + \sum_{i=1}^{N} \log p(y_i; \boldsymbol{\theta})$$

Gaussian for $y_i = 1$ or $y_i = 0$

Bernoulli, do not share parameters with Gaussian

- $\underset{\boldsymbol{\Sigma}_1}{\operatorname{argmax}} \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i) + \sum_{i=1}^{N} \log p(y_i) = \underset{\boldsymbol{\Sigma}_1}{\operatorname{argmax}} \sum_{i \in \{y_i = 1\}} \log p(\boldsymbol{x}_i | y_i)$

Is this model powerful?

- Not really if the data (feature vectors) per class is not Gaussian
- Usually powerful when paired with deep learning features

- Efficient model updates:
  - Let $\boldsymbol{\theta} = \{\lambda, \boldsymbol{\mu}_1, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2\}$
  - Let $\widehat{\boldsymbol{\theta}}^{(N)}$ be the parameters estimated from $D_{tr} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$
  - If now there is a new training example $(\boldsymbol{x}_{N+1}, y_{N+1})$ coming in
  - To estimate $\widehat{\boldsymbol{\theta}}^{(N+\mathbf{1})}$ from $D_{tr} + \{(\boldsymbol{x}_{N+1}, y_{N+1})\}$, we don't need to keep D_{tr}
  - Instead, we only need to keep $\widehat{\boldsymbol{\theta}}^{(N)}$ and some other information like N_1, N_0
  - $\boldsymbol{\theta}$ + $\{N_1, N_0\}$ can be viewed as the <u>sufficient statistics</u> of D_{tr} for GDA
  - That is, knowing $\boldsymbol{\theta}$ + $\{N_1, N_0\}$ of D_{tr} is sufficient for building up the GDA models

Naïve Bayes

- **Training data:** $D_{tr} = \left\{ \left(\boldsymbol{x}_i \in \mathbb{R}^D,\ \mathrm{y}_i \in \{1, 2, \dots, C\} \right) \right\}_{i=1}^{N}$
  - Each dimension of $\boldsymbol{x}_i$ can be binary, categorical, count, or numerical (real)

- **Goal:** construct $p(Y = c|\boldsymbol{x})$ for $\hat{y} = \max_c p(Y = c|\boldsymbol{x})$

Naïve Bayes

- **Training data:** $D_{tr} = \{ (\boldsymbol{x}_i \in \mathbb{R}^D,\ \mathrm{y}_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$
  - Each dimension of $\boldsymbol{x}_i$ can be binary, categorical, count, or numerical (real)

- **Goal:** construct $p(Y = c|\boldsymbol{x})$ for $\hat{y} = \max_{c} p(Y = c|\boldsymbol{x})$

- **How? Bayes' rules**
  - $p(Y = c|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|Y=c)p(Y=c)}{P(\boldsymbol{x})} \propto p(\boldsymbol{x}|Y = c)p(Y = c)$
  - $\operatorname{argmax}_{\mathrm{c}} p(Y = c|\boldsymbol{x}) = \operatorname{argmax}_{\mathrm{c}} p(\boldsymbol{x}|Y = c)p(Y = c)$

- **Key:** what probability models for $p(\boldsymbol{x}|Y = c; \boldsymbol{\theta})$ and $p(Y = c; \boldsymbol{\theta})$?

Naïve Bayes: modeling

- $p(Y = c; \boldsymbol{\theta})$: Bernoulli or categorical

- $p(\boldsymbol{x}|Y = c; \boldsymbol{\theta})$: **conditional independence assumption:**
  - $p(\boldsymbol{x}|Y = c) = p(x[1], x[2], x[3], \dots, x[D] \mid Y = c)$

$$= p(x[1] \mid Y = c)p(x[2] \mid Y = c) \dots \dots p(x[D] \mid Y = c) = \prod_{d=1}^{D} p(x[d] \mid Y = c)$$

Naïve Bayes: modeling

- $p(Y = c; \boldsymbol{\theta})$: Bernoulli or categorical

- $p(\boldsymbol{x}|Y = c; \boldsymbol{\theta})$: **conditional independence assumption:**
  - $p(\boldsymbol{x}|Y = c) = p(x[1], x[2], x[3], \dots, x[D] \mid Y = c)$

    $= p(x[1] \mid Y = c)p(x[2] \mid Y = c) \dots \dots p(x[D] \mid Y = c) = \prod_{d=1}^{D} p(x[d] \mid Y = c)$
  - Given the class label, the probability of observing the value of one dimension is conditionally independent from the values of the other dimensions
  - Ex: Spam filtering
    - $\boldsymbol{x}$: an email; $x[d]$: the *d*-th word or the *d*-th unique word; $c \in \{\text{spam}, \text{not spam}\}$
    - Clearly, this assumption is usually not true, but the model works well in practice.
    - <u>"All models are wrong, but some are useful."</u>

Naïve Bayes: data generation and inference

- Example:
  - Every time, sample a coin (from two "types")
  - Flip the sampled coin and measure the weight

0 1

Instance ID: *i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Coin	0	0	1	0	0	1	0	1
Head (1)/Tail (0)	1	1	1	1	0	0	1	0
Weight	2.2	2.3	2.5	1.7	1.9	2.1	1.5	2.7

y_i
$x_i[1]$
$x_i[2]$

  - Given the type of coin, observing head/tail is <u>conditionally independent</u> from the weight

- **Classification problem:** Now given a new observation $\boldsymbol{x}$ (e.g., "head" and "weight = 2.4"), what type of coin (0 or 1) is it from?

Naïve Bayes: data generation and inference

- Example:
  - Every time, sample a coin (from two "types")
  - Flip the sampled coin and measure the weight

0 1

Instance ID: *i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Coin	0	0	1	0	0	1	0	1
Head (1)/Tail (0)	1	1	1	1	0	0	1	0
Weight	2.2	2.3	2.5	1.7	1.9	2.1	1.5	2.7

y_i
$x_i[1]$
$x_i[2]$

  - Given the type of coin, observing head/tail is <u>conditionally independent</u> from the weight

- $p(Y = 1) \times p(x[1] = \text{head}|Y = 1) \times p(x[2] = 2.4|Y = 1)$ vs.
- $p(Y = 0) \times p(x[1] = \text{head}|Y = 0) \times p(x[2] = 2.4|Y = 0)$

Naïve Bayes: another example

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Maintained?
Car brand?
Mileage?

$$
\begin{aligned}
&(\boldsymbol{x}|Y = c) \\
&= p(x[1] \mid Y = c) \\
&\times p(x[2] \mid Y = c) \\
&\times p(x[3] \mid Y = c)
\end{aligned}
$$

Naïve Bayes: another example

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Maintained?
Car brand?
Mileage?

Bernoulli distribution: $p(x[d] = 1 | Y = c) = \lambda_{d,c}$

$$\lambda_{d,c} = \frac{\sum_{i=1}^{N} \mathbf{1}[y_i = c]\mathbf{1}[x_i[d] = 1]}{\sum_{i=1}^{N} \mathbf{1}[y_i = c]}$$

Each pair of (dimension, class) has a specific coin

The generative process:

1. Sample one class c
2. Flip the corresponding coin with $\lambda_{d,c}$

Naïve Bayes: another example

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Maintained?
Car brand?
Mileage?

category m

Categorical distribution: $p(x[d] = m | Y = c) = \lambda_{d,c}[m]$

$$\lambda_{d,c}[m] = \frac{\sum_{i=1}^{N} \mathbf{1}[y_i = c]\mathbf{1}[x_i[d] = m]}{\sum_{i=1}^{N} \mathbf{1}[y_i = c]}$$

Each pair of (dimension, class) has a specific dice

The generative process:

1. Sample one class c
2. Throw the corresponding dice with $\lambda_{d,c}$

Naïve Bayes: another example

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Maintained?
Car brand?
Mileage?

Gaussian distribution: $p(x[d]|Y = c) = \mathcal{N}\left(\mu_{d,c}, \sigma_{d,c}^2\right)$

$$\mu_{d,c} = \frac{1}{N_c} \sum_{i=1}^{N} \mathbf{1}[y_i = c] x[d]$$

$$\sigma_{d,c}^2 = \frac{1}{N_c} \sum_{i=1}^{N} \mathbf{1}[y_i = c] \left(x[d] - \mu_{d,c}\right)^2$$

Each pair of (dimension, class) has a specific Gaussian

The generative process:

1. Sample one class c
2. Sample from the corresponding Gaussian with $\mu_{d,c}$ and $\sigma_{d,c}^2$

How to learn $p(x[d] \mid Y = c)$ and $p(Y = c)$?

- Given training data
  - $\mathcal{D} = \{ (\boldsymbol{x}_i,\ \mathrm{y}_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

- Applying the MLE criterion
  - $\underset{\text{parameters}}{\operatorname{argmax}}\ p(\mathcal{D}; \text{parameters})$

- $p(\mathcal{D}; \text{parameters}) = p(\{\boldsymbol{x}_1, y_1\}, \dots, \{\boldsymbol{x}_N, y_N\}) = \prod_{i=1}^{N} p(\boldsymbol{x}_i, y_i)$
  - IID observations

Naïve Bayes: learning

- MLE:
  - Maximize the "joint probability" of the **IID** data instances

$$\log p(\{\boldsymbol{x}_1, y_1\}, \dots) = \log \prod_{i=1}^{N} p(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i) p(y_i)$$

Naïve Bayes: learning

- MLE:
  - Maximize the "joint probability" of the **IID** data instances
  - Separately optimize each probability model: each has independent parameters and can have different models!

$$\sum_{d=1}^{D}\sum_{i=1}^{N}\log p(\boldsymbol{x}_i[d] \mid y_i) + \sum_{i=1}^{N}\log p(y_i)$$

$$= \boxed{\sum_{i=1}^{N}\log p(\boldsymbol{x}_i[1] \mid y_i)} + \cdots \ldots + \boxed{\sum_{i=1}^{N}\log p(\boldsymbol{x}_i[D] \mid y_i)} + \boxed{\sum_{i=1}^{N}\log p(y_i)}$$

Categorical, Bernoulli, or 1-D Gaussian for each class *c*

Categorical or Bernoulli

Naïve Bayes: learning

- MLE: for each conditional term $\sum_{i=1}^{N} \log p(\boldsymbol{x}_i[d] \mid y_i)$, we must build a conditional probability model for each class *c*
  - $p(\boldsymbol{x}[d] \mid Y = 1), p(\boldsymbol{x}[d] \mid Y = 2), \ldots \ldots, p(\boldsymbol{x}[d] \mid Y = C)$

- How?
  - Look at data instances of class *c*
  - Pick one feature dimension *d*
  - Perform MLE over $\boldsymbol{x}_i[d]$ whose $y_i = c$ (i.e., $\{(\boldsymbol{x}_i, y_i = c)\}$) to build $p(x[d] \mid Y = c)$
    - If $x[d]$ is continuous (e.g., weight), use a Gaussian
    - If $x[d]$ is binary, use Bernoulli; if $x[d]$ is categorical, use categorical
    - <u>Of course, other distributions are applicable!</u>

Naïve Bayes: learning

- Example:

Instance ID: *i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Coin	0	0	1	0	0	1	0	1
Head (1)/Tail (0)	1	1	1	1	0	0	1	0
Weight	2.2	2.3	2.5	1.7	1.9	2.1	1.5	2.7

y_i
$x_i[1]$
$x_i[2]$

- What to learn?

Naïve Bayes: learning

- Example:

Instance ID: *i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Coin	0	0	1	0	0	1	0	1
Head (1)/Tail (0)	1	1	1	1	0	0	1	0
Weight	2.2	2.3	2.5	1.7	1.9	2.1	1.5	2.7

y_i
$x_i[1]$
$x_i[2]$

- What to learn?
  - P(Coin)
  - P(head or tail | Coin = 0) and P(head or tail | Coin = 1)
  - P(weight | Coin = 0) and P(weight| Coin = 1)

CSE 5523:
Naïve Bayes

The Ohio State University

HW & Others

- HW-2 is due on 10/1.

- Have you been reading the textbook?

Generative vs. Discriminative learning

- Training data: $D_{tr} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$

- Goal: fit $p(X, Y)$ to D_{tr}
  - $p(Y)p(\mathrm{X}|Y)$: generative learning
  - $p(X)p(Y|X)$: discriminative learning
    - Usually not estimating $p(X)$

- Prediction
  - Given a test example $\boldsymbol{x}$
  - $\hat{y} = \max_{c} p(\mathrm{Y} = \mathrm{c}|\boldsymbol{x})$; the Bayes Optimal Classifier

Estimating "Parameters of Probability" from Data

- **Maximum likelihood estimation (MLE)**
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\mathcal{D}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N); \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N); \boldsymbol{\theta}) \times \cdots p\big((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N); \boldsymbol{\theta}\big) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p\big((\boldsymbol{x}_n, y_n); \boldsymbol{\theta}\big)$

- **Maximum a posteriori estimation (MAP)**
  - $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, P(\boldsymbol{\theta}|\mathcal{D}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) \times p((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)|\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) \times \prod_n p\big((\boldsymbol{x}_n, y_n)|\boldsymbol{\theta}\big)$

Today

Naïve Bayes

- Review
- Inference
- Advanced study

Naïve Bayes: how to estimate parameters?

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Maintained?
Color?
Mileage?

$$
\begin{aligned}
&(\boldsymbol{x} | Y = c) \\
&= p(x[1] \mid Y = c) \\
&\times p(x[2] \mid Y = c) \\
&\times p(x[3] \mid Y = c)
\end{aligned}
$$

MLE or MAP?

Reading

- See “11. Naive_Bayes (additional reading).PPT” for the comparison on MLE and MAP for Naïve Bayes

Naïve Bayes: formal learning

- MLE:
  - Maximize the "joint probability" of the **IID** data instances

$$\log p(\{\boldsymbol{x}_1, y_1\}, \dots) = \log \prod_{i=1}^{N} p(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i) p(y_i)$$

Naïve Bayes: inference

- **Classification rule for a given $\boldsymbol{x}$:**
  - $\underset{c}{\operatorname{argmax}}\, p(Y = c|\boldsymbol{x}) = \underset{c}{\operatorname{argmax}}\, p(\boldsymbol{x}|Y = c)p(Y = c)$

$$= \underset{c}{\operatorname{argmax}} \prod_{d=1}^{D} p(x[d] \mid Y = c)\, p(Y = c)$$

$$= \underset{c}{\operatorname{argmax}} \left\{ \sum_{d=1}^{D} \log p(x[d] \mid Y = c) \right\} + \log p(Y = c)$$

Naïve Bayes: inference

- **Classification rule for a given $\boldsymbol{x}$:**
  - $\underset{c}{\operatorname{argmax}}\, p(Y = c|\boldsymbol{x}) = \underset{c}{\operatorname{argmax}}\, p(\boldsymbol{x}|Y = c)p(Y = c)$

$$= \underset{c}{\operatorname{argmax}} \prod_{d=1}^{D} p(x[d] \mid Y = c)\, p(Y = c)$$

$$= \underset{c}{\operatorname{argmax}} \left\{ \sum_{d=1}^{D} \log p(x[d] \mid Y = c) \right\} + \log p(Y = c)$$

  - Implementation:
    - For each c, compute $\prod_{d=1}^{D} p(x[d] \mid Y = c)\, p(Y = c)$
    - Usually, we compute $\sum_{d=1}^{D} \log p(x[d] \mid Y = c) + \log p(Y = c)$ to prevent overflow/underflow
    - Output the class c whose above value is the largest

Naïve Bayes: short summary

- **Conditional independence**
  - Can effectively model random variables with continuous and discrete values
  - Conditional independence saves the number of parameters to learn
    - With CI: P(head or tail | Coin = 0), P(heavy or light| Coin = 0)
    - Without CI: P(head or tail, heavy or light | Coin = 0)
  - Conditional independence saves the inference time
- **Bayes rules**
  - Modeling how the cause (e.g., coin type) leads to the effects (e.g., coin weight, head/tail)

Naïve Bayes: as linear classifier

- Assume that $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{+1, -1\}) \}_{i=1}^{N}$

- $\underset{\mathbf{c}}{\operatorname{argmax}} \prod_{d=1}^{D} p(x[d] \mid Y = c)\, p(Y = c)$ can be rewritten as $\operatorname{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$ if
  - $p(x[d] \mid Y = c)$ is Gaussian for all class c, all dimension d
  - For each dimension d, $\sigma_{d,c}^2$ is the same for all class c

Naïve Bayes: as linear classifier

- Assume that $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{+1, -1\}) \}_{i=1}^{N}$

- $\underset{c}{\operatorname{argmax}} \prod_{d=1}^{D} p(x[d] \mid Y = c)\, p(Y = c)$ can be rewritten as $\operatorname{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$ if
  - $p(x[d] \mid Y = c)$ is Gaussian for all class c, all dimension d
  - For each dimension d, $\sigma_{d,c}^2$ is the same for all class c

- $\log p(\boldsymbol{x} | Y = +1) p(Y = +1) = \sum_{d=1}^{D} \log p(x[d] \mid Y = +1) + \log p(Y = +1)$

Naïve Bayes: as linear classifier

- Assume that $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{+1, -1\}) \}_{i=1}^{N}$

- $\underset{c}{\operatorname{argmax}} \prod_{d=1}^{D} p(x[d] \mid Y = c)\, p(Y = c)$ can be rewritten as $\operatorname{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$ if
  - $p(x[d] \mid Y = c)$ is Gaussian for all class c, all dimension d
  - For each dimension d, $\sigma_{d,c}^2$ is the same for all class c

- $\log p(\boldsymbol{x} | Y = +1) p(Y = +1) = \sum_{d=1}^{D} \log p(x[d] \mid Y = +1) + \log p(Y = +1)$
  - Const $+ \sum_{d=1}^{D} -\frac{(x[d]-\mu_{d,c})^2}{2\sigma_{d,c}^2} + \log\lambda =$ Const $+ \sum_{d=1}^{D} \frac{-x[d]^2 + 2\mu_{d,c} x[d] - \mu_{d,c}^2}{2\sigma_d^2} + \log\lambda$
- Predict $+1$ if …

Naïve Bayes: as linear classifier

- Assume that $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{+1, -1\}) \}_{i=1}^{N}$

- $\underset{c}{\operatorname{argmax}} \prod_{d=1}^{D} p(x[d] \mid Y = c)\, p(Y = c)$ can be rewritten as $\operatorname{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$ if
  - $p(x[d] \mid Y = c)$ is Gaussian for all class c, all dimension d
  - For each dimension d, $\sigma_{d,c}^2$ is the same for all class c

- $\log p(\boldsymbol{x} | Y = +1) p(Y = +1) = \sum_{d=1}^{D} \log p(x[d] \mid Y = +1) + \log p(Y = +1)$
  - $\text{Const} + \sum_{d=1}^{D} -\frac{(x[d]-\mu_{d,c})^2}{2\sigma_{d,c}^2} + \log\lambda = \text{Const} + \sum_{d=1}^{D} \frac{-x[d]^2 + 2\mu_{d,c} x[d] - \mu_{d,c}^2}{2\sigma_d^2} + \log\lambda$

- Predict $+1$ if $\sum_{d=1}^{D} \frac{2\mu_{d,+1} x[d] - \mu_{d,+1}^2}{2\sigma_d^2} + \log\lambda > \sum_{d=1}^{D} \frac{2\mu_{d,-1} x[d] - \mu_{d,-1}^2}{2\sigma_d^2} + \log(1-\lambda)$

Naïve Bayes: as linear classifier

Naïve Bayes: special cases for counts

- **Application:** email spam filter
  - Every email has a different number of total word counts (denoted by K)
  - Assume there is a dictionary of a fixed number of <u>unique words</u> D

Naïve Bayes: special cases for counts

- **Application:** email spam filter
  - Every email has a different number of total word counts (denoted by *K*)
  - Assume there is a dictionary of a fixed number of <u>unique words</u> D
  - An email can be represented by a D-dimensional count vector $\boldsymbol{x}$, ignoring word orders
    - Ex: "the dog follows the cat" has the same vector as "the cat follows the dog"
    - Ex: $x[1]$ = # of "dog" in the email; $x[2]$ = # of "cat"; $x[3]$ = # of "the" … → $\boldsymbol{x} = [1, 1, 2, \ldots\ldots]^T$
    - This is called the bag of word (BoW) representation

Naïve Bayes: special cases for counts

- **How to represent** $p(\text{email} \mid Y = c)$?
  - The probability of seeing the count vector $\boldsymbol{x}$, i.e., $p(\boldsymbol{x}|Y = c)$

Naïve Bayes: special cases for counts

- **How to represent** $p(\text{email} \mid Y = c)$?
  - The probability of seeing the count vector $\boldsymbol{x}$, i.e., $p(\boldsymbol{x}|Y = c)$
  - The generative process:
    - Pick a class *c* (i.e., spam or not)
    - Pick an email length at random (i.e., a *K*-word email)
    - For each word *k* independently, choose a unique word (index *d*) from the dictionary with $p(d|Y = c)$

Dear students , I updated the programming part , ……

Same dice for all words of non-spam emails!

Naïve Bayes: special cases for counts

- **How to represent** $p(\text{email} \mid Y = c)$?
  - The probability of seeing the count vector $\boldsymbol{x}$, i.e., $p(\boldsymbol{x}|Y = c)$
  - The generative process:
    - Pick a class *c* (i.e., spam or not)
    - Pick an email length at random (i.e., a *K*-word email)
    - For each word *k* independently, choose <u>a unique word (index *d*) from the dictionary</u> with $p(d|Y = c)$
    - The probability of generating $\boldsymbol{x}$ of class c is

$$p(\boldsymbol{x}|Y = c) = \frac{K!}{x[1]! \times \ldots \times x[D]!} \prod_{d=1}^{D} p(d|Y = c)^{x[d]}$$

The probability to write the d-th unique word

How many times of writing it?

  - Parameters: $p(d|Y = c) = \lambda_c[d]$, where $\sum_{d=1}^{D} \lambda_c[d] = 1$; i.e., one dice for each class

Naïve Bayes: special cases for counts

- MLE:
  - Maximize the "joint probability" of the **IID** data instances

$$\log \prod_{i=1}^{N} p(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i) p(y_i)$$

$$= \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i) + \sum_{i=1}^{N} \log p(y_i) \propto \sum_{i=1}^{N} \log \prod_{d=1}^{D} p(d | y_i)^{\boldsymbol{x}_i[d]} + \sum_{i=1}^{N} \log p(y_i)$$

$$= \sum_{i=1}^{N} \sum_{d=1}^{D} \boldsymbol{x}_i[d] \log \lambda_{y_i}[d] + \sum_{i=1}^{N} \log p(y_i)$$

Naïve Bayes: special cases for counts

- Parameter estimation:
  - $\lambda_c[d] = \frac{\sum_{i=1}^{N} \boldsymbol{I}[y_i=c]x_i[d] \ + (\alpha_c[d]-1)}{\sum_{i=1}^{N} \boldsymbol{I}[y_i=c] \sum_{d=1}^{D} x_i[d] \ + \sum_{d=1}^{D}(\alpha_c[d]-1)}$
  - When $\alpha_c[d] = 1$, $\lambda_c[d] = \frac{\#\text{ of times word } d \text{ appears in all emails of class } c}{\#\text{ of words in all emails of class } c}$

- Prediction:
  - Given a new email with $\boldsymbol{x}$
  - $\underset{c}{\operatorname{argmax}} \prod_{d=1}^{D} p(d|Y=c)^{x[d]} \times p(Y=c)$

Naïve Bayes: as linear classifier

- Assume that $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{+1, -1\}) \}_{i=1}^{N}$

- $\underset{c}{\operatorname{argmax}} \prod_{d=1}^{D} p(d|Y = c)^{x[d]}\, p(Y = c)$ can be rewritten as $\operatorname{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$

Summary: Naïve Bayes

- To apply generative learning, it is common to start with
  - How to model $p(\text{a data instance} \mid Y = y)$
  - How to "sample" or "generate" a data instance given $Y = y$
  - Identify what parameters to learn
  - Write down the joint probability
  - Perform MLE or MAP

Summary: Naïve Bayes

- **Training data:** $D_{tr} = \{ (\boldsymbol{x}_i \in \mathbb{R}^D,\ \mathrm{y}_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$
  - Each dimension of $\boldsymbol{x}_i$ can be binary, categorical, count, or numerical (real)

- **How to construct $\boldsymbol{p(Y = c|x)}$? Bayes' rules**
  - $p(Y = c|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|Y=c)p(Y=c)}{P(\boldsymbol{x})} \propto p(\boldsymbol{x}|Y = c)p(Y = c)$
  - $\underset{\mathrm{c}}{\operatorname{argmax}}\, p(Y = c|\boldsymbol{x}) = \underset{\mathrm{c}}{\operatorname{argmax}}\, p(\boldsymbol{x}|Y = c)p(Y = c)$

Summary: Naïve Bayes

- $p(Y = c; \boldsymbol{\theta})$: Bernoulli or categorical

- $p(\boldsymbol{x}|Y = c; \boldsymbol{\theta})$: **conditional independence assumption:**
  - $\mathrm{p}(\boldsymbol{x}|Y = c) = \mathrm{p}(x[1], x[2], x[3], \dots, x[D] \mid Y = c)$

$$= p(x[1] \mid Y = c)p(x[2] \mid Y = c) \dots \dots p(x[D] \mid Y = c) = \prod_{d=1}^{D} p(x[d] \mid Y = c)$$

Summary: Naïve Bayes

- MLE:
  - Maximize the "joint probability" of the **IID** data instances

$$\log \prod_{i=1}^{N} p(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i) p(y_i)$$

$$= \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | y_i) + \sum_{i=1}^{N} \log p(y_i) = \sum_{i=1}^{N} \log \prod_{d=1}^{D} p(\boldsymbol{x}_i[d] \mid y_i) + \sum_{i=1}^{N} \log p(y_i)$$

$$= \sum_{d=1}^{D} \sum_{i=1}^{N} \log p(\boldsymbol{x}_i[d] \mid y_i) + \sum_{i=1}^{N} \log p(y_i)$$

Summary: generative learning for classification

- Given $\{(\boldsymbol{x}_i,\ \mathrm{y}_i)\}_{i=1}^{N}$
- Assume each instance is generated by
  - $\mathrm{y}_i \sim p(Y)$
  - $\boldsymbol{x}_i \sim p(X \mid Y = y_i)$
- Estimate parameters (MLE)
  - $p(Y), p(X \mid Y = c)$ for each class c
  - Note: $p(Y) \times p(X \mid Y) = p(X, Y)$
- Prediction for $\boldsymbol{x}$
  - $\underset{c}{\operatorname{argmax}}\, p(Y = c \mid \boldsymbol{x})$
  - $\underset{c}{\operatorname{argmax}}\, p(\boldsymbol{x} \mid Y = c)p(Y = c)$
  - Which class is more likely to generate $\boldsymbol{x}$?

Naïve Bayes

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Registered?
Color?
Mileage?

Naïve Bayes: MLE

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Registered?
Color?
Mileage?

Bernoulli distribution: $p(x[d] = 1 | Y = c) = \lambda_{d,c}$

$$\lambda_{d,c} = \frac{\sum_{i=1}^{N} \mathbf{1}[y_i = c]\mathbf{1}[x_i[d] = 1]}{\sum_{i=1}^{N} \mathbf{1}[y_i = c]}$$

Each pair of (dimension, class) has a specific coin

The generative process:

1. Sample one class c
2. Flip the corresponding coin with $\lambda_{d,c}$

Naïve Bayes: MAP

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Registered?
Color?
Mileage?

Bernoulli distribution: $p(x[d] = 1 | Y = c) = \lambda_{d,c}$

$$\lambda_{d,c} = \frac{\sum_{i=1}^{N} \mathbf{1}[y_i = c]\mathbf{1}[x_i[d] = 1] + (\alpha_{d,c} - 1)}{\sum_{i=1}^{N} \mathbf{1}[y_i = c] + (\alpha_{d,c} + \beta_{d,c} - 2)}$$

Each pair of (dimension, class) has a specific coin

The generative process:

1. Sample one class c
2. Flip the corresponding coin with $\lambda_{d,c}$

Naïve Bayes: MLE

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Registered?
Color?
Mileage?

category m

Categorical distribution: $p(x[d] = m | Y = c) = \lambda_{d,c}[m]$

$$\lambda_{d,c}[m] = \frac{\sum_{i=1}^{N} \mathbf{1}[y_i = c]\mathbf{1}[x_i[d] = m]}{\sum_{i=1}^{N} \mathbf{1}[y_i = c]}$$

Each pair of (dimension, class) has a specific dice

The generative process:

1. Sample one class c
2. Throw the corresponding dice with $\lambda_{d,c}$

Naïve Bayes: MAP

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Registered?
Color?
Mileage?

category m

Categorical distribution: $p(x[d] = m | Y = c) = \lambda_{d,c}[m]$

$$\lambda_{d,c}[m] = \frac{\sum_{i=1}^{N} \mathbf{1}[y_i = c]\mathbf{1}[x_i[d] = m] + (\alpha_{d,c}[m] - 1)}{\sum_{i=1}^{N} \mathbf{1}[y_i = c] + \sum_{m=1}^{M} \left(\alpha_{d,c}[m] - 1\right)}$$

Each pair of (dimension, class) has a specific dice

The generative process:

1. Sample one class c
2. Throw the corresponding dice with $\lambda_{d,c}$

Naïve Bayes: MLE

$\boldsymbol{x}$ is 3-dimensional car profile; $D_{tr} = \{ (\boldsymbol{x}_i,\ y_i \in \{1, 2, \dots, C\}) \}_{i=1}^{N}$

Registered?
Color?
Mileage?

Gaussian distribution: $p(x[d]|Y = c) = \mathcal{N}\left(\mu_{d,c}, \sigma_{d,c}^2\right)$

$$\mu_{d,c} = \frac{1}{N_c} \sum_{i=1}^{N} \mathbf{1}[y_i = c] x[d]$$

$$\sigma_{d,c}^2 = \frac{1}{N_c} \sum_{i=1}^{N} \mathbf{1}[y_i = c] \left(x[d] - \mu_{d,c}\right)^2$$

Each pair of (dimension, class) has a specific Gaussian

The generative process:

1. Sample one class c
2. Sample from the corresponding Gaussian with $\mu_{d,c}$ and $\sigma_{d,c}^2$

Naïve Bayes: special cases for counts: MAP

- Parameter estimation:
  - $\lambda_c[d] = \frac{\sum_{i=1}^{N} \boldsymbol{I}[y_i=c]x_i[d] \ + \ (\alpha_c[d]-1)}{\sum_{i=1}^{N} \boldsymbol{I}[y_i=c] \sum_{d=1}^{D} x_i[d] \ + \ \sum_{d=1}^{D}(\alpha_c[d]-1)}$
  - When $\alpha_c[d] = 1$, $\lambda_c[d] = \frac{\text{\# of times word } d \text{ appears in all emails of class } c}{\text{\# of words in all emails of class } c}$

- Prediction:
  - Given a new email with $\boldsymbol{x}$
  - $\underset{c}{\operatorname{argmax}} \prod_{d=1}^{D} p(d|Y=c)^{x[d]} \times p(Y=c)$

Naïve Bayes: special cases for counts: MLE

- Parameter estimation:
  - $\lambda_c[d] = \frac{\sum_{i=1}^{N} \boldsymbol{I}[y_i = c] x_i[d]}{\sum_{i=1}^{N} \boldsymbol{I}[y_i = c] \sum_{d=1}^{D} x_i[d]}$
  - $\lambda_c[d] = \frac{\text{\# of times word } d \text{ appears in all emails of class } c}{\text{\# of words in all emails of class } c}$

- Prediction:
  - Given a new email with $\boldsymbol{x}$
  - $\underset{c}{\operatorname{argmax}} \prod_{d=1}^{D} p(d|Y = c)^{x[d]} \times p(Y = c)$

CSE 5523:
Logistic Regression

The Ohio State University

Quick Notes

- HW2 is due next Tuesday at midnight.
- HW3 will be released by next Wednesday at midnight.
- Midterm is on 10/17 in class: you can have up to 5 papers (10 pages) of cheat sheets (A4/letter size).

Generative vs. Discriminative learning

- Training data: $D_{tr} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$

- Goal: fit $p(X, Y)$ to D_{tr}
  - $p(Y)p(\mathrm{X}|Y)$: generative learning
  - $p(X)p(Y|X)$: discriminative learning
    - Usually not estimating $p(X)$

- Prediction (in the MLE or MAP scenario, mostly we do)
  - Given a test example $\boldsymbol{x}$
  - $\hat{y} = \max_{c} p(\mathrm{Y} = \mathrm{c}|\boldsymbol{x})$; the Bayes Optimal Classifier

Naïve Bayes: generative learning

- $p(Y = c; \boldsymbol{\theta})$: Bernoulli or categorical

- $p(\boldsymbol{x}|Y = c; \boldsymbol{\theta})$: **conditional independence assumption:**
  - $\mathrm{p}(\boldsymbol{x}|Y = c) = \mathrm{p}(x[1], x[2], x[3], \dots, x[D] \mid Y = c)$

$$= p(x[1] \mid Y = c)p(x[2] \mid Y = c) \dots \dots p(x[D] \mid Y = c) = \prod_{d=1}^{D} p(x[d] \mid Y = c)$$

Naïve Bayes: generative learning

- $p(Y = c; \boldsymbol{\theta})$: Bernoulli or categorical

- $p(\boldsymbol{x}|Y = c; \boldsymbol{\theta})$: **conditional independence assumption:**
  - $\mathrm{p}(\boldsymbol{x}|Y = c) = \mathrm{p}(x[1], x[2], x[3], \dots, x[D] \mid Y = c)$

    $= p(x[1] \mid Y = c)p(x[2] \mid Y = c) \dots \dots p(x[D] \mid Y = c) = \prod_{d=1}^{D} p(x[d] \mid Y = c)$

- How to learn from data?
  - So far, the distributions we use give "closed-form" solutions
  - For some other distributions, gradient descents may be needed
  - Note: For a Bernoulli distribution, the outcomes are not necessarily $\{1, 0\}$.

Today

Logistic regression

- Modeling
- Learning
- Learning (implementation)
- GDA vs. NB vs. Logistic regression
- Another perspective

Logistic regression

- **Training data:** $D_{tr} = \left\{ \left(\boldsymbol{x}_i \in \mathbb{R}^D,\ \mathrm{y}_i \in \{+1, -1\} \right) \right\}_{i=1}^{N}$

- **How to construct $\boldsymbol{p}(Y = +1|\boldsymbol{x})$?**
  - Conditional Bernoulli; i.e., $p(Y = +1|\boldsymbol{x}) = \lambda(\boldsymbol{x})$
  - Assumption: $\lambda(\boldsymbol{x}) = \frac{1}{1+e^{-(\boldsymbol{w}^T \boldsymbol{x}+b)}} = \frac{1}{1+e^{-(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}})}}$
  - Reason 1: from Naïve Bayes (see HW #3)

- **Linear or nonlinear classifier?**

Logistic regression

- **Training data:** $D_{tr} = \{ (\boldsymbol{x}_i \in \mathbb{R}^D,\ \mathrm{y}_i \in \{+1, -1\}) \}_{i=1}^{N}$

- **How to construct $\boldsymbol{p}(Y = +1|\boldsymbol{x})$?**
  - Conditional Bernoulli; i.e., $p(Y = +1|\boldsymbol{x}) = \lambda(\boldsymbol{x})$
  - Assumption: $\lambda(\boldsymbol{x}) = \frac{1}{1+e^{-(\boldsymbol{w}^T\boldsymbol{x}+b)}} = \frac{1}{1+e^{-(\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}})}}$
  - Reason 1: from Naïve Bayes (see HW #3)

- **Linear or nonlinear classifier?**
  - Dependent on the decision boundary!
  - Bayes optimal classifier predicts $+1$ when $\lambda(\boldsymbol{x}) > 0.5$; i.e., when $\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}} > 0$

Logistic regression

- Reason 2: ratio for classification
  - Linear: $\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}} > 0$ for class + 1
  - Probability: $r = \frac{p(Y=+1|\boldsymbol{x})}{1-p(Y=+1|\boldsymbol{x})} > 1$ or $\log r = \log\frac{p(Y=+1|\boldsymbol{x})}{1-p(Y=+1|\boldsymbol{x})} > 0$

Logistic regression

- Reason 2: ratio for classification
  - Linear: $\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}} > 0$ for class + 1
  - Probability: $r = \frac{p(Y=+1|\boldsymbol{x})}{1-p(Y=+1|\boldsymbol{x})} > 1$ or $\log r = \log\frac{p(Y=+1|\boldsymbol{x})}{1-p(Y=+1|\boldsymbol{x})} > 0$

- Let $\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}} = \log\frac{p(Y=+1|\boldsymbol{x})}{1-p(Y=+1|\boldsymbol{x})}$
  - $e^{-(\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}})} = \frac{1-p(Y=+1|\boldsymbol{x})}{p(Y=+1|\boldsymbol{x})}$ and then $e^{-(\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}})}p(Y=+1|\boldsymbol{x}) = 1 - p(Y=+1|\boldsymbol{x})$
  - $p(Y=+1|\boldsymbol{x})\left(1+e^{-(\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}})}\right) = 1$
  - $p(Y=+1|\boldsymbol{x}) = \frac{1}{1+e^{-(\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{x}})}}$

Sigmoid function

- $p(Y = +1|\boldsymbol{x}) = \frac{1}{1+e^{-(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}})}} = \rho\left(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}\right)$
- $p(Y = -1|\boldsymbol{x}) = 1 - p(Y = +1|\boldsymbol{x})$
  - $1 - \frac{1}{1+e^{-(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}})}} = \frac{e^{-\left(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}\right)}}{1+e^{-(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}})}} = \frac{1}{1+e^{(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}})}}$

- $p(y|\boldsymbol{x}) = \frac{1}{1+e^{-y(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}})}} = \rho\left(y\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}\right)$

- Sigmoid function has
  - Domain: $\mathbb{R}$
  - Range: $[0, 1]$ (popular for probability)
  - Monotonically increasing

Logistic regression

- Modeling $p(y|\boldsymbol{x};\boldsymbol{\theta})$, so discriminative learning

- MLE: $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr};\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta}) p(\boldsymbol{x}_n)$

- MAP: $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) \prod_n p(y_n|\boldsymbol{x}_n,\boldsymbol{\theta}) p(\boldsymbol{x}_n)$

Logistic regression (MLE)

- MLE: $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr};\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

$$\log \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i;\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(y_i|\boldsymbol{x}_i;\boldsymbol{\theta}) = \sum_{i=1}^{N} \log \rho\left(y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)$$

$$= \sum_{i=1}^{N} \log \frac{1}{1+e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}} = \sum_{i=1}^{N} -\log\left(1+e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right)$$

- $\widetilde{\boldsymbol{w}}^{\mathrm{MLE}} = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmax}} \sum_{i=1}^{N} -\log\left(1+e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right) = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_{i=1}^{N} \log\left(1+e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right)$
  - How to solve?

Logistic regression (MLE)

- $\widetilde{\boldsymbol{w}}^{\mathrm{MLE}} = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmax}} \sum_{i=1}^{N} -\log\left(1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right) = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_{i=1}^{N} \log\left(1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right)$
  - How to solve?
  - No closed-form solution for $\nabla_{\widetilde{\boldsymbol{w}}} \left\{\sum_{i=1}^{N} \log\left(1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right)\right\} = 0$
  - Gradient descent!

Logistic regression (MAP)

- MAP: $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr};\boldsymbol{\theta})p(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) \prod_n p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta})p(\boldsymbol{x}_n)$
- $p(\boldsymbol{\theta}) = \mathcal{N}\left(\mathbf{0}, \sigma^2 \boldsymbol{I}\right)$

$$\log p(\boldsymbol{\theta}) \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i;\boldsymbol{\theta}) \propto \sum_{i=1}^{N} -\log\left(1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right) - \frac{1}{2\sigma^2}\|\widetilde{\boldsymbol{w}}\|_2^2$$

- $\widetilde{\boldsymbol{w}}^{\mathrm{MAP}} = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_{i=1}^{N} \log\left(1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right) + \frac{1}{2\sigma^2}\|\widetilde{\boldsymbol{w}}\|_2^2$
  - Gradient descent!

Logistic regression (MLE)

- Some textbooks use $y_i \in \{+1, 0\}$ rather than $y_i \in \{+1, -1\}$

$$\log \prod_{n=1}^{N} p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})$$

Logistic regression (MLE)

- Some textbooks use $y_i \in \{+1, 0\}$ rather than $y_i \in \{+1, -1\}$

Basically, no difference in algorithms, but difference in math expressions

$$\log \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta})$$

$$= \sum_{n=1}^{N} \log\left(p(+1|\boldsymbol{x}_n;\boldsymbol{\theta})^{y_n} \times \left(1 - p(+1|\boldsymbol{x}_n;\boldsymbol{\theta})\right)^{1-y_n}\right)$$

$$= \sum_{n=1}^{N} \boxed{y_n \log p(+1|\boldsymbol{x}_n;\boldsymbol{\theta}) + (1-y_n)\log\left(1 - p(+1|\boldsymbol{x}_n;\boldsymbol{\theta})\right)}$$

Negative of this is called Binary cross entropy (BCE) loss

$$= \sum_{n=1}^{N} y_n \times \log \frac{1}{1 + e^{-\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} + (1-y_n) \times \log\left(\frac{e^{-\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}}{1 + e^{-\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}}\right)$$

Logistic regression (demo)

$p(+1|\boldsymbol{x}_1; \boldsymbol{w}, b) > 0.5$

$p(+1|\boldsymbol{x}_2; \boldsymbol{w}, b) > 0.5$

$p(+1|\boldsymbol{x}_1; \boldsymbol{w}, b) > 0.5$

$p(+1|\boldsymbol{x}_2; \boldsymbol{w}, b) < 0.5$

Logistic regression (demo)

Gradient descent (GD) for logistic regression

- Initialize $\widetilde{\boldsymbol{w}'}$
- For $t = 1{:}T$
  - $\widetilde{\boldsymbol{w}'} \leftarrow \widetilde{\boldsymbol{w}'} - \eta \nabla_{\widetilde{\boldsymbol{w}}} E\left(\widetilde{\boldsymbol{w}'}\right)$; $E(\widetilde{\boldsymbol{w}})$: the loss function you want to minimize!
  - Until T is reached or $\widetilde{\boldsymbol{w}'}$ converges (e.g., $\left\|\nabla_{\widetilde{\boldsymbol{w}}} E(\widetilde{\boldsymbol{w}'})\right\|_2 < \varepsilon$ or change of $E(\widetilde{\boldsymbol{w}'})$ is small)

Gradient descent (GD) for logistic regression

- Initialize $\widetilde{\boldsymbol{w}'}$
- For $t = 1{:}T$
  - $\widetilde{\boldsymbol{w}'} \leftarrow \widetilde{\boldsymbol{w}'} - \eta \nabla_{\widetilde{\boldsymbol{w}}} E(\widetilde{\boldsymbol{w}'})$; $E(\widetilde{\boldsymbol{w}})$: the loss function you want to minimize!
  - Until T is reached or $\widetilde{\boldsymbol{w}'}$ converges (e.g., $\left\|\nabla_{\widetilde{\boldsymbol{w}}} E(\widetilde{\boldsymbol{w}'})\right\|_2 < \varepsilon$ or change of $E(\widetilde{\boldsymbol{w}'})$ is small)

- Note:
  - If $E(\widetilde{\boldsymbol{w}}) = \frac{1}{N}\sum_i e_i(\widetilde{\boldsymbol{w}})$; for example, $e_i(\widetilde{\boldsymbol{w}})$ is the loss on the *i*-th example
  - $\nabla_{\widetilde{\boldsymbol{w}}} E(\widetilde{\boldsymbol{w}}) = \frac{1}{N}\sum_n \nabla_{\widetilde{\boldsymbol{w}}} e_n(\widetilde{\boldsymbol{w}})$
  - See the derivations of the gradients in later slides
  - While the gradients formulas are different for $\mathrm{y}_i \in \{+1, -1\}$ or $\mathrm{y}_i \in \{+1, 0\}$, the gradient values are basically the same.

Gradient descent (GD) for logistic regression

- $e_n(\widetilde{\boldsymbol{w}}) = \log\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$ for $y_i \in \{+1, -1\}$

$$\boxed{\nabla_{\widetilde{\boldsymbol{w}}} e_n(\widetilde{\boldsymbol{w}}) = \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times \nabla_{\widetilde{\boldsymbol{w}}}\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)}$$

Derivative for Log

$$= \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}$$

$$= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}$$

$$= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times \frac{1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}{\rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}$$

$$= -y_n \left(1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)\right)(\widetilde{\boldsymbol{x}}_n)$$

Gradient descent (GD) for logistic regression

- $e_n(\widetilde{\boldsymbol{w}}) = \log\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$ for $y_i \in \{+1, -1\}$

$$\nabla_{\widetilde{\boldsymbol{w}}} e_n(\widetilde{\boldsymbol{w}}) = \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times \nabla_{\widetilde{\boldsymbol{w}}}\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$$

$$\boxed{= \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}}$$

Derivative for constants and exponential

$$= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}$$

$$= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times \frac{1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}{\rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}$$

$$= -y_n\left(1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)\right)(\widetilde{\boldsymbol{x}}_n)$$

Gradient descent (GD) for logistic regression

- $e_n(\widetilde{\boldsymbol{w}}) = \log\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$ for $y_i \in \{+1, -1\}$

$$\nabla_{\widetilde{\boldsymbol{w}}} e_n(\widetilde{\boldsymbol{w}}) = \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times \nabla_{\widetilde{\boldsymbol{w}}}\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$$

$$= \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}$$

$$\boxed{= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}}$$

The first term is just sigmoid

$$= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times \frac{1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}{\rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}$$

$$= -y_n\left(1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)\right)(\widetilde{\boldsymbol{x}}_n)$$

Gradient descent (GD) for logistic regression

- $e_n(\widetilde{\boldsymbol{w}}) = \log\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$ for $y_i \in \{+1, -1\}$

$$\nabla_{\widetilde{\boldsymbol{w}}} e_n(\widetilde{\boldsymbol{w}}) = \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times \nabla_{\widetilde{\boldsymbol{w}}}\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$$

$$= \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}$$

$$= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}$$

$$\boxed{= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times \frac{1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}{\rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}}$$

$$= -y_n\left(1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)\right)(\widetilde{\boldsymbol{x}}_n)$$

The last term can be re-written

Gradient descent (GD) for logistic regression

- $e_n(\widetilde{\boldsymbol{w}}) = \log\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$ for $y_i \in \{+1, -1\}$

$$\nabla_{\widetilde{\boldsymbol{w}}} e_n(\widetilde{\boldsymbol{w}}) = \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times \nabla_{\widetilde{\boldsymbol{w}}}\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$$

$$= \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}$$

$$= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}$$

$$= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times \frac{1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}{\rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}$$

$$= -y_n \boxed{\left(1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)\right)} (\widetilde{\boldsymbol{x}}_n)$$

Softer update

Gradient descent (GD) for logistic regression

- $e_n(\widetilde{\boldsymbol{w}}) = \log\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$ for $y_i \in \{+1, -1\}$

$$\nabla_{\widetilde{\boldsymbol{w}}} e_n(\widetilde{\boldsymbol{w}}) = \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times \nabla_{\widetilde{\boldsymbol{w}}}\left(1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}\right)$$

$$= \frac{1}{1 + e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}} \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}$$

$$= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times e^{-y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n}$$

$$= \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) \times (-y_n \widetilde{\boldsymbol{x}}_n) \times \frac{1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}{\rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)}$$

$$= -y_n \boxed{\left(1 - \rho\left(y_n \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)\right)} (\widetilde{\boldsymbol{x}}_n)$$

Softer update

Logistic regression:

$$\widetilde{\boldsymbol{w}'} \leftarrow \widetilde{\boldsymbol{w}'} + \eta \sum_n y_n \left(1 - \rho\left(y_n \widetilde{\boldsymbol{w}'}^T \widetilde{\boldsymbol{x}}\right)\right) \widetilde{\boldsymbol{x}}_n$$

$$\widetilde{\boldsymbol{w}'} \leftarrow \widetilde{\boldsymbol{w}'} + \eta \sum_n y_n \left(1 - p(y_n | \widetilde{\boldsymbol{x}}_n)\right) \widetilde{\boldsymbol{x}}_n$$

Perceptron:

$$\widetilde{\boldsymbol{w}'} \leftarrow \widetilde{\boldsymbol{w}'} + \eta y_n \widetilde{\boldsymbol{x}}_n$$

Gradient descent (GD) for logistic regression

- $e_n(\widetilde{\boldsymbol{w}}) = -\left[y_n \log p(+1|\boldsymbol{x}_n;\boldsymbol{\theta}) + (1-y_n)\log\left(1-p(+1|\boldsymbol{x}_n;\boldsymbol{\theta})\right)\right]$ for $y_i \in \{+1, 0\}$

$$\nabla_{\widetilde{\boldsymbol{w}}} e_n(\widetilde{\boldsymbol{w}}) = -\left[y_n \nabla_{\widetilde{\boldsymbol{w}}} \log p(+1|\boldsymbol{x}_n;\boldsymbol{\theta}) + (1-y_n)\nabla_{\widetilde{\boldsymbol{w}}} \log\left(1-p(+1|\boldsymbol{x}_n;\boldsymbol{\theta})\right)\right]$$
$$= -\left[y_n \nabla_{\widetilde{\boldsymbol{w}}} \log \rho\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right) + (1-y_n)\nabla_{\widetilde{\boldsymbol{w}}} \log\left(1-\rho\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_n\right)\right)\right]$$
$$= -\left[y_n \times \left(1-\rho\left(\boldsymbol{w}^T \boldsymbol{x}_n\right)\right)\boldsymbol{x}_n - (1-y_n) \times \rho\left(\boldsymbol{w}^T \boldsymbol{x}_n\right)\boldsymbol{x}_n\right]$$
$$= -\left(y_n - \rho\left(\boldsymbol{w}^T \boldsymbol{x}_n\right)\right)\boldsymbol{x}_n$$

$$\boxed{\text{“}=\text{”}: \frac{d\,\rho(a)}{d\,a} = \frac{d\left(\frac{1}{1+e^{-a}}\right)}{da} = \frac{e^{-a}}{(1+e^{-a})^2} = \frac{1}{1+e^{-a}} \times \frac{e^{-a}}{1+e^{-a}} = \rho(a)\left(1-\rho(a)\right)}$$

Stochastic gradient descent (SGD) for logistic regression

- Initialize $\widetilde{\boldsymbol{w}'}$
- For $t = 1{:}T$
  - Randomly sample an example $(\boldsymbol{x}_i,\ \mathrm{y}_i)$ from $D_{tr} = \{\ (\boldsymbol{x}_i\ \ \mathrm{y}_i)\ \}_{i=1}^{N}$
  - $\widetilde{\boldsymbol{w}'} \leftarrow \widetilde{\boldsymbol{w}'} - \eta \nabla_{\widetilde{\boldsymbol{w}}}\, e_i(\widetilde{\boldsymbol{w}'})$
  - Until T is reached or $\widetilde{\boldsymbol{w}'}$ converges

Caution

- Caution:
  - The learning rate η must be chosen properly for convergence. It can be dynamic.

Short summary: Logistic regression

- MLE: $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

$$\log \prod_{i=1}^{N} p(y_i | \boldsymbol{x}_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(y_i | \boldsymbol{x}_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \rho\left(y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)$$

$$= \sum_{i=1}^{N} \log \frac{1}{1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}} = \sum_{i=1}^{N} -\log\left(1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right)$$

- How to maximize?
  - Gradient descent!

GDA vs. NB vs. Logistic regression

- Training data: $D_{tr} = \left\{ \left(\boldsymbol{x}_i \in \mathbb{R}^D,\ \mathrm{y}_i \in \{+1, -1\} \right) \right\}_{i=1}^{N}$
- Under certain assumptions, all of them will lead to a linear classifier $\mathrm{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$
- However, the learned $\boldsymbol{w}$ and b are different
  - GDA and NB need NO gradient descents; they have closed-form solutions
  - GDA and NB are suitable for streaming settings

GDA vs. NB vs. Logistic regression

- Training data: $D_{tr} = \left\{ \left(\boldsymbol{x}_i \in \mathbb{R}^D,\ \mathrm{y}_i \in \{+1, -1\} \right) \right\}_{i=1}^{N}$
- Under certain assumptions, all of them will lead to a linear classifier $\mathrm{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$
- However, the learned $\boldsymbol{w}$ and b are different
  - GDA and NB need NO gradient descents; they have closed-form solutions
  - GDA and NB are suitable for streaming settings

- **Recap:**
  - A learning algorithm + A hypothesis class
  - GDA, NB, and logistic regression use the same hypothesis class but different algorithms

Generative vs. Discriminative learning

- Generative learning (MLE case)
  - $p(\boldsymbol{x}, y; \boldsymbol{\theta}) = p(\boldsymbol{x}|y; \boldsymbol{\theta})p(y; \boldsymbol{\theta})$
  - Make two modeling assumptions (conditional data + Bernoulli or categorical)
  - If the assumption is correct, then fewer training data are needed

- Discriminative learning (MLE case)
  - $p(\boldsymbol{x}, y; \boldsymbol{\theta}) = p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})p(\boldsymbol{x})$
  - Make one modeling assumptions (conditional Bernoulli or categorical)
  - more flexible but require more data to avoid overfitting

- If the data is truly from a Naive Bayes model, then Logistic Regression and Naive Bayes converge to the exact same result in the limit (but NB is faster).

Logistic regression: another perspective

- **Training data:** $D_{tr} = \left\{ \left(\boldsymbol{x}_i \in \mathbb{R}^D,\ \mathrm{y}_i \in \{+1, -1\} \right) \right\}_{i=1}^{N}$

- Learning: $\widetilde{\boldsymbol{w}}^{\mathrm{MLE}} = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_{i=1}^{N} \log\left(1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right)$

- Prediction: $\operatorname{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$

Logistic regression: another perspective

- **Training data:** $D_{tr} = \left\{ \left(\boldsymbol{x}_i \in \mathbb{R}^D,\ \mathrm{y}_i \in \{+1, -1\} \right) \right\}_{i=1}^{N}$

- Learning: $\widetilde{\boldsymbol{w}}^{\mathrm{MLE}} = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_{i=1}^{N} \log\left(1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right) =$

$$\boxed{\begin{aligned} &\underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_{i=1}^{N} \ell\left(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i, y_i\right) = \underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_{i=1}^{N} \ell\left(h_{\widetilde{\boldsymbol{w}}}(\widetilde{\boldsymbol{x}}_i), y_i\right) = \\ &\underset{\widetilde{\boldsymbol{w}}}{\operatorname{argmin}} \sum_{i=1}^{N} \ell(\hat{y}_i, y_i) \end{aligned}}$$

- Prediction: $\operatorname{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$

"soft" prediction!

Loss function

Empirical risk minimization recap

- In order to achieve a low generalization loss $E_{(x,y)\sim P}[\ell(h(x), y)]$
  - We minimize the training loss + certain regularization
  - Training loss: $\frac{1}{N}\sum_{n=1}^{N} \ell(h(\boldsymbol{x}_n), y_n)$, where $D_{tr} = \{ (\boldsymbol{x}_n \in \mathbb{R}^D,\ \mathrm{y}_n \in \{+1, -1\}) \}_{i=1}^{N}$
  - Ideally, h & ℓ in the training should be the same as h & ℓ in the test (generalization) loss

Empirical risk minimization recap

- In order to achieve a low generalization loss $E_{(x,y)\sim P}[\ell(h(x), y)]$
  - We minimize the training loss + certain regularization
  - Training loss: $\frac{1}{N}\sum_{n=1}^{N} \ell(h(\boldsymbol{x}_n), y_n)$, where $D_{tr} = \{ (\boldsymbol{x}_n \in \mathbb{R}^D,\ \mathrm{y}_n \in \{+1, -1\}) \}_{i=1}^{N}$
  - Ideally, h & ℓ in the training should be the same as h & ℓ in the test (generalization) loss

- For classification, we care
  - 0-1 loss: $\ell(y, \hat{y}) = 1[\mathrm{y} \neq \hat{y}]$; 1[True] = 1, 1[False] = 0 is called an indicator function

- For linear classifier, we use
  - $\hat{y} = h(\boldsymbol{x}) = \mathrm{sign}(\boldsymbol{w}^T \boldsymbol{x})$; the bias term b is absorbed into $\boldsymbol{w}$ for brevity

Problems with 0-1 losses

- $\ell(y, \hat{y}) = 1[\mathrm{y} \neq \hat{y}]$
  - Not differentiable w.r.t. $\hat{y}$ (i.e., the prediction)
  - Example: for $\mathrm{y}_i \in \{+1, -1\}$, we want
    - $\hat{y}$ (and thus $\boldsymbol{w}^T\boldsymbol{x}$) and y to be of the same sign

Problems with 0-1 losses

- $\ell(y, \hat{y}) = 1[\mathrm{y} \neq \hat{y}]$
  - Not differentiable w.r.t. $\hat{y}$ (i.e., the prediction)
  - Example: for $\mathrm{y}_i \in \{+1, -1\}$, we want
    - $\hat{y}$ (and thus $\boldsymbol{w}^T\boldsymbol{x}$) and y to be of the same sign
  - Hard to optimize $\boldsymbol{w}$
    - No signals of where $\boldsymbol{w}$ should move

Problems with 0-1 losses

- $\ell(y, \hat{y}) = 1[\mathrm{y} \neq \hat{y}]$
  - Not differentiable w.r.t. $\hat{y}$ (i.e., the prediction)
  - Example: for $\mathrm{y}_i \in \{+1, -1\}$, we want
    - $\hat{y}$ (and thus $\boldsymbol{w}^T\boldsymbol{x}$) and y to be of the same sign
  - Hard to optimize $\boldsymbol{w}$
    - No signals of where $\boldsymbol{w}$ should move

- We need a relaxed (surrogate) loss "in training" to encourage:
  - $\mathrm{y} = 1, \quad \boldsymbol{w}^T\boldsymbol{x} \uparrow$
  - $\mathrm{y} = -1, \quad \boldsymbol{w}^T\boldsymbol{x} \downarrow$

Logistic loss

- Let $p(+1|\boldsymbol{x};\boldsymbol{w}) = \rho\left(\boldsymbol{w}^T\boldsymbol{x}\right)$
- Let $p(-1|\boldsymbol{x};\boldsymbol{w}) = 1 - \rho\left(\boldsymbol{w}^T\boldsymbol{x}\right)$
- Want to maximize: $p(y_n|\boldsymbol{x}_n;\boldsymbol{w})$
- Want to minimize: $-\log p(y_n|\boldsymbol{x}_n;\boldsymbol{w})$
  $= -\log\rho\left(y_n\boldsymbol{w}^T\boldsymbol{x}\right)$

- When
  - $\mathrm{y} = 1, \quad \boldsymbol{w}^T\boldsymbol{x} \uparrow$
  - $\mathrm{y} = -1, \quad \boldsymbol{w}^T\boldsymbol{x} \downarrow$

Questions!

- What properties must a surrogate loss have?
- Are there any other loss functions?
- Are there any other regularization functions, beyond $\lambda\|\boldsymbol{w}\|_2^2$?

A first glance

- What properties must a surrogate loss have?
  - Upper bounding the 0-1 loss: so, small surrogate loss implies small 0-1 loss
- Are there any other loss functions?

- Are there any other regularization functions, beyond $\lambda \|\boldsymbol{w}\|_2^2$?
  - Yes: as long as it is "lower bounded" and characterizes model complexity

CSE 5523:
Linear SVM

The Ohio State University

Logistic regression

- MLE: $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

$$\log \prod_{i=1}^{N} p(y_i | \boldsymbol{x}_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(y_i | \boldsymbol{x}_i; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \rho\left(y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i\right)$$

$$= \sum_{i=1}^{N} \log \frac{1}{1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}} = \sum_{i=1}^{N} -\log\left(1 + e^{-y_i \widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{x}}_i}\right)$$

- How to maximize?
  - Gradient descent!

Today

Support vector machines (SVM)

- Overview
- Margin
- Maximum margin classifier
- Support vectors
- Soft-margin SVM
- Summary

Support vector machines (SVM)

- A linear classifier viewed as an extension of the perceptron algorithm

- Given a linear separable D_{tr}
  - There are infinite many $(\boldsymbol{w}, b)$ for zero 0-1 loss
  - Perceptron finds an arbitrary separating hyperplane

Support vector machines (SVM)

- A linear classifier viewed as an extension of the perceptron algorithm

- Given a linear separable D_{tr}
  - There are infinite many $(\boldsymbol{w}, b)$ for zero 0-1 loss
  - SVM picks the one that maximizes
    - the distance to the closest data points of both classes
  - SVM finds the maximum margin hyperplane

Margin

- Hypothesis class: hyperplane (a.k.a. linear boundary)
  - $\mathcal{H} = \{\boldsymbol{w}^T\boldsymbol{x} + b = 0\}$; here I explicitly write down b

- Given $D_{tr} = \left\{ \left(\boldsymbol{x}_n \in \mathbb{R}^D,\ \mathrm{y}_n \in \{+1, -1\}\right) \right\}_{i=1}^{N}$
  - Margin γ = "sign" distance from the hyperplane to the closest points of both classes
  - Sign: "positive" for correct prediction; "negative" for wrong prediction

$$\mathrm{sign}\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$$

Margin

- Let's begin with one data point $(\boldsymbol{x}_n, \mathrm{y}_n)$, what is its distance to a hyperplane?

$\boldsymbol{w}^T\boldsymbol{x} + b = 0$

$\boldsymbol{w}^T\boldsymbol{x} + b = 0$

Margin

- Let's begin with one data point $(\boldsymbol{x}_n, \mathrm{y}_n)$, what is its distance to a hyperplane?
  - Let $\boldsymbol{x}_n{}'$ be $\boldsymbol{x}_n$ projection onto the hyperplane
  - Let $\boldsymbol{d}$ be the vector such that $\boldsymbol{x}_n' + \boldsymbol{d} = \boldsymbol{x}_n$

$\boldsymbol{w}^T\boldsymbol{x} + b = 0$

$\boldsymbol{w}^T\boldsymbol{x} + b = 0$

Margin

- Let's begin with one data point $(\boldsymbol{x}_n, \mathrm{y}_n)$, what is its distance to a hyperplane?
  - Let $\boldsymbol{x}_n{}'$ be $\boldsymbol{x}_n$ projection onto the hyperplane
  - Let $\boldsymbol{d}$ be the vector such that $\boldsymbol{x}_n' + \boldsymbol{d} = \boldsymbol{x}_n$

- Some properties
  - $\boldsymbol{d} = \alpha \boldsymbol{w}$, where $\alpha \in \mathbb{R}$ (i.e., in parallel)
  - $\boldsymbol{w}^T \boldsymbol{x}_n{}' + b = 0$

$\boldsymbol{w}^T \boldsymbol{x} + b = 0$
$\boldsymbol{w}^T \boldsymbol{x} + b = 0$

Margin

- Let's begin with one data point $(\boldsymbol{x}_n, \mathrm{y}_n)$, what is its distance to a hyperplane?
  - Let $\boldsymbol{x}_n{}'$ be $\boldsymbol{x}_n$ projection onto the hyperplane
  - Let $\boldsymbol{d}$ be the vector such that $\boldsymbol{x}_n' + \boldsymbol{d} = \boldsymbol{x}_n$

$\boldsymbol{w}^T\boldsymbol{x} + b = 0$
$\boldsymbol{w}^T\boldsymbol{x} + b = 0$

- Some properties
  - $\boldsymbol{d} = \alpha\boldsymbol{w}$, where $\alpha \in \mathbb{R}$ (i.e., in parallel)
  - $\boldsymbol{w}^T\boldsymbol{x}_n{}' + b = 0$

- $\boldsymbol{w}^T\boldsymbol{x}_n' + b = \boldsymbol{w}^T(\boldsymbol{x}_n - \boldsymbol{d}) + b = \boldsymbol{w}^T(\boldsymbol{x}_n - \alpha\boldsymbol{w}) + b = 0$
  - $\alpha = \frac{\boldsymbol{w}^T\boldsymbol{x}_n + b}{\boldsymbol{w}^T\boldsymbol{w}}$
  - $\|\boldsymbol{d}\|_2 = \sqrt{\boldsymbol{d}^T\boldsymbol{d}} = \sqrt{\alpha^2\boldsymbol{w}^T\boldsymbol{w}} = \frac{|\boldsymbol{w}^T\boldsymbol{x}_n + b|}{\|\boldsymbol{w}\|_2}$

Margin

- "Sign" distance for $\boldsymbol{d}$: $\frac{y_n(\boldsymbol{w}^T\boldsymbol{x}_n+b)}{\|\boldsymbol{w}\|_2}$
  - Non-negative if $(\boldsymbol{x}_n, y_n)$ is on the correct side of the hyperplane
    - For $\boldsymbol{w}^T\boldsymbol{x}+b=0$, non-negative if $y_n=1$
    - For $\boldsymbol{w}^T\boldsymbol{x}+b=0$, non-negative if $y_n=-1$

Margin

- "Sign" distance for $\boldsymbol{d}$: $\frac{y_n(\boldsymbol{w}^T\boldsymbol{x}_n+b)}{\|\boldsymbol{w}\|_2}$
  - Non-negative if $(\boldsymbol{x}_n, y_n)$ is on the correct side of the hyperplane
    - For $\boldsymbol{w}^T\boldsymbol{x}+b=0$, non-negative if $y_n=1$
    - For $\boldsymbol{w}^T\boldsymbol{x}+b=0$, non-negative if $y_n=-1$

- Properties:
  - The hyperplane and distance are positive-scale-invariant
  - $\frac{y_n(\beta\boldsymbol{w}^T\boldsymbol{x}_n+\beta b)}{\|\beta\boldsymbol{w}\|_2}=\frac{y_n(\boldsymbol{w}^T\boldsymbol{x}_n+b)}{\|\boldsymbol{w}\|_2}$ for $\beta>0$

Margin

- "Sign" distance for $\boldsymbol{d}$: $\frac{y_n(\boldsymbol{w}^T\boldsymbol{x}_n+b)}{\|\boldsymbol{w}\|_2}$
  - Non-negative if $(\boldsymbol{x}_n, y_n)$ is on the correct side of the hyperplane
    - For $\boldsymbol{w}^T\boldsymbol{x}+b=0$, non-negative if $y_n=1$
    - For $\boldsymbol{w}^T\boldsymbol{x}+b=0$, non-negative if $y_n=-1$

- Properties:
  - The hyperplane and distance are positive-scale-invariant
  - $\frac{y_n(\beta\boldsymbol{w}^T\boldsymbol{x}_n+\beta b)}{\|\beta\boldsymbol{w}\|_2} = \frac{y_n(\boldsymbol{w}^T\boldsymbol{x}_n+b)}{\|\boldsymbol{w}\|_2}$ for $\beta > 0$

- Margin for D_{tr}: $\gamma = \min_{(\boldsymbol{x},y)\in D_{tr}} \frac{y(\boldsymbol{w}^T\boldsymbol{x}+b)}{\|\boldsymbol{w}\|_2}$

$\boldsymbol{w}^T\boldsymbol{x}+b=0$
$\boldsymbol{w}^T\boldsymbol{x}+b=0$

Maximum margin classifier

- Training data: $D_{tr} = \left\{ \left(\boldsymbol{x}_n \in \mathbb{R}^D,\ \mathrm{y}_n \in \{+1, -1\} \right) \right\}_{i=1}^{N}$ (linear separable)

- $(\boldsymbol{w}, b)^{\mathrm{SVM}} = \underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}}\, \gamma(\boldsymbol{w}, b) =$

$$\underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}} \min_{(\boldsymbol{x}_n, \mathrm{y}_n)} \frac{\mathrm{y}_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|_2} = \underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}} \frac{1}{\|\boldsymbol{w}\|_2} \min_{(\boldsymbol{x}_n, \mathrm{y}_n)} \mathrm{y}_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)$$

Maximum margin classifier

- Training data: $D_{tr} = \left\{ \left(\boldsymbol{x}_n \in \mathbb{R}^D,\ \mathrm{y}_n \in \{+1, -1\} \right) \right\}_{i=1}^{N}$ (linear separable)

- $(\boldsymbol{w}, b)^{\mathrm{SVM}} = \underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}}\ \gamma(\boldsymbol{w}, b) =$

$$\underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}} \min_{(\boldsymbol{x}_n, \mathrm{y}_n)} \frac{\mathrm{y}_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|_2} = \underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}} \frac{1}{\|\boldsymbol{w}\|_2} \min_{(\boldsymbol{x}_n, \mathrm{y}_n)} \mathrm{y}_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)$$

independent of $(\boldsymbol{x}_n, \mathrm{y}_n)$

Maximum margin classifier

- Training data: $D_{tr} = \left\{ \left(\boldsymbol{x}_n \in \mathbb{R}^D,\ \mathrm{y}_n \in \{+1, -1\} \right) \right\}_{i=1}^{N}$ (linear separable)

- $(\boldsymbol{w}, b)^{\mathrm{SVM}} = \underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}}\, \gamma(\boldsymbol{w}, b) =$

  $\underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}} \underset{(\boldsymbol{x}_n, \mathrm{y}_n)}{\min} \frac{\mathrm{y}_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|_2} = \underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}} \frac{1}{\|\boldsymbol{w}\|_2} \underset{(\boldsymbol{x}_n, \mathrm{y}_n)}{\min} \mathrm{y}_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)$

- Due to positive-scale-invariant, for $(\boldsymbol{w}, b)$ that perfectly separates data, we can
  - constrain $\underset{(\boldsymbol{x}_n, \mathrm{y}_n)}{\min} \mathrm{y}_n\left(\boldsymbol{w}^T \boldsymbol{x}_n + b\right) = 1$
  - $(\boldsymbol{w}, b)^{\mathrm{SVM}} = \underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}} \frac{1}{\|\boldsymbol{w}\|_2}$, s.t. $\underset{(\boldsymbol{x}_n, \mathrm{y}_n)}{\min} \mathrm{y}_n\left(\boldsymbol{w}^T \boldsymbol{x}_n + b\right) = 1$

Maximum margin classifier

- $(\boldsymbol{w}, b)^{\mathrm{SVM}} = \underset{(\boldsymbol{w}, b)}{\operatorname{argmax}} \frac{1}{\|\boldsymbol{w}\|_2} = \underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \|\boldsymbol{w}\|_2 = \underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \|\boldsymbol{w}\|_2^2 = \underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}$

- Optimization problem:
  - $\underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}, \qquad \text{s.t.} \min_{(\boldsymbol{x}_n, \mathrm{y}_n)} \mathrm{y}_n \left(\boldsymbol{w}^T \boldsymbol{x}_n + b \right) = 1$

Maximum margin classifier

- $(\boldsymbol{w}, b)^{\text{SVM}} = \underset{(\boldsymbol{w}, b)}{\operatorname{argmax}} \frac{1}{\|\boldsymbol{w}\|_2} = \underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \|\boldsymbol{w}\|_2 = \underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \|\boldsymbol{w}\|_2^2 = \underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}$

- Optimization problem:
  - $\underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}, \qquad \text{s.t.} \min_{(\boldsymbol{x}_n, \mathrm{y}_n)} \mathrm{y}_n \left(\boldsymbol{w}^T \boldsymbol{x}_n + b \right) = 1$

- "Equivalent" optimization problem with the same solution
  - $\underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}, \qquad \text{s.t.} \ \forall n, \mathrm{y}_n \left(\boldsymbol{w}^T \boldsymbol{x}_n + b \right) \geq 1$
  - A quadratic optimization problem (quadratic objective; linear constraints) that can be solved efficiently via a QCQP (Quadratically Constrained Quadratic Program) solver.
  - To prove the equivalence in your next homework!

Maximum margin classifier

- Optimization problem for linear "hard-margin" SVM (primal form)
  - $\underset{(\boldsymbol{w},\, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}, \qquad \text{s.t. } \forall n, \mathrm{y}_n \left(\boldsymbol{w}^T \boldsymbol{x}_n + b \right) \geq 1$

Maximum margin classifier

- Optimization problem for linear "hard-margin" SVM (primal form)
  - $\underset{(\boldsymbol{w},\, b)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}, \qquad \text{s.t. } \forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1$

- Properties:
  - Have a "unique" solution whenever a separating hyperplane exists
  - $\mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 0$ is already correct, but we ask more: $\mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1$
    - Perceptron only asks for the former $\mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 0$
  - Interpretation: find the simplest hyperplane (smallest $\|\boldsymbol{w}\|_2^2$) such that all data points lie at least one unit away from it
  - Margin: $\frac{1}{\left\|\boldsymbol{w}^{\mathrm{SVM}}\right\|_2}$

Support vectors

- Some $(\boldsymbol{x}_n, \mathrm{y}_n)$ will have $\mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) = 1$
  - Homework!
- We refer to them as "support vectors (SV)"

Support vectors

- Some $(\boldsymbol{x}_n, \mathrm{y}_n)$ will have $\mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) = 1$
  - Homework!
- We refer to them as "support vectors (SV)"

- Properties:
  - SVs define the margin and thus the hyperplane
  - Removing them, $(\boldsymbol{w}, b)^{\mathrm{SVM}}$ might change
  - Removing other data points won't!
- That is, you can obtain the same $(\boldsymbol{w}, b)^{\mathrm{SVM}}$ from a (much) smaller D_{tr}

Properties

- A maximum-margin hyperplane lies in the middle of two classes

Soft-margin SVM

- $\underset{(\boldsymbol{w},\, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}, \qquad \text{s.t. } \forall n, \mathrm{y}_n \left(\boldsymbol{w}^T \boldsymbol{x}_n + b \right) \geq 1$

Soft-margin SVM

- $\underset{(\boldsymbol{w},\, b)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}, \qquad \text{s.t. } \forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1$
  - No solution if the data is not linear separable (common cases)

- Allow "slight" violation with "slack" variables

- $\underset{(\boldsymbol{w},\, b, \xi_n)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n\,, \qquad \text{s.t. } \forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1 - \xi_n, \xi_n \geq 0$
  - C is a hyper-parameter
  - How to choose it?

Soft-margin SVM

- $\underset{(\boldsymbol{w},\, b)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}, \qquad \text{s.t. } \forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1$
  - No solution if the data is not linear separable (common cases)

- Allow "slight" violation with "slack" variables

- $\underset{(\boldsymbol{w},\, b, \xi_n)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n\,, \qquad \text{s.t. } \forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1 - \xi_n, \ \xi_n \geq 0$
  - C is a hyper-parameter
  - How to choose it? **A: validation**

Why?

Slack variables

- Allow some data points to be closer to the hyperplane or classified wrongly
  - But with penalties in the objective
  - $\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n$

- Extreme cases:
  - If C is large, more like the hard-margin SVM
  - If C is small, favor simple models with errors

$\text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$

Hinge loss

- Original soft-margin SVM
  - $\underset{(\boldsymbol{w}, b, \xi_n)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n$, s.t. $\forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1 - \xi_n, \xi_n \geq 0$
  - Convex quadratic programming: quadratic objective + linear constraints

Hinge loss

- Original soft-margin SVM
  - $\underset{(\boldsymbol{w}, b, \xi_n)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n$, s.t. $\forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1 - \xi_n, \xi_n \geq 0$
  - Convex quadratic programming: quadratic objective + linear constraints

- Unconstrained formulation
  - $\mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1 - \xi_n \Rightarrow \xi_n \geq 1 - \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)$
  - $\xi_n = \max\{1 - \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right), 0\} = \left[1 - \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)\right]_+$
  - $\geq$ disappears because we always want to minimize ξ_n

Hinge loss

- Original soft-margin SVM
  - $\underset{(\boldsymbol{w}, b, \xi_n)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n$, s.t. $\forall n, \mathrm{y}_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - \xi_n, \xi_n \geq 0$
  - Convex quadratic programming: quadratic objective + linear constraints

- Unconstrained formulation
  - $\mathrm{y}_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - \xi_n \Rightarrow \xi_n \geq 1 - \mathrm{y}_n(\boldsymbol{w}^T\boldsymbol{x}_n + b)$
  - $\xi_n = \max\{1 - \mathrm{y}_n(\boldsymbol{w}^T\boldsymbol{x}_n + b), 0\} = [1 - \mathrm{y}_n(\boldsymbol{w}^T\boldsymbol{x}_n + b)]_+$
  - $\geq$ disappears because we always want to minimize ξ_n
  - $\underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \boxed{\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}} + \boxed{C\sum_n[1 - \mathrm{y}_n(\boldsymbol{w}^T\boldsymbol{x}_n + b)]_+}$

training "hinge" loss

regularization

Soft-margin SVM (primal form)

- Unconstrained problem: $\underset{(\boldsymbol{w},\, b\,)}{\operatorname{argmin}} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \ + \ C\sum_n\left[1 - \ \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)\right]_+$
  - How to solve?

Soft-margin SVM (primal form)

- Unconstrained problem: $\underset{(\boldsymbol{w},\, b\,)}{\operatorname{argmin}} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \ + \ C\sum_n \left[1 - \ \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)\right]_+$
  - How to solve? (stochastic) gradient descent
  - $\left[1 - \ \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)\right]_+$ is differentiable except at $1 = \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)$
  - Solve like logistic regression, but now hinge loss

Soft-margin SVM (primal form)

- Unconstrained problem: $\underset{(\boldsymbol{w},\, b\,)}{\operatorname{argmin}} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \ + \ C\sum_n\left[1 - \ \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)\right]_+$
  - How to solve? (stochastic) gradient descent
  - $\left[1 - \ \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)\right]_+$ is differentiable except at $1 = \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)$
  - Solve like logistic regression, but now hinge loss
  - Penalize even for unconfident correct predictions

Support vectors

$\text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$

- Some $(\boldsymbol{x}_n, \mathrm{y}_n)$ will have $\mathrm{y}_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) = 1 - \xi_n$
- We refer to them as "support vectors (SV)"

Support vectors

$\text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$

- Some $(\boldsymbol{x}_n, \mathrm{y}_n)$ will have $\mathrm{y}_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) = 1 - \xi_n$
- We refer to them as "support vectors (SV)"

- Properties: same as before

- Cases (for SVs):
  - $\xi_n = 0$: points are $\frac{1}{\|\boldsymbol{w}^{\text{SVM}}\|_2}$ from the hyperplane
  - $\xi_n < 1$: correctly predicted
  - $\xi_n > 1$: wrongly predicted

Demo

- LibSVM: https://www.csie.ntu.edu.tw/~cjlin/libsvm/

C = 0.01

C = 1

C = 100

C = 10000

Iterative logistic regression and linear SVM

Summary: Support vector machines (SVM)

- Given a linear separable D_{tr}
  - There are infinite many $(\boldsymbol{w}, b)$ for zero 0-1 loss
  - SVM finds the maximum margin separating hyperplane

Summary: Hard-margin vs. Soft-margin SVM

- Linear "hard-margin" SVM
  - $\underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}, \qquad \text{s.t. } \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1$
  - No solution if the data is not linear separable (common cases)

- Linear "soft-margin" SVM
  - $\underset{(\boldsymbol{w}, b, \xi_n)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n\,, \qquad \text{s.t. } \forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1 - \xi_n, \xi_n \geq 0$
  - Allow "slight" violation with "slack" variables ξ_n
  - C is a hyper-parameter

Summary: Hinge loss

- Original soft-margin SVM
  - $\underset{(\boldsymbol{w}, b, \xi_n)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n$, s.t. $\forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1 - \xi_n, \xi_n \geq 0$
  - Convex quadratic programming: quadratic objective + linear constraints
  - Can be solved even faster by taking the special structures of the problem into account

- Unconstrained formulation
  - $\underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \boxed{\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}} + \boxed{C\sum_n\left[1 - \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)\right]_+}$

training "hinge" loss

regularization

Summary: Support vectors

- Some $(\boldsymbol{x}_n, \mathrm{y}_n)$ will have $\mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) = 1 - \xi_n$
  - We refer to them as "support vectors (SV)"
  - SVs define the decision boundary
  - Removing them change the decision boundary
  - Removing others for $\mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) > 1 - \xi_n$ won't

- Cases (for SVs):
  - $\xi_n = 0$: points are $\frac{1}{\|\boldsymbol{w}^{\mathrm{SVM}}\|_2}$ from the hyperplane
  - $\xi_n < 1$: correctly predicted
  - $\xi_n > 1$: wrongly predicted

Summary: Support vectors

- One of the most popular algorithms (its kernel version) before deep learning
- Theoretical guarantee on generalization error
- Convex optimization: convex objective function + convex constraint
  - Local optimum = global optimum
  - The solution is independent of the initialization
  - The optimal set is convex
  - If the objective function is strictly convex, then the problem has at most one optimal solution

CSE 5523:
Convex Optimization + Multi-Class Classification

The Ohio State University

Midterm exam and HW # 3

- Midterm
  - 10/17, in class
  - Not open book, but you can bring 5 “papers” (10 pages) of cheat sheets. Printing is fine.
  - This means, you can practice how to make concise notes for yourself!
  - More information will be announced via Carmen!

- HW # 3
  - Due: Due 10/15

- Homework
  - Not meant to be a practice for your exam
  - But to verify or practice some important concepts of ML

Today

Support vector machines (SVM)

- Review
- Soft-margin SVM
- Summary

Constrained convex optimization

Multi-class classification modeling

Recap: Support vector machines (SVM)

- Given a linear separable D_{tr}
  - There are infinite many $(\boldsymbol{w}, b)$ for zero 0-1 loss
  - SVM finds the maximum margin separating hyperplane

Recap: Margin

- "Sign" distance for $\boldsymbol{d}$: $\frac{y_n(\boldsymbol{w}^T\boldsymbol{x}_n+b)}{\|\boldsymbol{w}\|_2}$

- Properties:
  - The hyperplane and distance are positive-scale-invariant
  - $\frac{y_n(\beta\boldsymbol{w}^T\boldsymbol{x}_n+\beta b)}{\|\beta\boldsymbol{w}\|_2} = \frac{y_n(\boldsymbol{w}^T\boldsymbol{x}_n+b)}{\|\boldsymbol{w}\|_2}$ for $\beta > 0$

- Margin: $\gamma = \min_{(\boldsymbol{x},y)\in D_{tr}} \frac{y(\boldsymbol{w}^T\boldsymbol{x}+b)}{\|\boldsymbol{w}\|_2}$

Recap: Maximum margin classifier

- Training data: $D_{tr} = \left\{ \left(\boldsymbol{x}_n \in \mathbb{R}^D,\ \mathrm{y}_n \in \{+1, -1\} \right) \right\}_{i=1}^{N}$ (linear separable)

- $(\boldsymbol{w}, b)^{\mathrm{SVM}} = \underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}}\ \gamma(\boldsymbol{w}, b) =$

  $\underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}} \underset{(\boldsymbol{x}_n, \mathrm{y}_n)}{\min} \frac{\mathrm{y}_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|_2} = \underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}} \frac{1}{\|\boldsymbol{w}\|_2} \underset{(\boldsymbol{x}_n, \mathrm{y}_n)}{\min} \mathrm{y}_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)$

- Due to positive-scale-invariant, for $(\boldsymbol{w}, b)$ that perfectly separate data, we can
  - constrain $\underset{(\boldsymbol{x}_n, \mathrm{y}_n)}{\min} \mathrm{y}_n\left(\boldsymbol{w}^T \boldsymbol{x}_n + b\right) = 1$
  - $(\boldsymbol{w}, b)^{\mathrm{SVM}} = \underset{(\boldsymbol{w},\, b)}{\operatorname{argmax}} \frac{1}{\|\boldsymbol{w}\|_2}$, s.t. $\underset{(\boldsymbol{x}_n, \mathrm{y}_n)}{\min} \mathrm{y}_n\left(\boldsymbol{w}^T \boldsymbol{x}_n + b\right) = 1$

Recap: Maximum margin classifier

- $(\boldsymbol{w}, b)^{\text{SVM}} = \underset{(\boldsymbol{w}, b)}{\operatorname{argmax}} \frac{1}{\|\boldsymbol{w}\|_2} = \underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \|\boldsymbol{w}\|_2 = \underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \|\boldsymbol{w}\|_2^2 = \underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}$

- Optimization problem:
  - $\underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}, \qquad \text{s.t.} \min_{(\boldsymbol{x}_n, \mathrm{y}_n)} \mathrm{y}_n \left(\boldsymbol{w}^T \boldsymbol{x}_n + b \right) = 1$

- "Equivalent" optimization problem with the same solution
  - $\underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}, \qquad \text{s.t.} \ \forall n, \mathrm{y}_n \left(\boldsymbol{w}^T \boldsymbol{x}_n + b \right) \geq 1$

Recap: Soft-margin SVM

- $\underset{(\boldsymbol{w},\, b)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}, \qquad \text{s.t. } \forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1$
  - No solution if the data is not linear separable (common cases)

- Allow "slight" violation with "slack" variables

- $\underset{(\boldsymbol{w},\, b, \xi_n)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n\,, \qquad \text{s.t. } \forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1 - \xi_n, \xi_n \geq 0$
  - C is a hyper-parameter
  - How to choose it?

Recap: Slack variables

- Allow some data points to be closer to the hyperplane or classified wrongly
  - But with penalties in the objective
  - $\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n$

- Extreme cases:
  - If C is large, more like the hard-margin SVM
  - If C is small, favor simple models with errors

$$\text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$$

Hinge loss

- Original soft-margin SVM
  - $\underset{(\boldsymbol{w}, b, \xi_n)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n$, s.t. $\forall n, \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1 - \xi_n, \xi_n \geq 0$
  - Convex quadratic programming: quadratic objective + linear constraints

- Unconstrained formulation
  - $\underset{(\boldsymbol{w}, b)}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n\left[1 - \mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right)\right]_+$
  - How to solve? (stochastic) gradient descent
  - Penalize even for unconfident correct predictions

Constraint (convex) optimization

- General formulation: $\min_{\boldsymbol{w}} g(\boldsymbol{w})$, s.t. $\boldsymbol{w} \in \boldsymbol{A}$

Constraint (convex) optimization

- General formulation: $\min_{\boldsymbol{w}} g(\boldsymbol{w})$, s.t. $\boldsymbol{w} \in \boldsymbol{A}$
  - $\boldsymbol{A}$ is a set of feasible $\boldsymbol{w}$; i.e., you can only choose $\boldsymbol{w}$ from $\boldsymbol{A}$ to minimize $g(\boldsymbol{w})$
    - Example: <u>the shortest travel time from Columbus to NYC</u> <u>under $50</u>.
  - If $\boldsymbol{A}$ is empty, then the problem is infeasible
    - Example: let's say there is no way that you can travel to NYC under $50.

Recap: Hard-margin SVM

- $\underset{(\boldsymbol{w},\, b)}{\operatorname{argmin}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}, \qquad \text{s.t. } \forall n, \mathrm{y}_n \left(\boldsymbol{w}^T \boldsymbol{x}_n + b \right) \geq 1$
  - No solution if the data is not linear separable (common cases)

Constraint convex optimization

- $g(\boldsymbol{w})$ is a convex function:
  - $g(t\boldsymbol{w}_1 + (1-t)\boldsymbol{w}_2) \leq tg(\boldsymbol{w}_1) + (1-t)g(\boldsymbol{w}_2),\ \forall\, \boldsymbol{w}_1, \boldsymbol{w}_2;\ \forall t \in [0,1]$

Constraint convex optimization

- $g(\boldsymbol{w})$ is a convex function:
  - $g(t\boldsymbol{w}_1 + (1-t)\boldsymbol{w}_2) \leq tg(\boldsymbol{w}_1) + (1-t)g(\boldsymbol{w}_2),\ \forall\, \boldsymbol{w}_1, \boldsymbol{w}_2;\ \forall t \in [0,1]$

- $\boldsymbol{A}$ is a convex set:
  - If $\boldsymbol{w}_1 \in \boldsymbol{A}$ and $\boldsymbol{w}_2 \in \boldsymbol{A}$, then $t\boldsymbol{w}_1 + (1-t)\boldsymbol{w}_2 \in \boldsymbol{A}\ \forall t \in [0,1]$

Constraint convex optimization

- The optimal set $\boldsymbol{A}^*$: the set that contains the optimal solutions
  - Let $g^* = \min_{\boldsymbol{w}} g(\boldsymbol{w}),\ \text{s.t.}\ \boldsymbol{w} \in \boldsymbol{A}$
  - $\boldsymbol{A}^* = \{\boldsymbol{w} \in \boldsymbol{A}\ \ \text{AND}\ \ g(\boldsymbol{w}) = g^*\}$

- For convex optimization, the optimal set is convex:
  - That is, if $\boldsymbol{w}_1 \in \boldsymbol{A}^*$ and $\boldsymbol{w}_2 \in \boldsymbol{A}^*$, then $t\boldsymbol{w}_1 + (1-t)\boldsymbol{w}_2 \in \boldsymbol{A}^*\ \ \forall t \in [0,1]$

- Strictly convex functions:
  - $g(t\boldsymbol{w}_1 + (1-t)\boldsymbol{w}_2) < tg(\boldsymbol{w}_1) + (1-t)g(\boldsymbol{w}_2),\ \forall\, \boldsymbol{w}_1, \boldsymbol{w}_2;\ \forall t \in [0,1]$
  - If feasible, the optimization has a unique solution

Constraint convex optimization

- The constraint set ***A*** is usually represented by equalities or inequalities

Constraint convex optimization

- The constraint set ***A*** is usually represented by equalities or inequalities

$$w[1]^2 + w[2]^2 \leq 4$$

Constraint convex optimization

- The constraint set ***A*** is usually represented by equalities or inequalities

$$w[1]^2 + w[2]^2 \leq 4$$

$$2 \times w[1] + 2 \times w[2] \leq 4$$

Constraint convex optimization

- The constraint set ***A*** is usually represented by equalities or inequalities

$$w[1]^2 + w[2]^2 \leq 4$$

$$2 \times w[1] + 2 \times w[2] \leq 4$$

$$2 \times w[1] + 2 \times w[2] \geq -4$$

Constraint convex optimization

- The constraint set ***A*** is usually represented by equalities or inequalities

$$w[1]^2 + w[2]^2 \leq 4$$

$$2 \times w[1] + 2 \times w[2] \leq 4$$

$$2 \times w[1] + 2 \times w[2] \geq -4$$

$$1 \times w[1] - 1 \times w[2] = 0$$

Constrained convex optimization

- The convex constraint set $\boldsymbol{A}$ is usually represented by the "intersections of" equalities or inequalities
  - $\boldsymbol{A} = \{\boldsymbol{w} \text{ s.t. } u_i(\boldsymbol{w}) \leq 0, i = 1, \dots, M; \quad q_i(\boldsymbol{w}) = 0, i = 1, \dots, P\}$
  - $u_i(\boldsymbol{w})$ is a convex function
  - $q_i(\boldsymbol{w})$ is an affine (linear) function
  - $\boldsymbol{w} \in \boldsymbol{A}$ if it obeys all the constraints

- Example:
  - $\min_{(\boldsymbol{w}, b)} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}, \qquad \text{s.t. } -\mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) + 1 \leq 0, \forall n$
  - $\min_{(\boldsymbol{w}, b, \xi_n)} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_n \xi_n, \qquad \text{s.t. } -\mathrm{y}_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) + 1 - \xi_n \leq 0, \ -\xi_n \leq 0, \forall n$

Convex optimization

- Convex optimization: convex objective function (+ convex constrained set)
  - Local optimum = global optimum
  - The solution is independent of the initialization
  - The optimal set is convex
  - If the objective function is strictly convex, the problem has at most one optimal solution

- Example tool: https://web.stanford.edu/~boyd/software.html
  - If you can represent your convex optimization problem in the standard form, there are many sophisticated tool to help solve your problem.

Linear classifier for binary classes

- To learn $(\boldsymbol{w}, \quad b)$
- Decision rule: $\text{sign}\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)$

Linear classifier for binary classes

- To learn $(\boldsymbol{w}, \quad b)$
- Decision rule: $\text{sign}\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)$

Classification

- To separate the “feature space” into class territories

Linear classifier for multiple classes

- Linear separable?

Linear classifier for multiple classes

Treat other classes as "negative" classes

Higher ${\boldsymbol{w}_1}^T\boldsymbol{x} + b_1$ values

Lower ${\boldsymbol{w}_1}^T\boldsymbol{x} + b_1$ values

Linear classifier for multiple classes

Treat other classes as "negative" classes

Linear classifier for multiple classes

Treat other classes as "negative" classes

Linear classifier for multiple classes

Every class c has its associated $(\boldsymbol{w}_c,\ b_c)$ that aims to give data from class c higher $\boldsymbol{w}_c{}^T\boldsymbol{x} + b_c$ values

Linear classifier for multiple classes

Decision rule:

$$\hat{y} = \underset{c}{\operatorname{argmax}}\left(\boldsymbol{w}_c{}^T \boldsymbol{x} + b_c\right)$$

Multi-class learning

- Training data: $D_{tr} = \left\{ \left(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \right) \right\}_{i=1}^{N}$
  - $y \in \{1, \dots \dots, C\}$. (The integer C here is not the hyperparameter in SVM.)
  - $y = \boldsymbol{e}_c = [0, \dots \dots, 1, \dots \dots, 0]^T$, a C-dim one-hot vector with only its c-th element = 1

Multi-class learning

- Training data: $D_{tr} = \left\{ \left(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \right) \right\}_{i=1}^{N}$
  - $y \in \{1, \dots \dots, C\}$. (The integer C here is not the hyperparameter in SVM.)
  - $y = \boldsymbol{e}_c = [0, \dots \dots, 1, \dots \dots, 0]^T$, a C-dim one-hot vector with only its c-th element = 1

- Softmax regression
- One-vs-all (or one-vs-other) formulation: extension of SVM, logistic regression
- Crammer-Singer SVM
- Other variants: one-vs-one, DAG (Directed Acyclic Graph)

Multi-class learning

- Training data: $D_{tr} = \{ (\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n) \}_{i=1}^{N}$
  - $y \in \{1, \dots\dots, C\}$. (The integer C here is not the hyperparameter in SVM.)
  - $y = \boldsymbol{e}_c = [0, \dots\dots, 1, \dots\dots, 0]^T$, a C-dim one-hot vector with only its c-th element = 1

- Softmax regression
- One-vs-all (or one-vs-other) formulation: extension of SVM, logistic regression
- Crammer-Singer SVM
- Other variants: one-vs-one, DAG (Directed Acyclic Graph)

- **Cautions: several "notations" (like λ, e_c) may be used in different contexts**

Softmax regression (cf. logistic regression)

- **Training data:** $D_{tr} = \{ \left(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \{1, \dots \dots, C\} \right) \}_{i=1}^{N}$

- **Discriminative learning: to construct** $p(Y = c|\boldsymbol{x})$

Softmax regression (cf. logistic regression)

- **Training data:** $D_{tr} = \{ (\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \{1, \dots \dots, C\}) \}_{i=1}^{N}$

- **Discriminative learning: to construct** $p(Y = c|\boldsymbol{x})$
  - Conditional categorical; i.e., $p(Y = c|\boldsymbol{x}) = \lambda(\boldsymbol{x})[c]$
  - Modeling: $\lambda(\boldsymbol{x})[c] = \dfrac{e^{(\boldsymbol{w}_c{}^T \boldsymbol{x} + b_c)}}{\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T \boldsymbol{x} + b_{c'})}}$

$\boldsymbol{x}$

Softmax regression (cf. logistic regression)

- **Training data:** $D_{tr} = \{ (\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \{1, \dots \dots, C\}) \}_{i=1}^{N}$

- **Discriminative learning: to construct** $p(Y = c|\boldsymbol{x})$
  - Conditional categorical; i.e., $p(Y = c|\boldsymbol{x}) = \lambda(\boldsymbol{x})[c]$
  - Modeling: $\lambda(\boldsymbol{x})[c] = \dfrac{e^{(\boldsymbol{w}_c{}^T \boldsymbol{x} + b_c)}}{\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T \boldsymbol{x} + b_{c'})}}$
  - Properties 1: $1 \geq \lambda(\boldsymbol{x})[c] \geq 0, \sum_{c=1}^{C} \lambda(\boldsymbol{x})[c] = 1$
  - Properties 2: $\boldsymbol{w}_c{}^T \boldsymbol{x} + b_c \ \uparrow, p(Y = c|\boldsymbol{x}) \ \uparrow$
  - Properties 3: $\boldsymbol{w}_c{}^T \boldsymbol{x} + b_c$ is usually called the decision value or logit

  - Question: what if $C = 2$? We get $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ but logistic regression only has $\boldsymbol{w}$ (HW # 4)

Softmax regression

- **Learning (MLE):** $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

Softmax regression

- **Learning (MLE):** $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

$$\log \prod_{n=1}^{N} p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta}) = \sum_{n=1}^{N} \log \frac{e^{(\boldsymbol{w}_{y_n}{}^T \boldsymbol{x}_n)}}{\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T \boldsymbol{x}_n)}}$$, ignore b_c

$$= \sum_{n=1}^{N} \left[\boldsymbol{w}_{y_n}{}^T \boldsymbol{x}_n - \log \sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T \boldsymbol{x}_n)} \right]$$

Softmax regression

- **Learning (MLE):** $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr};\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

$$\log \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \frac{e^{(\boldsymbol{w}_{y_n}{}^T \boldsymbol{x}_n)}}{\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T \boldsymbol{x}_n)}}$$, ignore b_c

$$= \sum_{n=1}^{N} \left[\boldsymbol{w}_{y_n}{}^T \boldsymbol{x}_n - \log \sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T \boldsymbol{x}_n)} \right]$$

- $\{\boldsymbol{w}_c\}_{c=1}^{C}{}^{\text{MLE}} = \underset{\{\boldsymbol{w}_c\}_{c=1}^{C}}{\operatorname{argmin}} \sum_{n=1}^{N} \left[-\boldsymbol{w}_{y_n}{}^T \boldsymbol{x}_n + \log \sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T \boldsymbol{x}_n)} \right]$

Softmax regression: gradient

- Gradient for each data instance $(\boldsymbol{x}_n,\ y_n)$

$$\nabla_{\boldsymbol{w}_c}\left[-\boldsymbol{w}_{y_n}{}^T\boldsymbol{x}_n + \log\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n)}\right]$$

Softmax regression: gradient

- Gradient for each data instance $(\boldsymbol{x}_n,\ y_n)$

$$\nabla_{\boldsymbol{w}_c}\left[-\boldsymbol{w}_{y_n}{}^T\boldsymbol{x}_n + \log\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n)}\right] = -\mathbf{1}[y_n == c]\times\boldsymbol{x}_n + \frac{\nabla_{\boldsymbol{w}_c}\left[\sum_{c'=1}^{C} e^{\left(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n\right)}\right]}{\sum_{c'=1}^{C} e^{\left(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n\right)}}$$

$$= -\mathbf{1}[y_n == c]\times\boldsymbol{x}_n + \frac{\nabla_{\boldsymbol{w}_c} e^{\left(\boldsymbol{w}_c{}^T\boldsymbol{x}_n\right)}}{\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n)}} = -\mathbf{1}[y_n == c]\times\boldsymbol{x}_n + \frac{e^{\left(\boldsymbol{w}_c{}^T\boldsymbol{x}_n\right)}\boldsymbol{x}_n}{\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n)}}$$

$$= -\mathbf{1}[y_n == c]\times\boldsymbol{x}_n + p(c|\boldsymbol{x}_n;\boldsymbol{w})\boldsymbol{x}_n = -\{\mathbf{1}[y_n == c] - p(c|\boldsymbol{x}_n;\boldsymbol{w})\}\boldsymbol{x}_n$$

- Note: Every $(\boldsymbol{x}_n, y_n)$ contributes to the update of all $\boldsymbol{w}_c, \forall c \in \{1, \dots\dots, C\}$, not only to $\boldsymbol{w}_{y_n}$

Softmax regression: cross entropy loss

- **Learning (MLE):** $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

$$-\log \prod_{n=1}^{N} p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})$$

Softmax regression: cross entropy loss

- **Learning (MLE):** $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

$$-\log \prod_{n=1}^{N} p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})$$

$$= \sum_{n=1}^{N} \sum_{c=1}^{C} -\mathbf{1}[y_n == c] \times \log p(c | \boldsymbol{x}_n; \boldsymbol{\theta})$$

Softmax regression: cross entropy loss

- **Learning (MLE):** $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

$$-\log \prod_{n=1}^{N} p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})$$

$$= \sum_{n=1}^{N} \sum_{c=1}^{C} -\mathbf{1}[y_n == c] \times \log p(c | \boldsymbol{x}_n; \boldsymbol{\theta})$$

$$= \sum_{n=1}^{N} \sum_{c=1}^{C} -\boldsymbol{e}_{y_n}[c] \times \log p(c | \boldsymbol{x}_n; \boldsymbol{\theta}) = -\sum_{n=1}^{N} \boldsymbol{e}_{y_n}{}^{T} \begin{bmatrix} \log p(1 | \boldsymbol{x}_n; \boldsymbol{\theta}) \\ \vdots \\ \log p(C | \boldsymbol{x}_n; \boldsymbol{\theta}) \end{bmatrix}$$

Softmax regression: cross entropy loss

- **Learning (MLE):** $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr};\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

$$-\log \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n|\boldsymbol{x}_n;\boldsymbol{\theta})$$

$$= \sum_{n=1}^{N} \sum_{c=1}^{C} -\mathbf{1}[y_n == c] \times \log p(c|\boldsymbol{x}_n;\boldsymbol{\theta})$$

$$= \sum_{n=1}^{N} \sum_{c=1}^{C} -\underbrace{\boxed{\boldsymbol{e}_{y_n}[c]}}_{\text{target}} \times \underbrace{\boxed{\log p(c|\boldsymbol{x}_n;\boldsymbol{\theta})}}_{\text{prediction}} = -\sum_{n=1}^{N} \boldsymbol{e}_{y_n}{}^{T} \begin{bmatrix} \log p(1|\boldsymbol{x}_n;\boldsymbol{\theta}) \\ \vdots \\ \log p(C|\boldsymbol{x}_n;\boldsymbol{\theta}) \end{bmatrix}$$

Softmax regression: cross entropy loss

- **Learning (MLE):** $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(D_{tr}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_n p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})\, p(\boldsymbol{x}_n)$

$$-\log \prod_{n=1}^{N} p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta})$$

$$= \sum_{n=1}^{N} \sum_{c=1}^{C} -\mathbf{1}[y_n == c] \times \log p(c | \boldsymbol{x}_n; \boldsymbol{\theta})$$

$$= \sum_{n=1}^{N} \sum_{c=1}^{C} -\underbrace{\boxed{\boldsymbol{e}_{y_n}[c]}}_{\text{target}} \times \underbrace{\boxed{\log p(c | \boldsymbol{x}_n; \boldsymbol{\theta})}}_{\text{prediction}} = -\sum_{n=1}^{N} \boldsymbol{e}_{y_n}{}^{T} \begin{bmatrix} \log p(1 | \boldsymbol{x}_n; \boldsymbol{\theta}) \\ \vdots \\ \log p(C | \boldsymbol{x}_n; \boldsymbol{\theta}) \end{bmatrix}$$

- Re-write the objective
- Want the target and the prediction to be similar (i.e., to have high inner product)

Softmax regression: prediction

- **Prediction by Bayes optimal classifier**
  - $\hat{y} = \underset{c}{\operatorname{argmax}}\, p(c|\boldsymbol{x};\boldsymbol{\theta}) = \underset{c}{\operatorname{argmax}} \frac{e^{(\boldsymbol{w}_{\boldsymbol{c}}{}^T \boldsymbol{x})}}{\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T \boldsymbol{x})}} = \underset{c}{\operatorname{argmax}}\, e^{(\boldsymbol{w}_{\boldsymbol{c}}{}^T \boldsymbol{x})} = \underset{c}{\operatorname{argmax}}\, \boldsymbol{w}_{\boldsymbol{c}}{}^T \boldsymbol{x}$

Midterm exam and HW # 3

- Midterm
  - 10/17, in class
  - Not open book, but you can bring 5 “papers” (10 pages) of cheat sheets. Printing is fine.
  - This means, you can practice how to make concise notes for yourself!
  - More information will be announced via Carmen!

- HW # 3
  - Due (problem): 10/15
  - Due (programming): 10/22

- Homework
  - Not meant to be a practice for your exam
  - But to verify or practice some important concepts of ML

HW regrading

- TA will create Google forms about it.

CSE 5523:
Multi-class + ERM

The Ohio State University

Today

Multi-class classification

- Basics
- Modeling + Learning
- Extension

ERM + regularization

Today

Multi-class classification

- Basics
- Modeling + Learning
- Extension

ERM + regularization

Softmax regression: gradient

- Gradient for each data instance $(\boldsymbol{x}_n,\ y_n)$

$$\nabla_{\boldsymbol{w}_c}\left[-\boldsymbol{w}_{y_n}{}^T\boldsymbol{x}_n + \log\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n)}\right]$$

Softmax regression: gradient

- Gradient for each data instance $(\boldsymbol{x}_n,\ y_n)$

$$\nabla_{\boldsymbol{w}_c}\left[-\boldsymbol{w}_{y_n}{}^T\boldsymbol{x}_n + \log\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n)}\right] = -\mathbf{1}[y_n == c]\times\boldsymbol{x}_n + \frac{\nabla_{\boldsymbol{w}_c}\left[\sum_{c'=1}^{C} e^{\left(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n\right)}\right]}{\sum_{c'=1}^{C} e^{\left(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n\right)}}$$

$$= -\mathbf{1}[y_n == c]\times\boldsymbol{x}_n + \frac{\nabla_{\boldsymbol{w}_c}\ e^{\left(\boldsymbol{w}_c{}^T\boldsymbol{x}_n\right)}}{\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n)}} = -\mathbf{1}[y_n == c]\times\boldsymbol{x}_n + \frac{e^{\left(\boldsymbol{w}_c{}^T\boldsymbol{x}_n\right)}\boldsymbol{x}_n}{\sum_{c'=1}^{C} e^{(\boldsymbol{w}_{c'}{}^T\boldsymbol{x}_n)}}$$

$$= -\mathbf{1}[y_n == c]\times\boldsymbol{x}_n + p(c|\boldsymbol{x}_n;\boldsymbol{w})\boldsymbol{x}_n = -\{\mathbf{1}[y_n == c] - p(c|\boldsymbol{x}_n;\boldsymbol{w})\}\boldsymbol{x}_n$$

- Note: Every $(\boldsymbol{x}_n, y_n)$ contributes to the update of all $\boldsymbol{w}_c, \forall c \in \{1, \dots\dots, C\}$, not only to $\boldsymbol{w}_{y_n}$

One-vs-all (or one-vs-other)

- One-vs-all learns $\boldsymbol{w}_c$ by viewing data of class c as class +1, others as class -1

One-vs-all (or one-vs-other)

- One-vs-all learns $\boldsymbol{w}_c$ by viewing data of class c as class +1, others as class -1
  - $\boldsymbol{w}_1$:

  - $\boldsymbol{w}_2$:

  - $\boldsymbol{w}_3$:

One-vs-all (or one-vs-other)

- Learn each $\boldsymbol{w}_c$ independently
- For $c \in \{1, \dots \dots, C\}$
  - Temporally set $y_n = \begin{cases} +1, \text{if } y_n = c \\ -1, \text{othewise} \end{cases}$, such that $D_{tr} = \{ (\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \{+1, -1\}) \}_{i=1}^{N}$
  - $\boldsymbol{w}_c$ is the learned $\boldsymbol{w}$ from binary classification (e.g., by SVM, logistic regression)

One-vs-all (or one-vs-other)

- Learn each $\boldsymbol{w}_c$ independently
- For $c \in \{1, \dots \dots, C\}$
  - Temporally set $y_n = \begin{cases} +1, \text{if } y_n = c \\ -1, \text{othewise} \end{cases}$, such that $D_{tr} = \{ (\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \in \{+1, -1\}) \}_{i=1}^N$
  - $\boldsymbol{w}_c$ is the learned $\boldsymbol{w}$ from binary classification (e.g., by SVM, logistic regression)

- Prediction: $\underset{c}{\operatorname{argmax}}\, \boldsymbol{w}_c{}^T \boldsymbol{x}$
  - In training, we encourage $\boldsymbol{w}_c{}^T \boldsymbol{x}_n$ to be large if $y_n = c$; small if $y_n \neq c$
  - Problem: no calibration among $\boldsymbol{w}_c$

- Note: multi-class logistic regression vs. softmax regression

Crammer-singer SVM (learn $\boldsymbol{w}_c$ together)

- **Prediction:** $\underset{c}{\operatorname{argmax}}\, \boldsymbol{w}_c{}^T \boldsymbol{x}$
  - That is, if $\boldsymbol{x}$ belongs to class y, we hope that $\boldsymbol{w}_y{}^T \boldsymbol{x} > \max_{c \neq y} \boldsymbol{w}_c{}^T \boldsymbol{x}$

- **Learning:**
  - $\min \ \frac{1}{2} \sum_c \boldsymbol{w}_c{}^T \boldsymbol{w}_c + \lambda \sum_n \max\left\{0, \left(\max_{c \neq y_n} \boldsymbol{w}_c{}^T \boldsymbol{x}_n + 1\right) - \boldsymbol{w}_{y_n}{}^T \boldsymbol{x}_n\right\}$
  - $\min \ \frac{1}{2} \sum_c \boldsymbol{w}_c{}^T \boldsymbol{w}_c + \lambda \sum_n \max\left\{0, \left(\max_{c} \boldsymbol{w}_c{}^T \boldsymbol{x}_n + \mathbf{1}[y_n \neq c]\right) - \boldsymbol{w}_{y_n}{}^T \boldsymbol{x}_n\right\}$
  - The 1 or $\mathbf{1}[y_n \neq c]$ here is the enforced margin
  - Here I use λ to denote the hyper-parameter

Other variants

- **One-vs-one learning:**
  - Build $C \times (C-1)/2$ binary classifies for pairwise comparison
  - Ex: $\boldsymbol{w}_{1-2}$:

$\boldsymbol{w}_{3-2}$:

- **Prediction:**
  - Accumulate the decision values of each class, for predicting the label for $\boldsymbol{x}$
    - Ex: score(c) $= \sum_{c' \neq c} \left(\boldsymbol{w}_{c-c'}^T \boldsymbol{x} \right)$
  - Comparison along a DAG (Directed Acyclic Graph)

Comparison

- Modeling and prediction:
  - $\underset{c}{\operatorname{argmax}}\, \boldsymbol{w}_c{}^T \boldsymbol{x}$: softmax regression, Crammer-Singer, one-vs-all
  - One-vs-one$\boldsymbol{w}_{c-c'}$: $\underset{c}{\operatorname{argmax}}\ \sum_{c' \neq c} \left(\boldsymbol{w}_{c-c'}^T \boldsymbol{x} \right)$ or DAG or other accumulation rule

- Learning:
  - Learn $\{\boldsymbol{w}_c\}_{c=1}^{C}$ jointly: softmax regression, Crammer-Singer
  - Learn $\{\boldsymbol{w}_c\}_{c=1}^{C}$ or $\left\{\boldsymbol{w}_{c-c'}\right\}_{c=1,c'=1}^{C,C}$ independently: one-vs-all, one-vs-one

- Problems for one-vs-all and one-vs-one:
  - Limited or imbalanced data (too many -1 data than +1 data)

Summary

- Modeling
  - **Decision values:** Linear, nonlinear, kernelized, nearest neighbor, …
    - $\boldsymbol{w}_c{}^T \boldsymbol{x} + b_c$
    - $\boldsymbol{w}_c{}^T \boldsymbol{\phi}(\boldsymbol{x})$
    - $\sum_n \alpha_{c,n} k_{\boldsymbol{\phi}}(\boldsymbol{x}_n, \boldsymbol{x})$
  - **Parametric vs. nonparametric:** Does the number of parameters grow with training data?

- Learning
  - Empirical risk minimization (ERM)
  - Structured risk minimization, regularized risk/loss minimization
  - Probabilistic: MLE, MAP

Today

Multi-class classification

- Basics
- Modeling + Learning
- Extension

ERM + regularization

- Warm up
- Basic

Empirical risk minimization

- **Goal:** find h: $\mathcal{X} \mapsto \mathcal{Y}$ to minimize $E_{(x,y)\sim P}[\ell_{te}(h(x), y)]$, where P is unknown
- What do we know about P?
  - $D_{tr} = \left\{ \left(\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n \right) \right\}_{i=1}^{N}$ that are sampled IID from P

Empirical risk minimization

- **Goal:** find h: $\mathcal{X} \mapsto \mathcal{Y}$ to minimize $E_{(x,y)\sim P}[\ell_{te}(h(x), y)]$, where P is unknown
- What do we know about P?
  - $D_{tr} = \{ (\boldsymbol{x}_n \in \mathbb{R}^D,\ y_n) \}_{i=1}^{N}$ that are sampled IID from P

- **Empirical risk minimization (regularized risk/loss minimization)**
  - $\hat{h} = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \ \boxed{\frac{1}{N}\sum_{n=1}^{N} \ell_{tr}(h(\boldsymbol{x}_n), y_n)} + \boxed{\lambda \times r(h)}$

    Training loss Regularization

  - ℓ_{tr}: (continuous) function that penalizes training error
  - r: (continuous) function that penalizes classifier/hypothesis complexity
  - λ: hyper-parameter

Empirical risk minimization: example

- Linear soft SVM:
  - $y \in \{+1, -1\}, \quad h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathrm{sign}\left(\boldsymbol{w}^T \boldsymbol{x}_n\right)$
  - $\underset{\boldsymbol{w}}{\mathrm{argmin}} \; \boxed{\frac{1}{N}\sum_n \left[1 - \mathrm{y}_n\left(\boldsymbol{w}^T \boldsymbol{x}_n\right)\right]_+} + \boxed{\lambda \|\boldsymbol{w}\|_2^2}$

    Hinge-loss L2-regularization
  - For homework implementation, please look at **"HW3.pdf"**

Empirical risk minimization: example

- Linear soft SVM:
  - $y \in \{+1, -1\}, \quad h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathrm{sign}\left(\boldsymbol{w}^T \boldsymbol{x}_n\right)$
  - $\underset{\boldsymbol{w}}{\mathrm{argmin}} \underbrace{\boxed{\frac{1}{N}\sum_n \left[1 - \mathrm{y}_n\left(\boldsymbol{w}^T \boldsymbol{x}_n\right)\right]_+}}_{\text{Hinge-loss}} + \underbrace{\boxed{\lambda \|\boldsymbol{w}\|_2^2}}_{\text{L2-regularization}}$
  - For homework implementation, please look at **"HW3.pdf"**
- Any other kinds of (surrogate) loss functions and regularization functions?
  - If so, what are the principles or requirements?