1 Intro to ML

1.1 ML Pipeline

1.1.1 Definition of Models

- Model can refer to equations (linear/non-linear), code/rules, Neural networks, decision trees, bayesian graph, Deep NN
- In simple terms, model defines a function that maps inputs to outputs and is mostly mathematical equations

1.1.2 Machine Learning Pipeline

Machine learning follows an iterative pipeline in the following order:

Data collection $\longrightarrow$ Data Extraction $\longrightarrow$ Data pre-processing $\longrightarrow$ Model choice/design $\longrightarrow$ Model training $\longrightarrow$ Model validation (evaluation) $\longrightarrow$ Model understanding (explainability) $\longrightarrow$ Model deployment

2 Paradigms of ML

2.1 Classes of ML techniques

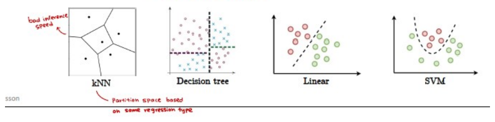
1. **Supervised learning**
   - Given a set of data features and labels, design an algorithm that predicts label for new data (never observed before).
   - Regression task: e.g. predict price (continuous value) of houses given existing data label (house size)
   - Classification task (binary): e.g. predict (discrete value) whether benign or malignant tumor based on tumor size
   - Classification task (multiple classes): e.g. predict class of disease given size of tumor
   - Classification task (multi-feature)
2. **Unsupervised learning**
   - In supervised learning, data and label always comes in a pair but in unsupervised learning, it is likely that we do not have label or any additional information
   - The goal is to find structure in data that solves tasks like clustering, classification, compression and generative model
3. **Reinforcement learning** (not tested)
   - Learns a sequence of actions that maximizes a reward
   - RL agents learns to plan the future to win
   - Main issue is that lots of data is required because reward is sparse (e.g. whole chess game must be played to get a score of +1, -1 or 0)
4. **Self-supervised learning** (not tested)

2.2 Supervised Learning with kNN

- Goal is to estimate a function f such that $y = f_\theta(x)$
  - x is a raw data point
  - y is the label (either a real number for regression task or categorical variable for classification task)
  - θ are the parameters of the predictive function
- y is annotated by humans which is extremely time consuming, expensive, biased and is limited to human knowledge

2.2.1 Difference between kNN, Decision tree, SVM

- What differentiates these three models?* ⟶ typically is how their feature space is partitioned
  - Geometry of decision function/boundary
  - Learnable vs non-learnable models ⟶ for param θ
    - Param
    - discrete hyperparam
    - using gradient optimisation
  - Complexity (speed and memory) of training and inference
    - ⟶ inference vs training speed



2.2.2 k-Nearest-Neighbor

- Assumes that close data points have similar labels
- Basic Algorithm to predict the label of a new data point x*:
  1. Find the closest k data points in the training set, $S = \{(x_i, y_i)_{i=1:N}\}$
  2. Assign the label of the **mean/mode** to the new data point
- Distance metric:
  1. L2/Euclidean distance
  2. L1/Manhatten distance
  3. Cosine distance (angle)
  4. Jaccard distance (sets)
- Larger the value of k, the more smooth the decision boundary, less overfitting
- Hyper-parameter k acts as regularizer which increases robustness (new & similar input used, output is same) of predictor
- Terrible at high-dimensionality since data points sampled from random distribution will have about the same distance from each other (curse of dimensionality)
- Performs well for few number of meaningful features
- Can be used for both regression and classification tasks

2.2.3 Time Complexity of kNN

- No training needed
- O($n \cdot d$) for nearest neighbor and O($n \cdot d \cdot k$) for kNN
  - n = number of training data points
  - d = number of data features
  - k = number of nearest neighbor
- High memory usage as all training data is loaded into memory
- k-d tree and hashing techniques to speed up

2.2.4 Curse of Dimensionality vs Blessing of Structure

- Curse of dimensionality states that if points were **chosen randomly**, distance between them are about equal $\longrightarrow$ kNN will not work well
- Blessing of structure instead says that **real world data is unlikely to be randomly distributed** and have structures like edges or structures (for images)
  - This brings down dimensionality of data to be $< R^d$

2.2.5 Pros and Cons of kNN

- Extremely simple and expressive (can produce non-linear boundary decision)
- As $n \to \infty$, kNN is provably very accurate but requires huge amount of space
- As $d \to \infty$, kNN fails due to curse of dimensionality

3 Decision Trees

3.1 Motivation of kd-tree

- kNN is extremely slow and time consuming when we use a large n since we **need to compute distance to all of the points** each time we make a query
- Want a way to **prune away points** that we know are extremely far away to cut computation time

3.2 How kd-tree works

3.2.1 Tree Construction

1. Split recursively in half along each feature dimension
   - We want to have about the same data points at each side of the split for max efficiency
   - Which direction we choose to split first is chosen greedily
2. Iterate over all feature dimensions
3. End up with a depth of $O(\log_2 n)$

* In general, number of partition is about 2^d to capture largest variation of data

3.3 Searching k-d tree

1. Given a point x, identify which set x lies in and find the closest neighbor in that set, x_{NN}
2. Compute distance $d(x, C)$, where C is the cut
3. If $d(x, C) > d(x, x_{NN})$, discard everything that is in the set of the cut, else find nearest neighbor in the other set of the cut
4. Repeat for all cuts

3.4 Complexity of k-d tree

- Space complexity of k-d tree: $O(2^p) \to O(n)$, where $p = O(\log_2 n)$
- Time complexity to build k-d tree: $O(n \log_2 n)$
- Inference speed:
  - Best case: $O(\log_2 n + d)$
  - Worst case: $O(dn)$, basically nearest neighbor
  - Average case: $O(d \log n)$

3.5 Pros and cons of k-d tree

- Pros:
  - Exact kNN but no need to backtrack in parent nodes
  - Easy to implement
  - Average inference much faster than kNN, $O(d \log_2 n)$ vs $O(dn)$
- Cons:
  - Cuts are axis aligned $\longrightarrow$ not good in high dimensionality

3.6 Decision trees

- kNN requires full training data to make prediction
- k-d tree uses the fact that data concentrate in regions to speed things up
- Ultimately, goal is not to find closest data points but to get a classification or regression value
  - e.g. If new data point falls in cluster of 1000 +ve points, can just classify as +ve without calculating distance
  - Want to leverage the idea of clustering so that **no need to load full training set**

3.6.1 Inference using Decision Tree

- Once tree constructed, don't need to keep training set in memory
- Only need to store **tree structure** ($O(\log_2 n)$ depth) as well as **class probability/regression value/class label** in leaf nodes
- Inference is extremely fast at $O(\log_2 n)$, independent of d

3.6.2 Optimal decision tree

- In theory if no 2 data points have same features but different labels, we can ensure min depth
- In practice, finding minimum size tree is NP-hard
- Compromise is to **minimize a function that measures label purity**

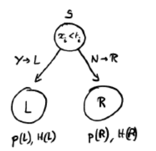
3.6.3 Purity

- Purity is defined as the fraction of data with label k in a set, $p_k = \dfrac{|S_k|}{|S|}$
- Worst case is when all leaves are random prediction, i.e. $p_k = \dfrac{1}{c}, \forall k$, where c is the number of classes
- Want to maximize Kullback-Leiber distance between random prediction and best candidate p

3.6.4 Entropy

- Maximizing KL distance is equivalent to minimizing entropy
- If probability of each class is equal, then entropy is at its maximum

$H(S)$



$$H(L, R) = p(L)H(L) + p(R)H(R)$$
$$= \frac{|L|}{|S|}H(L) + \frac{|R|}{|S|}H(R)$$

- Entropy $H = -\sum p_k \log p_k$
- Ideally, entropy of L and R should be less than S, i.e. $H(L), H(R) < H(S)$

3.6.5 Information Gain

- Information gain (IG) is the difference between the entropy of the original set S and the weighted sum of the entropy of the subset S_k

$$IG(S, S_1, \cdots, S_c) = H(S) - H(S_1, \cdots, S_c) = H(S) - \sum_{k=1}^{c} p(S_k)H(S_k)$$

3.6.6 Construction of Decision Tree

- Goal is to find subsets that maximizes the information gain, achieving the purest possible subsets.
  - Identifying the purest subsets is to find a feature x_i and a threshold value t_i.
- Decision tree construction (pseudo-code)

```
While leaf nodes are not pure (or ≥ threshold)
    Loop over (remaining) feature dimensions, i.e. $x_1, x_2, \dots, x_d$
        Loop over n thresholds (e.g. middle points between two consecutive points, such as $t_i = (x_{i+1} - x_i)/2$)
            Compute information gain for R and L
            Save (dimension, threshold value) with maximum information gain.
    Split space with best (dimension, threshold value) and remove dimension $x_i$ from loop.
```

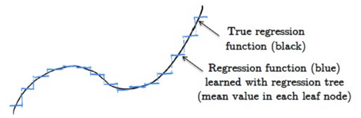
- Exact complexity is O(n.d). Approximations are used in practice for speed-up.

3.7 Regression tree

- It is **easy to extend decision trees to regression trees** as long as a purity function can be defined for the new task

$$\mathrm{H}(S) = -\sum p_k \log p_k \quad \to \quad \mathrm{L}(S) = \frac{1}{|S|} \sum_{(x,y) \in S} (y - \bar{y}_S)^2 \ \text{(variance)}$$
$$\text{with } \bar{y}_S = \frac{1}{|S|} \sum_{(x,y) \in S} y \ \text{(mean)}$$

Classification task — Regression task



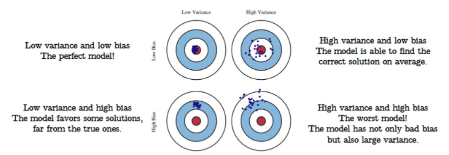
3.7.1 Complexity of Regression Tree

- Regression tree space, time and inference complexity all follows that of k-d tree

3.8 Bagging

- Decision trees are extremely fast but have high variance, i.e. weak learners
- **High variance**: quality of the classification/regression solutions vary significantly

3.8.1 Bias and Variance

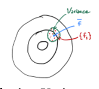

Low Variance
High Variance
Low Bias
High Bias
Low variance and low bias
The perfect model!
High variance and low bias
The model is able to find the correct solution on average.
Low variance and high bias
The model favors some solutions, far from the true ones.
High variance and high bias
The worst model!
The model has not only bad bias but also large variance.

- Formalization

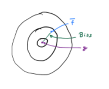
$$\underbrace{\mathbb{E}_{(x,y)}}_{\text{Mean over data points}}\left[(\underbrace{f_S}_{\text{Predictive model}}(\overset{\text{data}}{x}) - \underbrace{y}_{\text{target}})^2\right]$$

Assumption: no noise

$$= \mathbb{E}_{(x,y)}\left[(f_S(x) - \underbrace{\bar{f}(x)}_{\text{Average predictor}} + \bar{f}(x) - y)^2\right], \text{ with } \bar{f}(x) = \int_{S' \subset S^*} f_{S'}(x) \underbrace{p(S')}_{\text{True data distribution}} dS'$$



$$= \underbrace{\mathbb{E}_{(x,y)}\left[(f_S(x) - \bar{f}(x))^2\right]}_{\substack{\text{Error between prediction model}\\ \text{and average over all predictors}\\ \text{Variance}}} + \underbrace{\mathbb{E}_{(x,y)}\left[(\bar{f}(x) - y)^2\right]}_{\substack{\text{Error between average}\\ \text{predictor and target}\\ \text{Bias}^2}}$$



3.8.2 Reducing Variance

- Most common way to reduce variance is to take average of several solution, i.e. ensembling
- Relies on the **law of large numbers** where the average of many small predictors will tend to the average of the true predictor
- However this requires us to have access to more training sets which in most cases we do not have

3.8.3 Bagging algorithm

1. Sample m datasets $S_1, \dots, S_m$ from original S **with replacement**
2. For each training set S_j, train a classifier f_{S_j}
3. Final ensemble classifier is $\hat{f}(x) = \frac{1}{m} \sum_{j=1}^{m} f_{S_j}(x)$

* Note that sampling with replacement breaks the assumption of independent and identically distributed (i.i.d) data and therefore does not have **theoretical guarantee** that the ensemble will give a good estimation of the true mean actual predictor

3.8.4 Advantages of bagging

- In practice, bagging reduces variance quite effectively, but after a large number m, will have diminishing returns
- Bagging can **reduce the variance without increasing the error** of an unbiased classifier

3.8.5 Random forests

- One of the most popular and useful bagging algorithms
- Random forest is an ensemble of decision trees
- Hyper-parameters: $k = \sqrt{d}$

3.8.6 Boosting

- Boosting is used to help reduce bias on weak learners such as decision trees with limited depth
- Uses the idea of ensembling which combines large number of weak learners to generate a strong learner with low bias
- Note that both boosting and bagging are both agnostic to algorithm used

4 Linear Models

4.1 Applications of Linear Models

1. **Classification**: sign($\theta^T x$), $y = \pm 1$, e.g. Approve or reject
2. **Regression**: $\theta^T x$, $y \in \mathbb{R}$, e.g. Amount of credit
3. **Logistic regression**: sigmoid($\theta^T x$), $y \in [0, 1]$, e.g. probability of defaulting

4.2 Equation of Linear Model

- $f_\theta(x) = \theta^T x = \sum_{i=1}^{d} \theta_i x_i$, where θ_i is the weight/importance of feature x_i
- Linear classification: $\begin{cases} \theta^T x > 0 \Rightarrow +1 \\ \theta^T x < 0 \Rightarrow -1 \end{cases}$
- Linear regression: $\theta^T x \Rightarrow$ Amount (scalar)

4.3 Loss function

$$\mathrm{L}(\theta) = \frac{1}{n} \sum_{j=1}^{n} (f_\theta(x^j) - y^j)^2 \text{ – Mean squared error}$$

where $f_\theta(x^j)$ is the predicted value from the model and y^j is the actual labeled value

- Goal of models is to choose a θ that **minimizes** the mean squared error

4.4 Finding best θ

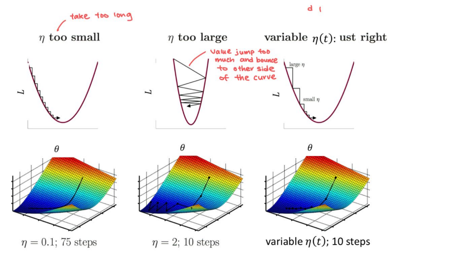
- Mainly uses 2 approaches: **Normal equations** and **Gradient descent**
- Normal equation solution (only for linear regression):

$$\theta = \left((X^T X)^{-1} X^T\right) y$$

where $(X^T X)^{-1} X^T$ is known as the pseudo-inverse of X and is **computationally expensive to compute if n or m is large**

4.4.1 Gradient Descent

- Works in high-dimensional spaces, convergence independent of data dimensionality d
- Steps to carry out gradient descent
  1. Start at a random $\theta(t)$
  2. At each step, update the value of parameter: $\theta(t+1) = \theta(t) + v$, where v is the opposite direction of the greatest +ve gradient
  3. Stop when gradient < arbitrary threshold
- $v = -\eta \dfrac{\partial L}{\partial \theta}(\theta(t))$, where η is the step size
  - No good way to find the right step size, just have to try a few values and pick best one



4.4.2 Stochastic Gradient Descent

- Main problem with gradient descent is that there are **lots of saddle points / flat regions in high dimensional spaces**

Stochastic gradient descent

- A fast variation of gradient descent that considers only a small set of data points to update the value of the parameters is stochastic gradient descent (SGD):
  - Pick a (random) small subset of m training data (e.g. a single/512 data points), i.e. $(x^{(k)}, y^{(k)})$
  - Compute the loss value for this subset, i.e. $\mathrm{L} = \frac{1}{m}\sum_{k=1}^{m} \mathrm{L}(f_\theta(x^{(k)}), y^{(k)})$
  - Compute the gradient for this subset, i.e. $v = \frac{1}{m}\sum_{k=1}^{m} -\nabla \mathrm{L}\,(f_\theta(x^{(k)}), y^{(k)})$
  - Update parameters, i.e. $\theta(t+1) = \theta(t) - \eta\, v$
- Advantages
  - Faster update of parameters : Gradient computed with m=512 rather than all data points.
  - Stochastic optimization : Helps escape saddle points in high-dimensional space
  - Simple to implement

4.4.3 Gradient Descent vs Normal Equation

- Two methods for solving the linear regression task :
  - Gradient Descent
    - Works well for very large n, #training data, with SGD (300B tokens with GPT3)
    - Works well for very large d, #data features (d=1M with 1,000×1,000 images)
    - Works well for very large $|\theta|$, #parameters features (175B with GPT3)
    - Very slow and requires to select time step η
  - Normal Equations
    - Very fast for n=O(10^6) data points, do not require to choose η
    - Need to compute $(\mathbf{X}^T\mathbf{X})^{-1}$, O($n^3$) operation but faster approximations exist
- Gradient Descent is a universal optimization technique as long as the considered loss is continuous and differentiable (as gradient is required).
  - GD does not work for discrete losses like win/lose at the game of Go.

4.5 Logistic Regression

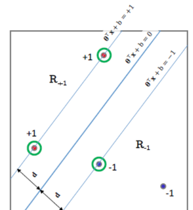
- Used to **predict probability** e.g. probability of heart attack, given training data of person's health information and whether they have a heart attack
- $P_\theta(y|x) = \begin{cases} f_\theta(x) & \text{for y = +1} \\ 1 - f_\theta(x) & \text{for y = -1} \end{cases}$
- Goal is to minimize cross-entropy loss (a.k.a. log loss):

$$min_\theta L(\theta) = \frac{1}{n} \sum_{j=1}^{n} \log(1 + \exp(-y^j \theta^T x^j))$$

where n is the number of training data

4.6 Support Vector Machine (SVM)

- Goal of SVM is to find a linear separator that partitions the feature space into 2 regions



- We want margin d to be as **large as possible** to generalize to more data points
- $d = \dfrac{2}{\|\theta\|}$

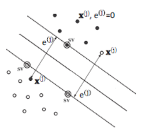
4.6.1 Soft-margin SVM
- Aims to solve the problem that real-world data is typically **non-linearly separable** due to outliers $\longrightarrow$ no mathematical solution to SVM
- Idea is to **introduce a slack variable** $e(j)$ for each data point that represents the prediction error

$$\min_\theta \|\theta\|^2 \text{ such that } \begin{cases} f_\theta(\mathbf{x}) \geq +1 \text{ for } \mathbf{x} \in R_{+1} \\ f_\theta(\mathbf{x}) \leq -1 \text{ for } \mathbf{x} \in R_{-1} \end{cases} \quad \textbf{Standard SVM}$$

$$\Downarrow$$

$$\min_{\theta,e} \|\theta\|^2 + C.\sum_{j=1}^n e^{(j)} \text{ such that } \begin{cases} f_\theta(\mathbf{x}^{(j)}) \geq +1 - e^{(j)} \text{ for } \mathbf{x} \in R_{+1} \\ f_\theta(\mathbf{x}^{(j)}) \leq -1 + e^{(j)} \text{ for } \mathbf{x} \in R_{-1} \\ e^{(j)} \geq 0,\ C \geq 0 \end{cases}$$

Soft-margin SVM

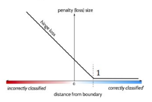


- Soft-margin **penalizes misclassifications and correct classifications that fall within margin**
- Typically would have a **much smaller margin** than standard SVM
- Uses hinge loss function:

$$\min_{\theta,e} \|\theta\|^2 + C.\sum_{j=1}^n e^{(j)} \text{ such that } \begin{cases} f_\theta(\mathbf{x}^{(j)}) \geq +1 - e^{(j)} \text{ for } \mathbf{x} \in R_{+1} \\ f_\theta(\mathbf{x}^{(j)}) \leq -1 + e^{(j)} \text{ for } \mathbf{x} \in R_{-1} \\ e^{(j)} \geq 0,\ C \geq 0 \end{cases}$$

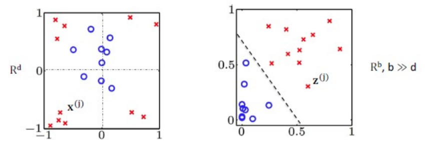
$$\Updownarrow$$

$$\min_\theta \|\theta\|^2 + C.\sum_{j=1}^n \underbrace{\max\left(0, 1 - y^{(j)} f_\theta(\mathbf{x}^{(j)})\right)}_{\text{Hinge loss}}$$



4.7 Kernel SVM
- Linear models are limited to **linearly separable data points** $\longrightarrow$ cannot handle complex/non-linear data sets
- To use a linear kernel, one must first map data from original space R^d to a higher dimensional space $R^b, b >> d$ so that the classes can be distinguished using linear functions

$$\mathbf{x} = (x_1, x_2, ..., x_n) \xrightarrow{\Phi} \mathbf{z} = (z_1, z_2, ..., z_n)$$



5 Bias & Variance

5.1 Definition of Bias, Variance and Noise
- Bias: **intrinsic** error/difference between the average prediction model and the true regression value. Inherent to the model and independent of data. Also known as **model error**

$$Bias(\mathcal{H}) = \mathbb{E}(\mathcal{H}) - Y$$

- Variance: captures how much the learner changes when it is computed on different training sets. Indicates **expressivity**$\longrightarrow$ complex algo have high expressivity

$$Var(\mathcal{H}) = \mathbb{E}[f - \mathbb{E}(\mathcal{H})^2], \text{where} f \in \mathcal{H}$$

- Noise: ambiguity inherent to the data distribution and feature representation. Impossible to get rid of and is considered as a stochastic process that is added to "clean" data
- Good hypothesis has low bias and low variance
  - To compare between 2 models, just add their bias and variance and compare the sum, lower the better

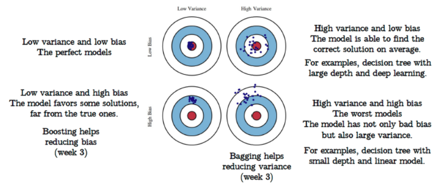
5.2 Bias-Variance-Error Decomposition

$$\underbrace{\mathbb{E}_{x,y,D}\left[(f_D(x) - y)^2\right]}_{\text{Expected test error}} = \underbrace{\mathbb{E}_{x,D}\left[(f_D(x) - \bar{f}(x))^2\right]}_{\text{Variance}} + \underbrace{\mathbb{E}_x\left[(\bar{f}(x) - \bar{y}(x))^2\right]}_{\text{Bias}^2} + \underbrace{\mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]}_{\text{Data noise}}$$

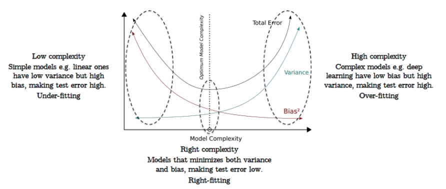
- Most fundamental equation of supervised learning known as the bias-variance-error trade off

5.3 Over/Under Fitting
- Over-fitting: Occurs when learner $f_D(x)$ has zero prediction error on a training set D, but have large test error
  - Learner is too expressive and will become over-specialized of the training data, unable to extrapolate to unseen data.
  - Typically have high variance to capture complexity of whole dataset
- Under-fitting: learner $f_D(x)$ which is not able to predict correctly a training set D (very unlikely to happen), have both **high training and test error**
  - Learner is not expressive enough. It will make error on the provided training set, i.e. unable to benefit from all information present in the training data
  - Typically have no enough variance/complexity $\longrightarrow$ just using a simple algorithm

5.4 Bias-Variance Trade-off


Low Variance
High Variance
Low Bias
High Bias
Low variance and low bias
The perfect models
High variance and low bias
The model is able to find the correct solution on average.
For examples, decision tree with large depth and deep learning.
Low variance and high bias
The model favors some solutions, far from the true ones.
Boosting helps reducing bias (week 3)
High variance and high bias
The worst models
The model has not only bad bias but also large variance.
For examples, decision tree with small depth and linear model.
Bagging helps reducing variance (week 3)

- Test error vs. model complexity



5.5 How to reduce variance and bias
- How to reduce high variance?
  - Reduce model complexity : Lower model expressivity or use regularizers (week 6)
  - Remove non-informative/bad data features : E.g. house features such house color, back door, etc but challenging to decide bad features (not done in practice).
  - Bagging (week 3) : Averaging weak high-variance learners produces strong learner with low variance (requires fast computation of weak learners in practice).
- How to reduce high bias?
  - Increase model complexity : More expressive models (deep learning)
  - Add more informative data features : E.g. house features such as storage space, garden, security system, etc (effective but time and money consuming).
  - Boosting (week 3) : Adding weak high-bias learners produces strong learner with low bias (requires fast computation of weak learners in practice s.a. small-depth decision trees).
- Add more training data or use data augmentation to reduce both bias and variance.

6 Overfitting Regularization

6.1 Loss Regularization
- Method to **reduce overfitting** by decreasing expressivity of model (e.g. from 10th order to 2nd order function)
- Typically use Regularization with stochastic gradient descent $\longrightarrow$ SGD speeds up gradient descent and also regularizes predictive function (w.r.t. θ), allowing for better generalization
- Uses the concept of **constraint optimization with soft constraints**, basically saying that we are setting the $\theta_3, \theta_4, \cdots, \theta_n$ of a hypothesis space $\mathcal{H} = \theta_1 + \theta_2 x + \theta_3 x^2 + \cdots + \theta_n x^{n-1}$ to $\leq C$

$$\min_\theta L_\theta \text{ such that } \sum_{j=0}^d \theta_j^2 = \theta^T \theta \leq C, C > 0 \tag{1}$$

- If C is small $\rightarrow \theta_{j \geq 3} \approx 0$, i.e. $\mathcal{H}_{10} = \mathcal{H}_2$
- If C is large, then basically doing unconstrained optimization (MSE)

6.2 Relationship between constraint regularized problem and unconstrained optimization problem
- For every general constraint regularized problem like in (1), there exist and equivalent unconstrained optimization problem (**easier to solve**)

$$min_\theta L(\theta) + \lambda\theta^T\theta, \lambda > 0 \tag{2}$$

- For each value C, there exists a value λ such that (1) $\equiv$ (2) (Lagrange multiplier), $C \propto \frac{1}{\lambda}$

6.3 Original MSE loss vs Regularized MSE

Original MSE loss :

$$\min_\theta L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{j=1}^n(\boldsymbol{\theta}^T\mathbf{x}^{(j)} - y^{(j)})^2 = \frac{1}{n}\|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|^2$$

Set gradient of loss to zero :

$$\nabla L = \frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{0} \Rightarrow \mathbf{X}^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) = \mathbf{0} \Rightarrow \boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

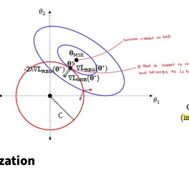
Regularized MSE loss :

$$\min_\theta L(\boldsymbol{\theta}) = \frac{1}{n}\|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|^2 \text{ s.t. } \boldsymbol{\theta}^T\boldsymbol{\theta} \leq C$$
$$\Leftrightarrow \min_\theta L(\boldsymbol{\theta}) = \frac{1}{n}\|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|^2 + \lambda\, \boldsymbol{\theta}^T\boldsymbol{\theta}$$

Set gradient of loss to zero :

$$\nabla L = \frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{0} \Rightarrow \frac{1}{n}\mathbf{X}^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) + \lambda\boldsymbol{\theta} = \mathbf{0} \Rightarrow \boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X} + \lambda n\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

- Minimizer of the total loss : $\boldsymbol{\theta}^* = \text{argmin}_\theta\, L_{MSE}(\boldsymbol{\theta}) \text{ s.t. } L_{REG}(\boldsymbol{\theta}) \leq C$ $\Leftrightarrow \boldsymbol{\theta}^* = \text{argmin}_\theta\, L_{MSE}(\boldsymbol{\theta}) + \lambda\, L_{REG}(\boldsymbol{\theta})$


θ_2
θ_1
θ_{MSE}
θ^*
$-2\lambda\nabla L_{REG}(\theta^*)$
$\nabla L_{REG}(\theta^*)$
$\nabla L_{MSE}(\theta^*)$
C
solution closest to MSE
θ that is closest to MSE and belongs to L2 ball

Solution $\boldsymbol{\theta}^*$ is as close as the MSE solution $\boldsymbol{\theta}_{MSE}$ as allowed by the L2-ball constraint $\|\boldsymbol{\theta}\|^2 \leq C$.

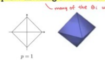
Gradient of loss at $\boldsymbol{\theta}^*$:
$\nabla(L_{MSE}(\boldsymbol{\theta}^*) + \lambda\, L_{REG}(\boldsymbol{\theta}^*)) = 0$
$\Rightarrow \nabla L_{MSE}(\boldsymbol{\theta}^*) = -\lambda\nabla L_{REG}(\boldsymbol{\theta}^*)$
Gradients of L_{MSE} and L_{REG} are aligned (in the opposite direction) at the solution.

6.4 L2 vs L1 regularization
- The L_2-ball regularization can be generalized to L_p-ball, $p \in [0, +\infty]$.

$$\min_\theta L_{MSE}(\boldsymbol{\theta}) \text{ s.t. } \|\boldsymbol{\theta}\|_p \leq C \Leftrightarrow \min_\theta L_{MSE}(\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}\|_p \text{ where } \|\boldsymbol{\theta}\|_p = \left(\sum_{i=1}^d|\theta_i|^p\right)^{\frac{1}{p}}$$

- L_2-ball/L_2 regularization, a.k.a. weight decay
  - Advantages : Strictly convex, differentiable, fast optimization, robust w.r.t. perturbation.
  - Limitations : Although θ_i values are minimized, solutions are dense, i.e. $\theta_i > 0$. This means no feature selection in e.g. $f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \cdots + \theta_{10}x^{10}$, as all data features are used for prediction. will never completely ignore features even if they are not impt

- L_1 regularization
  - Advantages : Convex (but not strictly), fast optimization algorithms exist, robust w.r.t. perturbation, solutions are guaranteed to be sparse meaning feature selection, as only a few data features are used for prediction. many of the θi will be 0
  - Limitations : Not differentiable at the origin. no gradient but easily solvable in practise



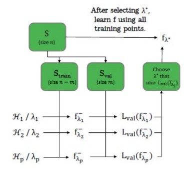
6.5 Cross Validation
- We need a surrogate of the test set to estimate the regularization parameter λ which identifies the right model complexity that minimizes the test error
- Since we cannot touch data from the test set, we **split training set into training and validation set**
- Use p hypothesis/values to estimate λ :

We consider p values $\lambda_1, ..., \lambda_p$.

Use S_{train} to learn $f^-_{\lambda_j}$ for each λ value.

Evaluate $f^-_{\lambda_j}$ using S_{val} : $L_{val}\left(f^-_{\lambda_j}\right)$ for $j = 1, ..., p$

Select value $\lambda^* = \lambda_j$ with smallest L_{val}

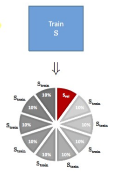

After selecting λ*, learn f using all training points.
S (size n)
f_{λ^*}
S_{train} (size n − m)
S_{val} (size m)
Choose λ* that min $L_{val}(f^-_{\lambda_j})$
$\mathcal{H}_1 / \lambda_1$
$\mathcal{H}_2 / \lambda_2$
$\mathcal{H}_p / \lambda_p$
$f^-_{\lambda_1}$
$f^-_{\lambda_2}$
$f^-_{\lambda_p}$
$L_{val}(f^-_{\lambda_1})$
$L_{val}(f^-_{\lambda_2})$
$L_{val}(f^-_{\lambda_p})$

- Estimation of error of validation set $L_{val} = L_{test} \pm O(\frac{1}{\sqrt{m}})$, where m is the number of data points in validation set
  - To have L_{val} be as close to L_{test}, we want m to be as large as possible so that error of validation step is a good representation of the actual test set
  - Typically m is 20% of training set

6.5.1 Effects of m
- We have 2 opposite cases
  - Recall : Model f is trained on the full training set of n data, and f^- is trained on the $n - m$ training set
  - Case #1 : Small number m of validation data / large number $n - m$ of training data
    - Advantage : $L_{test}(f) \approx L_{test}(f^-)$ as f^- is well estimated.
    - Limitation : $L_{test}(f^-) \neq L_{val}(f^-)$ as the validation set is too small.
  - Case #2 : Large number m of validation data / small number $n - m$ of training data
    - Advantage : $L_{test}(f^-) \approx L_{val}(f^-)$ as the validation set is large enough.
    - Limitation : $L_{test}(f) \neq L_{test}(f^-)$ as f^- is badly estimated.
- How to reconcile the two cases?

Train
n=90
Validation, m=10
Test
Train, n=10
Validation
m=90
Test

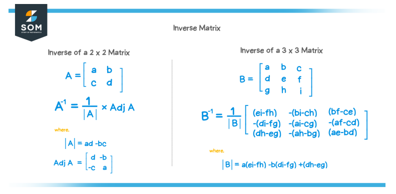
6.5.2 k-fold Cross Validation
- k-fold cross-validation technique :
  - Split the original training set into k parts, i.e. each fold has n/k data points.
  - Repeat for all folds : Train on k-1 parts and leave one part out as validation set.
- Advantages
  - Each data in the original training set will be used as a validation data.
  - For each fold, we have a large training set to train a good learner, i.e. $L_{test}(f) \approx L_{test}(f^-)$.
  - We also have a good estimate of the validation error by averaging the validation error over all folds, i.e. $L_{test} \approx \text{mean}_{folds} L_{val}(f^-)$.



6.5.3 Early Stopping
- Cross validation extremely computationally expensive
- To speed things up, we will stop optimization after M number of gradient steps, **when the validation error starts increasing**, even if optimization has not converged yet

7 Miscellaneous

7.1 Matrix Inverse



Inverse Matrix

Inverse of a 2 x 2 Matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad A^{-1} = \frac{1}{|A|} \times \text{Adj } A$$

where,

$$|A| = ad - bc \qquad \text{Adj } A = \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Inverse of a 3 x 3 Matrix

$$B = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$B^{-1} = \frac{1}{|B|}\begin{bmatrix} (ei-fh) & -(bi-ch) & (bf-ce) \\ -(di-fg) & -(ai-cg) & -(af-cd) \\ (dh-eg) & -(ah-bg) & (ae-bd) \end{bmatrix}$$

where,

$$|B| = a(ei-fh) - b(di-fg) + (dh-eg)$$

7.2 Matrix Dot Product

$$A \bullet B = \begin{array}{c} \vec{r_1} \rightarrow \\ \vec{r_2} \rightarrow \\ \vec{r_3} \rightarrow \end{array} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \bullet \begin{array}{ccc} \vec{c_1} & \vec{c_2} & \vec{c_3} \\ \downarrow & \downarrow & \downarrow \\ \end{array} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 6 \\ 7 & 2 & 5 \end{bmatrix} = \begin{bmatrix} 26 & 16 & \vec{r_1} \bullet \vec{c_3} \\ \vec{r_2} \bullet \vec{c_1} & \vec{r_2} \bullet \vec{c_2} & \vec{r_2} \bullet \vec{c_3} \\ \vec{r_3} \bullet \vec{c_1} & \vec{r_3} \bullet \vec{c_2} & \vec{r_3} \bullet \vec{c_3} \end{bmatrix}$$

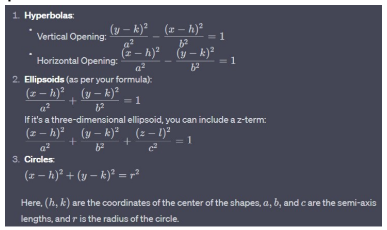
$$= \begin{bmatrix} 26 & 16 & 1 \bullet 1 + 2 \bullet 6 + 3 \bullet 5 \\ 4 \bullet 1 + 5 \bullet 2 + 6 \bullet 7 & 4 \bullet 2 + 5 \bullet 4 + 6 \bullet 2 & 4 \bullet 1 + 5 \bullet 6 + 6 \bullet 5 \\ 7 \bullet 1 + 8 \bullet 2 + 9 \bullet 7 & 7 \bullet 2 + 8 \bullet 4 + 9 \bullet 2 & 7 \bullet 1 + 8 \bullet 6 + 9 \bullet 5 \end{bmatrix}$$

$$= \begin{bmatrix} 26 & 16 & 28 \\ 56 & 40 & 64 \\ 86 & 64 & 100 \end{bmatrix}$$

7.3 Differentiation Nonsense
1. $\frac{d}{dx}(x) = 1$
2. $\frac{d}{dx}(ax) = a$
3. $\frac{d}{dx}(x^n) = nx^{n-1}$
4. $\frac{d}{dx}(\cos x) = -\sin x$
5. $\frac{d}{dx}(\sin x) = \cos x$
6. $\frac{d}{dx}(\tan x) = \sec^2 x$
7. $\frac{d}{dx}(\cot x) = -\csc^2 x$
8. $\frac{d}{dx}(\sec x) = \sec x \tan x$
9. $\frac{d}{dx}(\csc x) = -\csc x(\cot x)$
10. $\frac{d}{dx}(\ln x) = \frac{1}{x}$
11. $\frac{d}{dx}(e^x) = e^x$
12. $\frac{d}{dx}(a^x) = (\ln a)a^x$
13. $\frac{d}{dx}(\sin^{-1} x) = \frac{1}{\sqrt{1-x^2}}$
14. $\frac{d}{dx}(\tan^{-1} x) = \frac{1}{1+x^2}$
15. $\frac{d}{dx}(\sec^{-1} x) = \frac{1}{|x|\sqrt{x^2-1}}$

- [Chain Rule] $\frac{d}{dx}f(g(x)) = f'(g(x))g'(x)$
- [Product Rule] $\frac{d}{dx}(f(x)g(x)) = f(x)g'(x) + f'(x)g(x)$
- [Quotient Rule] $\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{g(x)f'(x) - f(x)g'(x)}{g(x)^2}$

7.4 Useful Equations



1. **Hyperbolas**:
   - Vertical Opening: $\frac{(y-k)^2}{a^2} - \frac{(x-h)^2}{b^2} = 1$
   - Horizontal Opening: $\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$
2. **Ellipsoids** (as per your formula):
   $\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$
   If it's a three-dimensional ellipsoid, you can include a z-term:
   $\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} + \frac{(z-l)^2}{c^2} = 1$
3. **Circles**:
   $(x-h)^2 + (y-k)^2 = r^2$

Here, (h, k) are the coordinates of the center of the shapes, a, b, and c are the semi-axis lengths, and r is the radius of the circle.

7.5 When is ML not useful?
1. the task is fully deterministic
2. there's a straightforward formula equating x and y
3. the task demands different outputs every time, regardless of inputs (randomness)

7.6 kNN for Regression task

2.1 k-NN for Regression

k-NNs can be used for regression as well as classification. With a new observation x_{test}, calculate the mean of the k nearest neighbours and assign it as the prediction for this new observation.

$$y_{\text{test}} = \frac{1}{k}\sum_{i=1}^{k} x_i \qquad x_i \in S$$

where S is the sorted distances between all $x \in X$.

7.7 Logistic Regression as Binary Classification

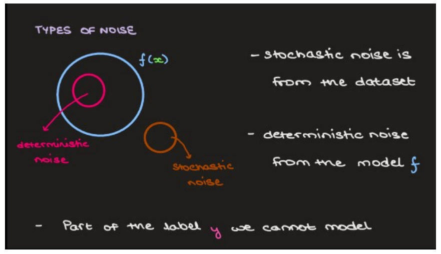
While Linear Regression is for regression, Logistic Regression is for classification – mostly problems with 2 classes (i.e., binary classification). Logistic Regression is similar to Linear Regression in that we compute $\hat{y} = w^T \cdot x + b$. However, for Logistic Regression, $\hat{y}$ is a real number and not a classification. To make the decision of classifying x as class 0 or 1, we use the Sigmoid Function:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

It takes any number and squashes it into the range $[0, 1]$. If the final Sigmoid value is ≤ 0.5, we classify it as class 0, and it's ≥ 0.5, we classify it as class 1:

$$\hat{y} = \begin{cases} 0 & \sigma(z) < 0.5 \\ 1 & \sigma(z) \geq 0.5 \end{cases}$$

7.8 Types of noise


TYPES OF NOISE
f(x)
- stochastic noise is from the dataset
- deterministic noise from the model f
deterministic noise
stochastic noise
- Part of the label y we cannot model