Supervised Learning: Classification, Regression
Unsupervised: Clustering, Recco System

1) Dimension Reduction using PCA/LDA

PCA: PC (nm) = Actual Data (nm) x Eigenvectors of Cov(X) (mm)

- Eigenvector = unit vector of the PC
- Eigenvalue = SS distances/variation of data captured
- If plot = scree plot
- PC1 highest eigenvalue = highest VAR captured
- Loses interpretability, but a huge reduction in data

LDA: New axis are created by $max\frac{\sum d_i^2}{\sum s_i^2}$
Maximise the distance of each category centroid to the overall centroid.
If 2 category it is just the differences between the 2 means
while minimising variation within each category.

- LD1 does the best, LD2 second best, ortho to each other

2) Regularization
β: coefficients of the features. In NN, they will be weights
These are penalty terms added to the loss function scaled by λ
L1 reg $\|\beta\|_1$: Manhattan norm, $\sum|\beta_i|$
L2 reg $\|\beta\|_2^2$: Elucidean norm squared, $\sum\beta_i^2$

LASSO: **L1** driven by the number of params
Drives unnecessary ones to 0. Multiple solutions

Ridge Regression: **L2** reg driven by the size of the params
Drives unnecessary ones to near 0, unique solution

Net Elastic: **L1** + **L2** reg

Confusion Matrix (Of Classification Model):
Precision: TP / (TP+FP)
TPR/ Recall/ Sensitivity : TP/ (TP+FN)
TNR/ Specificity: TN/ (TN+FP)
FNR/ Type II: FN/ (FN+TP)
FPR/ Type I: FP/ (TN+FP)
Accuracy: (TP+TN)/ (All)
F1 Score: 2*Prec*Recall/(Prec+Recall)
Harmonic mean, if any is low, F1 will be low

AUC Curve:
TPR, Recall/ Sensitivity against **FPR, 1 - Specificity**,
- To determine best probability cut-off (threshold) for a given model, find the Pareto optimal value
- When comparing models' AUCs, the higher the AUC the better the model
- Increase t, lower TPR & FPR

If minority is +ve, very easy to cover all +ve actual and produce good results, AUC is bad for imbalanced data with few TP.

AUPRC: Places **emphasis on +ve minority cases**

If multiclass: agg of the metrics for each class
e.g. mAP in obj detection, cross-entropy loss etc

Variance: Variability of the model's prediction, the mean squared deviation of the predicted points, **E(pred^2) - E(pred)^2**

Bias: Difference between the average prediction of our model and the correct value, **E(pred) - actual**

Err(x) = Var(X) + Bias(X)^2 + Var(e), Irreducible Error

- The **more complex** a model is:

- **Train Err decreases** as model fits data better
- **Test Err decreases then increases** once overfitting begins
- **Variance increases** as model fits data better
- **Bias decreases and plateaues** once overfitting begins
- **Irreducible error never changes** since independent on model, only dependent on data

Underfitting: Increase size model, train more, reduce reg
Overfitting: Add more data, increase reg, dropout, early stopping, and feature selection to reduce model complexity/size

Discrete Variables Handling E.g. Categorical (Red, Blue, Green)

- **1) 1 hot encoding:**
  No. of features = No. of classes - 1
  (2 0/1 columns needed minimally)
- **2) Label encoding:**
  Each class given a index number (1, 2, 3)
- **3) Target encoding:**
  Use the target label column calculate weighted mean for each category (e.g. 0.37, 0.29, 0.57)
  Using target encoding can result in data leakage, as we are using our targets to affect the predictors, resulting in overfitting
- **4) k-fold Target encoding:**
  Split the data into folds, for each fold use the rest of the data for their target encoding.

VALIDATION:

1. **Validation set approach: (but underfit)**
   Split data evenly and roughly
2. **Leave one out Cross v: (but overfit)**
   Leave out 1 obs, train on rest
3. **k fold cross v: (balanced data)**
   Divide data in k folds, iterate leave out 1 to test, train on rest, average the error
4. **Stratitfied k fold cross v: (unbalanced data)**
   Split the data in k folds with similar distribution in each fold

Clustering
1. K-means Clustering
Puts the data into the specified k numebr of clusters

- First selects k random central points.
- Then assign each observation to nearest central point
- Then calculate mean of each cluster as new central point, then repeat until cluster no change
- Then find the variation within the clusters
- Repeat the entire process with different starting points and return the one with best results overall

How select k: Elbow method to find k just before the SS distances does not reduce much

2. Hierarchical Clustering
Finds pairwise what two observation are most similar. If need k clusters, cut the graph at where number of nodes = number of clusters

3. K-Nearest Neighbour for Classification

- Given a data point and k neighbours.
- k nearest points are selected using a specified distace measurement.
- Category with highest votes, will be used to classify the data point

Approx. Nearest Neighbour Variation
Searches only using a subset of candidate points variation of KNN, using a specified algorithm

In fact for ELK semantic search they use ANN with **Hierarchical Navigable Small World (HNSW)**

4. Support Vector Machines:
Cross v. finds the best soft margin
- **Soft margin classifier** = **Support vector classifier**
- **Support** = Point on the **Support vector hyperplane**
- Points within soft margin are missclassified

But with many misclassification (Non-linear data), we can apply a non-linear transformation (**higher dimension space)** to a feature, then find a **Support vector classifier**. However it is computationally extensive.

We extend to use **Support vector machines**
- In the **new higher dimension vector space**, we find the relationship **(dot product) between each pair of points** which is the **kernel function** and use that relationship value to determine the **Support vector classifier** (when brought down to current dimension space, is a **non-linear hyperplane**)
- Note we did not do any transformation/calculation of the data to any higher dimension

(CART) DECISION TREE: Partition predictor space

a) Regression Tree (Predict numeric data)
A leaf's prediction value is given by the mean of it's response values

Recursive binary splitting: Consider all predictors, then for each predictor, attempt all possible cuts (pairwise means between data points). Selects predictor with cut that has the lowest RSS

b) Classification Tree (Classification)
A leaf's prediction is given by the most commonly occurring class of it's values
Uses a measure of **impurity** instead of RSS

Similar recursive splitting, selecting predictor that has lowest impurity

- **Gini Impurity of Leaf: [0:0.5]**
  $= 1 - \sum_{k=\{classes\}}(p_k^2)$
- Total Gini of a split = Weighted average of the Gini Impurities of it's leaves
  Weight = fraction of observations/residuals captured
- For cont. features, calculate the avg value for each sorted adjacent value pairs
  Calculate the gini impurity for each pair to determine best split
- **Entropy: [0:1]**
  $-\sum_{k=\{classes\}} p_k log(p_k)$
- Taken from concept of surprise: $log(1/prob(x))$
  High probability = Low Surprise

GOOD: Good interpretability and visual, handles continuous variables
BAD: Not robust and accurate. Builds complex tree with **high variance** if no pruning

Cost Complexity Pruning/ Weakest Link Pruning:

Instead of building a single CART

$$\min(\text{tree score}) = \min(\text{Total RSS/impurity} + \alpha|T, \text{No. of terminal nodes}|)$$

- Using kfold cv, for given fold, produce subtrees with a range of **α (complex. param)**
- Select the best subtree for each fold, then pick the value of **α** with the lowest validation RSS/impurity

Ensemble Learning: Combining weak learners

1) Bagging (Random Forests)
Reduces overfitting, decreases variance

Generate B different **bootstrapped** training datasets (repeatedly sample observation from original **WITH** replacement until same as original data sample size) to build a tree individually. During tree building, at each split a random subset of predictors is being used (to ensure decorrelation)

For each observation, find all trees where it was an OOB samples to make predictions, the final prediction is a majority voting (classification)/ average prediction (regression). For each observation, the error is the proportion of misclassification/ MSE.

The final OOB error is the average error across all observations.

BUT if there are few strong predictors, most bagged trees will use similar predictors at the top of the split, trees are still highly correlated, hence variance is not reduced much.

ALSO not interpretable.

2) Boosting
Reduces underfitting, increases variance

Generate m subsets of training data random **WITHOUT** replacement**,** where each subsequent training data also includes the misclassified data points. Hence this minimizes errros made from previous iterations.

XGBoost

- For each binary recursive split of a predictor space, starting from 0.5
- Calculate **similarity scores** for all nodes, with regularization parameter λ
  Reg: $\frac{\sum(\text{residuals/obs})^2}{\sum(1)+\lambda} = \frac{\sum(\text{residuals/obs})^2}{\text{No. of residuals}+\lambda}$
  Class: $\frac{\sum(\text{residuals})^2}{\sum[\text{prev prob}_i \cdot (1-\text{prev prob}_i)]+\lambda}$
  where the term to the left of λ is called the **cover, sum of hessians**
- **Calculate gain** to determine how to split the data
  $\text{Left}_{simi} + \text{Right}_{simi} - \text{Root}_{simi}$
- Higher the **gain** the better the split is at splitting residuals into a group of similar values
- If $\text{Gain} \leq \gamma$, **tree complexity parameter**, then prune it
  γ = minimum reduction in loss/impurity or gain in similarity scores

- Gradient boosting concept (Reduces underfitting)

- Formalized as a GDA over an objective function
- **Simple example after mathematical operations**
- For cont. target label, use the mean as starting point
- Find the $\text{pseudo residuals} = \text{actual weight} - \text{starting point}$
- Build 1st tree to classify the pseudo residuals
  $\text{Leaf value} = \text{Average of pseudo residuals}$
- $\text{Prediction} = \text{Starting point} + \text{Leaf Value} \times \text{Learning rate}$
- With new pseudo residuals, repeat and each iteration the pseudo residuals should decrease

- Cache awareness access (Memory efficient)

- **Gradients and Hessians** are cached to output similarity scores faster

- Parallel tree building (Speed efficient)

- Utilizes **weighted quantile sketch**, which splits the large data & calculates the quantile in parallel to give an approximate quantile representation of data. Each bin will have same **sum of weights/sum of hessians**
- Uses it for each iteration, to find the best split among all features that results in largest gain in loss objective function
- Hence uses **approximate greedy tree learning** instead of level wise expansion at each split
- If dataset is small, just defaults to **greedy tree learning**, very fast

- Regularization (Reduces overfitting)

- **L1/L2 reg** on the leaf scores
- **Learning rate** on GDA
- **Gamma in loss function** for pruning

- Handles missing + sparse data (Imperfect data)

- Utilizes **sparsity-aware split finding** by splitting the dataset into missing and non-missing values. During gain calculation, it places the non-missing values to the left and the right and selects option with higher gain

Softmax: $p_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$
Argmax: $\text{argmax}(p) = \underset{i}{\text{argmax}}\ p_i$

Common Activation Function

- **Sigmoid:** Squashes real numbers to range between [0,1]
  $\frac{1}{1+e^{-x}}$
- **Tanh:** Squashes real numbers to range between [-1,1]
  $tanh(x)$
- **Rectified Linear (ReLU):** 0 when x < 0 and then linear with slope 1 when x > 0
  $max(0, x)$
- **Leaky ReLU:** 0 or have a small negative slope (of 0.01, or so) when x < 0, then linear with slope 1 when x > 0
  $max(0.1x, x)$

Types of Loss Functions and Applications:
1. Mean Squared Error (MSE):
- $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Regression problems.
- Penalizes larger errors more than smaller ones, making it sensitive to outliers. Assumes Gaussian error distribution.

2. Mean Absolute Error (MAE):
- $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- Regression problems.
- Gives equal weight to all errors, robust to outliers. Assumes Laplace error distribution.

3. Huber Loss:
- $L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \leq \delta, \\ \delta(|a| - \frac{1}{2}\delta) & \text{if } |a| > \delta. \end{cases}$
- Regression with both small errors and outliers.
- Combines MSE (small errors) and MAE (large errors) characteristics.

4. Categorical Cross-Entropy (CCE):
- $\text{CCE} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$
- Multi-class classification problems.
- Measures how well a probability distribution aligns with true class labels.

5. Binary Cross-Entropy (BCE):
- $\text{BCE} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$
- Binary classification problems.
- Optimized for binary outcomes and probability prediction.

6. Hinge Loss:
- $\text{Hinge} = \sum_{i=1} \max(0, 1 - y_i\hat{y}_i)$
- Binary classification (e.g., SVMs).
- Encourages a decision boundary with a margin of at least 1.

7. Kullback-Leibler (KL) Divergence:
- $D_{KL}(P\|Q) = \sum_i P(i)\log\left(\frac{P(i)}{Q(i)}\right)$
- Comparing probability distributions.
- Measures divergence between two probability distributions.

Data Imputation Techniques
1. Mean Imputation:
- Continuous data without extreme outliers.
- Simple and effective for symmetric distributions.

2. Median Imputation:
- Continuous data with outliers.
- Robust to extreme values compared to mean.

3. Mode Imputation:
- Categorical data.
- Ensures valid category imputation.

4. K-Nearest Neighbors (KNN) Imputation:
- Both continuous and categorical data with patterns.
- Estimates values based on the similarity to other observations.

5. Interpolation:
- Time series or sequential data.
- Uses linear, spline, or polynomial methods to estimate missing values from neighbors.

6. Regression Imputation:
- When to Use: When relationships between variables are well-understood.
- Why: Predicts missing values using other features via regression models.

7. Deep Learning-based Imputation:
- When to Use: Large datasets with complex missing patterns.
- Why: Leverages models like autoencoders or GANs to learn intricate relationships for better imputation.

Transformers

1. Input Embeddings + Positional Encoding

Inputs are tokenized into a vector space resulting in an embedding matrix (which captures the semantic meaning)

E.g. Input sentence "Hi how are you?" with word tokenization into input vector [1, 44, 23, 24] gets mapped onto an embedding matrix to [[0.213, 0.32, 0.34], [0.423, 0.234. 0.456], [0.69, 0.954, 0.43]] of dimension 3.

Positional encoding is added to the embedding to maintain the word order information and its relevance.

2. Self Attention
LLMs assign attention weights to different words based on their relevance to the current word being processed.

Multiheaded Attention:
The Attention module splits its **Query**, **Key**, and **Value** parameters N-ways and passes each split independently through a separate Head of linear layers.

Note that there is still a single Q, K, and V matrix; each head just takes a logical section of the matrices.

Suppose for i_{ith} token, we multiply it by a weight matrix, W_q to produce a query vector q_{th} vector. Intuitively the query vector is a question about the token like "Are there any adjectives before me?"

We do the same for the key matrix, which is like the token answering a question. As a result, we will have a matrix which is the dot product of every key and query vector. If the dot product is very high, it implies the key answers the query well (attends very well).

Then to prevent exploding values, and for the attention to act as weights, we apply softmax and divide by a numerical stability factor $\sqrt{d_k}$ which is the size of the key-query space.

If masking is required to ensure future tokens are not seen by previous tokens, we will need to assign the sub-diagonal values to negative infinity, such that softmax is 0.

Next, we need to update the embeddings of the tokens to cause a change in the embedding values of other tokens. For example the token: fluffy should influence the token: creature to "move"/transform the vector of the creature towards a place in the embedding space that better represents a fluffy creature. This is done using the value matrix.

$$\text{Attention} = \text{softmax}\left(\frac{QK^T}{\text{Numerical stability factor}, \sqrt{d_k}}\right)V$$

3. Feed Forward NN to Single Linear Layer:
Then it passes through a simple NN (stores facts from pertaining), before compressing to a final single layer.

For e.g. The input mentions Lebron James, but no where in the context mentions basketball, however the LLM manages to predict he is a basketball legend.

4. Softmax
Then to get the final output, we have a softmax layer to convert the linear layer into a probability distribution on potential subsequent words or phrases, considering a series of input words

- **Greedy vs Random(weighted) sampling:**
  - Select highest probability
  - Select using random-weighted strategy across the probabilities of token
- **top-k:**Select an output from the top-k results after applying the random weighted sampling
- **top-p:** Select an output from the top results with cumulative probability <=p
- **temperature:** The hotter the temperature, the flatter the probability distribution

Quantization: Summary

	Bits	Exponent	Fraction	Memory needed to store one value
FP32	32	8	23	4 bytes
FP16	16	5	10	2 bytes
BFLOAT16	16	8	7	2 bytes
INT8	8	-/-	7	1 byte

- Reduce required memory to store and train models
- Projects original 32-bit floating point numbers into lower precision spaces
- Quantization-aware training (QAT) learns the quantization scaling factors during training
- BFLOAT16 is a popular choice

FLAN T5

Encoder (auto-encoder, masked LM): *BERT, ROBERTA*
- Encodes the inputs with contextual understanding (bi-directional) and produces 1 vector per input token
- Sentiment analysis, NER

Decoder (auto-regressive/ casual LM): *GPT, Bloom*
- Accepts input tokens, to generate and predict new tokens (uni-directional, hence the term casual, where before affects the future)
- Text generation

Encoder-Decoder (Seq2Seq): *T5 Flan, Bart*
- Reconstruct a span of tokens
- Translation, Text Summarization, Q&A

Prompt Engineering
- 0/Few Shot/Dynamic Inference inside the system prompt

Finetuning
- Datasets of Q&A pairs
- But if on 1 task, will lead to catastrophic forgetting and reduction in ability in other tasks

- Overcome using **multi-task instruction fine-tuning**
- Many different system prompts, but requires many examples for training. E.g. FLAN model family

LLM Evaluation Metrics:
1. ROUGE:
- Used for text summarization
- Compares output to golden reference answer

ROGUE-1 Recall: unigram matches/unigram in ref
ROGUE-1 Precision: unigram matches/unigram in output
ROGUE-2: Bigrams
ROGUE-L: Numerator uses No. Longest Common Subsequence, Denominator uses unigram

Clipping: Unique matches only, else if match the same words more will boost ROGUE

2. BLEU Score
- Used for text translation
- Avg precision across a range of n-gram sizes

3. Other metrics
- For example, for speech-to-text, there is a **Word Error Rate**

Benchmarks:
Frameworks of tasks and metrics:

- **SuperGLUE** benchmark
- Holistic Evaluation of Language Models (**HELM**)
  - 7 metrics: Accuracy, Calibration, Robustness,
  - Fairness, Bias, Toxicity, Efficiency

Parameter Efficient Finetuning (PEFT)
Full fine-tuning for each task creates multiple copies of the large LLM models, not efficient

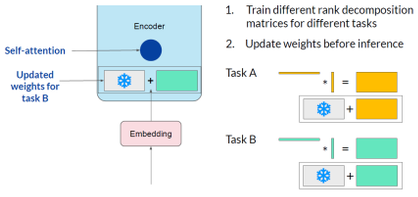
1) Low Rank Adaption of LLMs (LORA):
- Q is quantized version
- Reparameterization

2) Prompt Tuning:
- Addictive

3)Prefix Tuning
Adding embedding layers to every layer that are trainable.

4) IA3 (Infused Adapter by Inhibiting and Amplifying Inner Activations)
- Employs learned vectors to rescale inner activations in a transformer-based model that are integrated into attention and feedforward modules.

This reduces the number of trainable parameters significantly compared to approaches like LoRA.



Out-of-vocabulary (OOV) tokens:

BERT (Bidirectional Encoder Representations from Transformers):
- Uses subword tokenization, specifically WordPiece tokenization, which allows it to represent OOV words by breaking them down into smaller subword units.
- If an OOV word is "unrecognizable", BERT might tokenize it into ["un", "##recognizable"] where "##" denotes a continuation of a subword token.

GloVe (Global Vectors for Word Representation):
- Typically assign a zero vector to OOV tokens.

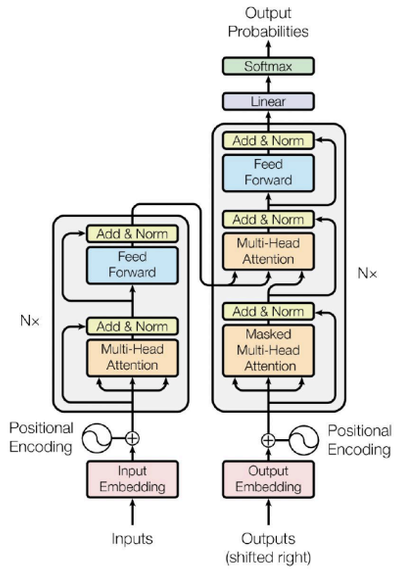
Word2Vec:
- Often assigns a special token like <unk> to represent OOV tokens.

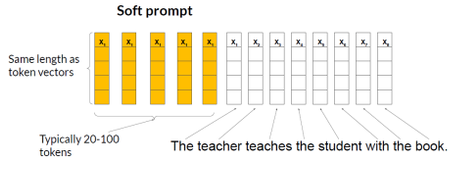
FastText:
- Uses subword tokenization, specifically the n-gram approach, which can handle OOV words by breaking them down into character-level n-grams.
- If an OOV word is "onomatopoeia", might represent it as ["ono", "nom", "oma", "top", "ope", "ia"].

ELMo (Embeddings from Language Models):
- Uses character-level embeddings as part of its modeling approach, which allows it to handle OOV words by constructing embeddings based on character-level information.
- If an OOV word is "onomatopoeia", ELMo might construct its embedding based on characters "o", "n", "o", "m", "a", "t", "o", "p", "o", "e", "i", "a".

Transformer-based models (e.g. GPT):
Handle OOV tokens in various ways, such as using a special token like <unk>, or by assigning a random initialization vector.

Each of these models has its way of dealing with OOV





STATISTICS: t-test if population variance not known, else z-test

The choice of 1 or 2-sided test depends on the **PROBLEM, NOT the sample data**

i) z test: Sample from a normal distribution.
If the sample size is large, CLT ensures normality (else use a normal QQ plot to determine)

ii) t test: Sample deviation is known, n-1 degree of freedom

- **Type I Error**: Reject H_{null} when its true in reality, α
- **Type II Error**: Fail to reject H_{null} when it's false in reality, β (1-β = Power)
- **p-value**: Strength of evidence against H_{null}
  Probability of observing the value of the test statistic given H_{null} is true
  Reject H_{null} if **p-value <= alpha**
- **CI:** Range of plausible values for the **parameter** based on the sample data such that we do not reject the H_{null}

AB TESTING

1. Problem Statement:
Question the interviewer on the problem

2. Hypothesis Testing:
State null and alternative hypothesis

Success metrics must be **measurable**, **sensitive** and **timely**

H: appines, how satisfied are users
E: gangement, how engaged are they in the product
A: cqusition, how many new users
R: etention, what are daily active users
T: ask Success, how long to complete a task in the product

3. State values (To determine sample size)

- **Alpha:** 0.05, P(type I error|given ...): Threshold for rejecting H_{null} given H_{null} is true
- **Power:** 0.80, Sensitivity of the test to detect an effect exists given H_{alt} is true
- **MDE:** 1-5% lift/ prac sig, min effect size (diff in parameters) as detection of a difference between control and treatment group

Lower alpha (more stringent), stronger evidence required, **larger sample size**

To meet a **greater power**, **larger sample size** needed to meet the increased sensitivity

Greater MDE implies lower minimum effect size, hence **smaller sample size** needed

p-value not significant, but there must be an effect, so increase power to increase sensitivity

4. Run the experiment

Duration of experiment: Based on the domain

Ensure the pipeline runs through the entire duration and avoid peeking at the p values, for statistical integrity. As some might stop at desired p-value, each check is technically another test. Hence the probability of finding a false positive increase.

5. Validity Checks:
Ensure no bias
- From instrument effect
- External factors (e.g. holiday, competitors)
- Selection Bias (Determined using A/A test)
- Sample ratio mismatch (Determined using chi-squared test)
- Novelty effects (Initial excitement or curiosity from the treatment group, hence maybe segment new from old users)

6. Interpret and Launch
Talk with business users, about cost vs other metric tradeoffs

Process of Designing Red Flags:

- **Data Analysis:**
  Analyze historical transactional data to identify patterns and potential risk scenarios.
- **Risk Identification:**
  Identify specific risk scenarios that may indicate compliance issues.
- **Hypothesis Generation:**
  Formulate hypotheses on red flags that can detect these risk scenarios.
- **Implementation:**
  Implement the red flags into the monitoring system.
- **Testing and Validation:**
  Conduct rigorous testing and validation, including A/B testing, to evaluate red flag performance.
- **Iteration and Optimization:**
  Refine and optimize the red flags based on test results and feedback.

How to improve 100k data mining

1. Data Cleaning:

- Detect and handle outliers
  (E.g. data outside 1.5 times IQR range, using Box whisker plot)
- Utilize data validation
  (E.g. sweetviz automated EDA in python, Data governance platforms like Collibra with data attributes)
- Checking back with the business user

2. Feature Selection:

- Feature importance techniques built into models like Gradient boosting
- Using regularized logistic regression (LASSO, L1)
- Using PCA

3. Sampling:

- Stratified sampling, to ensure balanced representation from different groups
- Cluster sampling or even bootstrapping (bagging) sampling

4. Parallel Processing:

- Utilize distributed computing frameworks such as Apache Spark, pyspark wrapper in Python which has inbuilt parallelization
- Store as HDFS
  - **Scalability**:
    HDFS can handle large volumes of data and easily accommodate dataset growth without performance degradation.
  - **Fault tolerance**:
    HDFS replicates data across multiple nodes, ensuring data availability and reducing the risk of data loss.
  - **Data locality**:
    HDFS stores data in a distributed manner, minimizing data transfer over the network and improving performance by maximizing data locality.
  - **Integration with big data ecosystem**:
    HDFS seamlessly integrates with other big data tools and frameworks, allowing for the use of scalable and distributed data mining algorithms.

5. Tools

- Utilize workflow management tools like Apache Airflow, MLFlow, Luigi to automate, schedule processes in batches
- Even workspaces like CDSW
- Utilize proper data pipeline frameworks like Kedro for instance

6. Documentation and Reproducibility

Engagement Metrics:	**Revenue**	**User Experience Metrics:**
- Click-through Rate (CTR)	- Return on Investment	- User Satisfaction
- Engagement Rate	- Customer Lifetime Value	- Response Time
- App Usage Frequency	- Incremental Revenue	- Personalization Effectiveness
- App Session Duration		- Opt-out Rate
- Time-on-Page	**App Store Ratings**	- Message Deliverability
- Message Interactivity	- App Rating Improvement	- Device Type Effectiveness
- In-App Purchase Rate	- Customer Acquisition Cost	- Personal Data Consent Rate
- Social Media Mentions		- Notification Opt-in Rate
	Retention Metrics:	
Conversion Metrics:	- Retention Rate	**Response Metrics:**
- Conversion Rate	- Churn Rate	- Response Rate
	- Uninstall Rate	- Time-to-Action
	- Subscriber Growth Rate	- Geographic Targeting Efn.