Date: __/__/20__ Day: _____

HANDWRITTEN NOTES FOR MACHINE LERNING

By

Mirza Yasir Abdullah Baig

Data Preprocessing

- ML Workflow
- Data Cleaning
- Data Preprocessing in Python
- Feature Scaling
- Feature Extraction
- Feature Engineering
- Feature Selection Techniques

Exploratory Data Analysis

- Exploratory Data Analysis
- Exploratory Data Analysis in Python
- Advance EDA
- Time Series Data Visualization

Model Evaluation

- Regularization in Machine learning
- Confusion Matrix
- Precision, Recall & F1-Score
- AUC-ROC Curve
- Cross-Validation
- Hyperparameter Tuning

Date: __/__/20__ Day: _____

Module: 2 Supervised learning

Linear Regression

- Introduction to Linear Regression
- Gradient Descent in Linear Regression
- Multiple Linear Regression

Logistic Regression

- Understanding Logistic Regression
- Cost function in Logistic Regression

Decision Trees

- Decision Tree in Machine Learning
- Types of Decision tree algorithms
- Decision Tree- Regression (implementation)
- Decision Tree - Classification (implementation)

Support Vector Machines (SVM)

- Understanding SVMs
- SVM Hyperparameter Tuning - GridSearch CV
- Non-Linear SVM

K-Nearest Neighbors (K-NN)

- Introduction to KNN
- Decision Boundaries in K-Nearest Neighbors (KNN)

Naive Bayes

- Introduction to Naive Bayes
- Gaussian Naive Bayes
- Multinominal Naive Bayes
- Complement Naive Bayes

Random Forest (Bagging Algorithm)

- Introduction to Random forest
- Random forest Classifier
- Random Forest Regression
- Hyperparameter Tuning in Random Forest

Date: __/__/__ Day: _____

Introduction to Ensemble Learning

- Bagging
- Boosting

Module 3: Unsupervised Learning

Clustering

Centroid - Based Methods

- K-Means clustering
- Elbow Method for optimal value of K in KMeans
- K-Means ++ clustering
- K-Mode clustering
- Fuzzy C-Means (FCM) Clustering

Distribution-Based Methods

- Gaussian mixture models
- Expectation-Maximization Algorithm
- Dirichlet process mixture models (DPMMs)

Connectivity Based Methods

- Hierarchical clustering
- Agglomerative Clustering
- Divisive clustering
- Affinity propagation

Density Based Methods

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
- OPTICS (Ordering Points To Identify the clustering Structure)

Dimensionality Reduction

- Principle Component Analysis (PCA)
- t-distributioned Stochastic Neighbor Embedding (t-SNE)
- Non-negative Matrix Factorization (NMF)
- Independent Component Analysis (ICA)
- Isomap
- Locally Linear Embedding (LLE)

Date: __/__/20__ Day: _____

Association Rule

- Apriori algorithm
- Implementing apriori algorithm
- FP-Growth (Frequent Pattern - Growth)
- ECLAT (Equivalence Class Clustering and bottom-up Lattice Traversal)

Module 4: Reinforcement Learning

Model Based Methods

- Markov decision processes (MDPs)
- Bellman equation
- Value iteration algorithm
- Monte Carlo Tree Search

Model Free Methods

- Q-Learning
- SARSA
- Monte Carlo Methods
- Reinforce Algorithm
- Actor-Critic Algorithm
- Asynchronous Advantage Actor-Critic (A3C)

Module 5: Semi Supervised Learning

- Semi Supervised Classification
- Self-Training is Semi-Supervised learning
- Few-shot learning in Machine learning

Module 6: Forcasting Models

- ARIMA (Auto-Regressive Integrated Moving Average)
- SARIMA (Seasonal ARIMA)
- Exponential Smoothing (Holt-Winters)

Date: __/__/20__ Day: _____

Module 7: Deployment of ML Model

- Machine learning deployment
- Deploy ML Model using streamlit library
- Deploy ML Web app on Heroku
- Create UIs for prototyping ML model with Gradio

Fast Api / Flask

- Deploy Machine learning Model using Flask
- Deploy ML Models as API using FastAPI

MLOps

- MLOPS
- Contineous Integration & Contineous Deployment (CI/CD)
- End-to-End MLOPS

Module No: 1 Machine learning Pipeline

ML Workflow

Machine learning lifecyle is the step-by-step process to built, deploy, and maintain ML models. It make sure the model is accurate, reliable, scalable.

Steps of ML Lifecycle

1- Problem Defenation

- Work with stakeholders to understand business goals.
- Define objectives, scope, and succes criteria.
- Be clear on the desired outcomes.

2- Data Collection

- Collect relevant data for the problem
- Ensure quality (accurate, ethical)
- Gather enough data (quantity)
- Use diverse datosets for broad coverage

Date: __/__/20 Day: _____

3- Data Cleaning & Preprocessing
- Fix missing values, outliers, and inconsistencies
- Standarized formats, scale values, encode categories
- Prepare data for meaningful analysis.

4. Exploratory Data Analysis (EDA)
- Use statistics and visuals to explore
- Find hidden patterns, trends. and challenges
- Gain insights for decision-making
- Help in feature engineering and model choice
-

5. Feature Engineering & Selection
- Engineering: Create or transform features to capture patterns
- Selection: Pick only important features for efficiency
- Use domain knowledge for better features
- Balance accuracy vs complexity

6. Model Selection
- Choose model based on problem type, data & complexity
- Consider performance, interpredability, scalability
- Test multiple models and compare results

7 Model Training
- Train the chosen model on data
- Adjust parameters to reduce errors
- Optimize for better predictions
- Validate on unseen data

8. Model Evaluation & Tuning
- Test with validation/ test datasets
- Use metrics: accuracy, precision, recall, F1.

Date: __/__/20 Day: _____

- Identify strengths & weaknesses
- Tune hyperparameters for improvement
- Ensure robustness and reliability

9. Model Deployment
- Integrate the trained model into real systems
- Provide APIs or pipelines for production
- Ensure scability, security, and smooth use

10. Model Monitering & Maintenance
- Track performance over time
- Detect data drift or accuracy drops
- Retrain when needed
- Keep logs and alerts for real-time issues

Data Cleaning

Data cleaning means fixing missing, duplicate, irrelevant, or inconsistent data. Row data (logs, transaction, audio, video, etc) is usually noisy, incomplete, inconsistent, and harms model accuracy.

Goal:

Make data accurate, consistent, & error free

Clean data also makes EDA easier & helps in better insights

Benefits of Data Cleaning
- Improves model performance
- Increases accuracy
- Better data representation
- Improves data quality & reliability
- Improves data security (removes sensitive info)

Date: __/__/20__ Day: _____

How to perform Data Cleaning

- Remove Unwanted Observations - delete duplicates, irrelevant, noisy data
- Fix Structural Errors - standarize formats, variable type
- Manage Outliers - detect & remove or transform extreme values
- Handle Missing Data - Imputation, deletion, or advanced methods

Implementation (Titanic Dataset Example)

Step 1: Import libraries & load dataset

```
import pandas as pd, numpy as np
df = pd.read_csv('Titanic-Dataset.csv')
```

Step 2: Check Duplicates

```
df.duplicated()
```

Step 3: Identify column data types

```
cat_col = [c for c in df.columns if df[c].dtype == 'object']
num_col = [c for c in df.columns if df[c].dtype != 'object']
```

Step 4: Count unique categorical values

```
df[cat_col].nunique()
```

Step 5: Calculate missing values (%)

```
round((df.isnull().sum()/df.shape[0])*100, 2)
```

Step 6: Drop irrelevant / heavy-missing columns

```
df1 = df.drop(columns = ['Name', 'Ticket', 'cabin'])
df1.dropna(subset = ['Embarked'], inplace = True)
df1.['Age'].fillna(df1['Age'].mean(), inplace = True)
```

Step 7: Detect outliers with Boxplot

```
import matplotlib.pyplot as plt
plt.boxplot(df1['Age'], vert = False); plt.show()
```

Date: __/__/20__ Day: _____

Step 8: Remove outliers (mean ± 2*std)

```
mean, std = df1['Age'].mean(), df1['Age'].std()
lb, ub = mean - 2*std, mean + 2*std
df2 = df1[(df1['Age'] >= lb) & (df1['Age'] <= ub)]
```

Step 9: Fill missing again if needed

```
df3 = df2.fillna(df2['Age'].mean())
```

Step 10: Recalculate & remove outliers again

```
mean, std = df3['Age'].mean(), df3['Age'].std()
lb, ub = mean - 2*std, mean + 2*std
df4, = df3[(df3['Age'] >= lb) & (df3['Age'] <= ub)]
```

Step 11: Data validation & feature selection

- Independent (x): Pclass, Sex, Age, SibSp, Parch, Fare, Embarked
- Target (Y): Survived

```
X = df4[['Pclass', 'Sex', 'Age', 'Sibsp', 'Parch', 'Fare', 'Embarked']]
Y = df4['Survived']
```

Step 12: Data formating (Scalling & Normalization)

- Min-Max Scaling (0-1 range)

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler((0, 1))
X[num_col] = scaler.fit_transform(X[num_col])
```

- Standarized (Z-score)

$z = (x - \mu) / \sigma$

Centers around mean = 0, std = 1

Data Cleaning Tools

- OpenRefine – free, open-source
- Trifacta Wrangler – AI-powered cleaning
- TIBCO Clarity – Profiling & Cleansing
- Cloudingo – cloud-based deduplication (good for CRM)
- IBM InfoSphere QualityStage – enterprise-level cleansing

Date: __/__/20 Day: _____

Advantages
- Better performance & accuracy
- Cleaner, more reliable data
- Better data representation
- Can remove sensitive info

Disadvantages
- Very time-consuming (big data)
- Risk of losing useful info
- Needs resources & tools
- Over-cleaning may cause overfitting
-

Data Preprocessing in Python

Data preprocessing = cleaning, transforming, and organizing raw data → makes it accurate, consistent, and ready for ML models

Why Important?
- Clean Data = models learn real patterns, not noise
- Prevents misleading inputs → reliable predictions
- Helps in EDA (patterns, trends easier to see)
- Improves overall model performance

Steps of Data Preprocessing

Step 1: Import Libraries & load Dataset

```
import pandas as pd, numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import seaborn as sns, matplotlib.pyplot as plt.

df = pd.read_csv('diabetes.csv')
```

Date: __/__/20__ Day: _____

Step 2: Inspect Data & Missing Values

df.info()

df.isnull().sum()

→ check size, datatypes, missing values.

Step 3: Summary & Outliers

- df.describe() → mean, std, min/max, quartiles
- Boxplots → visualize spread & outliers

Step 4: Remove Outliers (IQR Method)

- IQR = Q3 - Q1
- Outliers = < Q1 - 1.5/QR or Q3 + 1.5IQR

q1, q3 = np.percentile(df['Insulin'], [25, 75])

iqr = q3 - q1

lower, upper = q1 - 1.5 * iqr, q3 + 1.5 * iqr

clean_df = df[(df['Insulin'] >= lower) & (df['Insulin'] <= upper)]

Step 5: Correlation Analysis

- df.corr() → pairwise correlations
- Heatmap → visualize relationships
- Sort correlations with corr['Outcome']

Step 6: Target Distribution

- Check class balance (Diabetes Vs Not)

plt.pie(df['outcome'].value_counts(), labels=['Diabetes', 'Not Diabetes'], autopct = '%.f%%%')

Step 7: Separate Features & Target

X = df.drop(columns = ['Outcome'])

Y = df['Outcomes']

Step 8: Feature Scaling (Normalization & Standarization)

- Normalization (Min-Max, 0-1)

Good for K-NN Neural Nets

scaler = MinMaxScaler()

X_norms = scaler.fit_transform(X)

Date: __/__/20__ Day: _____

- Standorized (Z-score, mean = 0, std = 1)

Good for Gaussian assumptions

```
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

Advantages

- Better Data quality
- Higher model accuracy
- Less overfitting
- Faster training
- Ensures algorithm compability.

Feature Scaling

Feature engineering = creating, transforming, or selecting features to improve ML models

Why Important?

- Good features = models learn better
- Reduces noise & irrelevant info
- Prevents overfitting
- Makes models easier to interpret

Main Techniques

1- Absolute Maximum Scaling

- Formula: $X_scaled = X_i / max(|X|)$
- Values → range [-1, 1]
- Problem: Very sensitive to Outliers
- Use: Simple scaling, sparse data

```
max_abs = np.max(np.abs(df), axis = 0)
scaled_df = df / max_abs
```

Date: __/__/20 Day: _____

2- Min Max Scaling

- Formula: $(x_i - X_{min}) / (X_{max} - X_{min})$
- Values → range [0, 1] (or custom range)
- Problem: Sensitive to outliers
- Use: Neurals nets, bounded input features

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_df = pd.DataFrame(scaler.fit_transform(df),
                         columns = df.columns)
```

3- Normalization (Vector Norm)

- Formula: $X_i / ||X||$ (row scaled to unit length)
- Focuses on direction, not magnitude
- Not affected by outliers (works row by row)
- Use: Text classification, clustering, cosine similority

```
from sklearn.preprocessing import Normalizer
scale = Normalizer()
scaled_df = pd.DataFrome(scaler.fit_transform(df),
                         columns = df.columns)
```

4- Standardization (Z-Score)

- Formula: $(X_i - \mu) / \sigma$
- Values → means = 0, variance = 1
- Moderate Sensitively to Outliers
- Use: Works best if data ≈ normal distribution. Used in most ML models (LR, SVM, NN)

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_df = pd.DataFrame(scaler.fit_transform(df),
                         columns = df.columns)
```

5- Robust Scaling

- Formula: $(X_i - median) / IQR$
- Uses median & IQR (middle 50%)

Date: __/__/20 Day: _____

- Not affected much by outliers
- Use: Skewed data, outliers present

```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
scaled_df = pd.DataFrame(scaler.fit_transform(df),
                         columns = df.columns)
```

Advantages of Scaling

- Better accuracy & predictions
- Faster training (convergence)
- Pervents large features dominating
- Improves numerical stability
- Ensures compatibility with ML algorithms (SVM, KNN, NN etc)

Feature Extraction

Feature extraction = transform row data → smaller informative features

- Makes data easier for ML models
- Improves accuracy & reduces computation

Importance

- Less Computation: simpler Data, faster Processing
- Better performance → focuses on key features
- Clear Insights → removes noises
- Prevents overfitting → avoids too many features

Key Techniques

1- Statistical Methods

- Mean → average
- Median → middle value
- Std Dev → spread of data

Date: __/__/20__ Day: _____

- Correlation & Covarience → relationships
- Regression → relation b/w variables

2- Dimensionally Reduction
- PCA → keeps features with most variance
- LDA → separates classes clearly
- t-SNE → visualize high-dimensional data (2D/3D)

3- Text Feature Engine xtration
- BoW → word counts (ignores order)
- TF-IDF → highlights important/unique words

4- Signal Processing
- Fourier Transform → time → frequency
- Wavele Transform → time → frequency (for changing signals)

5- Image Feature Extraction
- HOG → edge/gradient patterns (object detection)
- CNN Features → deep layers extract hierarchial image patterns.

Choosing Method
- Depends on data type & problem
- Risk: may lose info or need high computation

Applications
- Vision → self-driving cars, object detection
- NLP → spam filters, sentiment analysis
- Biomedical → EEG/MRI signals → diagnosis
- Industry → predictive maintainence
- Finance → fraud detection

Date: __/__/20 Day: _____

Tools & Libraries

- Scikit-learn → PCA, LCA, preprocessing
- OpenCV → SIFT, SURF, ORB (image features)
- TensorFlow/Keras, PyTorch → deep learning, CNN features
- NLTK → BoW, TF-IDF, embeddings

Advantages

- Simplifies data
- Speeds training & predictions
- Boosts accuracy
- Focuses on patterns
- Improve generalization

Challenges

- High-dimensional data → hard to manage
- Over/underfitting risk (too many/too few features)
- Expensive computation
- Reduadant/noisy features hurt efficiency

Feature Engineering

Feature Engineering = changing raw data into useful features that improves machine learning models

A. feature (or attribute) is an input property of the data (numerical, categorical, or text)

Example:

In house price prediction, features = bedrooms locations, property age

Why is Feature Engineering Important?

- Improve Accuracy → right features → better predictions
- Reduce overfitting → fewer, important feature → generalize well

Date: __/__/20__ Day: _____

- Boost interpretability - easier to explain model
- Enhance efficiency - less computation, faster training

Main Process

1- Feature Creation - make new features
- Domain - specific (industry knowledge)
- Data - driven (patterns in data)
- Synthetic (combine existing features)

2- Feature Transformation - modify features
- Normalization & Scaling
- Encoding (categorical - numeric)
- Log/Math transformations

3- Feature Extraction - get meaningful info
- Dimensionally reduction (PCA)
- Aggregation (sum/avg)

4- Feature Selection - pick best features
- Filter (correlation, stats)
- Wrapper (test subsets with models)
- Embedded (built in model methods like Lasso)

5- Feature Scaling - balance contribution
- Min-Max scaling ($0 \rightarrow 1$)
- Standard scaling (mean = 0, variance = 1)

Steps in Feature Engineering
- Data Cleaning - fix errors/missing values
- Data Transformation - scaling, encoding
- Feature Selection - keep only important ones

Date: __/__/20 Day: _____

- Feature Extraction – makes new ones from old
- Feature Iteration – test → refine → repeat

Common Techniques

1- One-Hot Encoding – categorical – Binary columns

```
import pandas as pd
df = pd.DataFrame({'Color': ['Red', 'Blue', 'Green', 'Blue']})
print(pd.get_dummies(df, columns = ['Color'], prefix =
    'Color'))
```

2- Binning – contineous – ranges

```
bins = [0, 20, 40, 60, 100]
labels = ['0-20', '21-40', '41-60', '61+']
df['Age Group'] = pd.cut(df['Age'], bins= bins,
    labels = labels, right = False)
```

3- Text Preprocessing – Clean text

- Remove stopwords
- Stemming
- Vectorization

4- Feature Splitting – break into parts

```
df[['Street', 'City', 'Zip']] = df['Full_Address'].str
    .extract(r'([0-9]+\s([\w\s]+),
        \s([\w\s]+), \s(\d+)')
```

Tools for feature Engineering

- Featuretools – auto feature creation
- TPOT – genetic algorithms for features & models
- DataRobot – full ML automation
- Alteryx – drag & drop workflows
- H2O.ai – scaling, encoding, imputation + visualization

Date: __/__/20__ Day: _____

Feature Selection Techniques

Feature selection = Keep only the features that really help prediction.
Result → simpler faster, accurate, less overfitting, easier to interpret models.

Why Feature Selection is Needed

- Better Accuracy - only useful features → model learns well
- Faster training - fewer features → saves time & computation
- More Interpretability - easier to explain results
- Avoid Curse of Dimensionality - too many features hurt performance

Types of Features Selection Models

1- Filter Methods

- Works in preprocessing, independent of model
- Select features using statistics (correlation, tests)

Advantages: Fast, simple, works with any model
limit: Ignores interactions b/w features, choice of metric matters.

Techniques:

- Information Gain - measures reduction in entropy
- Chi-square Test - tests relation b/w categorical features
- Fisher's Score - higher score = better feature
- Pearson Correlation - -1 to 1, measures linear relationship
- Variance Threshold - drop low-variance features
- Mean Absolute Difference - like variance but without squaring
- Dispersion Ratio - (Arithematic Mean ÷ Geometric Mean)
  - Higher = more relevant

Date: __/__/20 Day: _____

2- Wrapper Methods

- Train model on different features subsets
- Add/remove features until best set is found (greedy search)

Advantages: Model-specific, often higher performance

Limits: Slow, Costly, risk of overfitting

Techniques

- Forward Selection - start empty → keep adding best features
- Backward Elimination - start with all → remove weakest
- Recursive Elimination (RFE) - repeatedly drop least important until desired size

3- Embedded Methods

- Do feature selection during model training
- Mix of filter + wrapper

Advantages: Efficient, model-based learning

Limits: Harder to interpret, not all models support

Techniques

- L1 ~~Regression~~ Regularization (Lasso) - forces some coefficients to 0 - selects features
- Decision Trees / Random Forests - naturally pick important features (splitting)
- Gradient Boosting - selects features that reduce error most.

Choosing the Right Method

- Large Dataset → Filter (fast)
- Small dataset → wrapper (better accuracy)
- Tree models → Embedded (builtin)
- Need explainability → Filter
- Limited resources → Avoid wrapper (too slow)

Date: __/__/20__ Day: _____

Exploratory Data Analysis

Exploratory Data Analysis

EDA = analyzing and visualizing data to understand features, find patterns, detect errors, and prepare data for modeling

Why EDA is Important?

- Understand datasets (features, types, distributions)
- Find hidden patterns & relationships
- Detect Errors, missing data, outliers
- Identify important features for modeling
- Choose the best modeling methods

Types of EDA

1 Univariate Analysis (one variable)

- Histograms → distribution
- Box plots → outliers & spread
- Bar charts → categorical data
- Summary stats → mean, median, mode, variance, std

2- Bivariate Analysis (two variables)

- Scatter plot → relation b/w contineous vars
- Pearson correlation → strength of Linear relation
- Cross-tabulation → relation b/w categorical vars
- Line graph → compare vars over time
- Covariance → how two vars change together

3- Multivariate Analysis (more than two variables)

- Pair Plots → multiple variable relations
- PCA → reduce complexity
- Spatial analysis → maps/geographical data

Date: __/__/20 Day: _____

- Time series analysis → trends, seasonality (line plots, autocorrelation, moving, avg, ARIMA)

Steps of EDA

1- Understand Problem and Data
- Define business / research goal
- Know variables & data type
- Spot data issues / limitations

2- Import & Inspect Data
- load dataset (Python, R, Excel etc)
- Check rows, columns, structure
- Finding missing values & data types
- Detect inconsistencies & errors

3- Handle Missing Data
- Identify type (MCAR, MAR, MNAR)
- Options: remove rows/columns or impute values
- Imputation → mean, median, regression, KNN, ML methods

4- Explore Data Characteristics
- Use Summary stats (mean, median, std, skewness, kurtosis)
- Understand spread, shape & anomalies

5- Transform Data
- Scale/ normalize (min-max, standarization)
- Encode categorical (one-hot, Label encoding)
- Fix skewness (log, sqrt)
- Create new features (ratios, combinations)
- Group/ aggregate data

Date: __/__/20__ Day: _____

6. Visualize Data

- Categorical → bar, pie, freq tables
- Numerical → hist, box, violin, density
- Relationships → scatter, correlation matrix, Pearson/Spearman tests

7. Handle Outliers

- Detect → IQR, Z-score, domain rules
- Remove or adjust based on context

8. Communicate Findings

- State goals & key insights
- Use visuals to explain results
- Highlights anomalies, patterns, limitations
- Suggest next steps.

Tools

- Python: Pandas, Matplotlib, Seaborn, Plotly
- R: ggplot2, dplyr, tidyr

Exploratory Data Analysis in Python

EDA = using stats + visualizations to understand patterns, trends, relationships in data.

Python libraries: pandas, numpy, matplotlib, seaborn

Steps for EDA in Python

Step 1: Import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings as wr
wr.filterwarnings('ignore')
```

Date: __/__/20 Day: _____

Step 2: Load Dataset

df = pd.read_csv("/content/wineQT.csv")

df.head()

Step 3: Analyze Data

- df.shape → rows & columns
- df.info() → column types, nulls, memory
- df.describe().T → stats (mean, std, min, quartiles)
- df.columns.tolist() → column names

Step 4: Check missing Values

df.isnull().sum()

→ Null values per column

Step 5: Check Duplicate / unique

df.nunique()

→ unique values in each column

Step 6: Univariate Analysis (one variable)

1- Bar Plot → wine quality counts

2- Kernal Density Plot → distribution + skewness

- Skew = 0 → symmetric
- Skew ≥ 1 → right skew (long right tail)

3- Swarm Plot → outliers (quality vs alcohol)

Step 7: Bivariate Analysis (two variables)

1- Pair Plot → distribution + relationships

- Diagonal = histograms
- Off-diagonal = scatter plots
- Symmetry shows balanced relation

Date: __/__/20__ Day: _____

2- Violin Plot (alcohol vs quality)

- Width = density (more data points)
- Median line = central tendency
- Long tails = high variability

3- Box Plot (alcohol vs quality)

- Box = IQR (spread)
- Median line = central value
- Whiskers = min/max range
- Dots = outliers

Step 8: Multivariate Analysis (3+ variables)

- Correlation Heatmap

```
sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='Pastels2')
```

- +1 = strong positive
- -1 = strong negative
- 0 = no correlation
- Dark color = strong relation

Advanced EDA

EDA = First step in data analysis. Helps us see patterns, find outliers, test assumptions before ML models. Its stats + visualization

Descriptive Statistics

- Mean → Average value
  - Best: when data is normal & no extreme values
  - Bad: when outliers skew results
  - Example: Avg monthly sales of a shop

- Median: → middle value
  - Best: skewed data or with outliers

Date: __/__/20 Day: _____

- Bad: when you want the true average
- Example: Typical household income

- Mode → most frequent value
  - Best: categorical / discrete data
  - Bad: contineous data (like height)
  - Example: Most sold product in store

- Standard Deviation (SD) → spread around mean
  - Best: measure variability
  - Bad: misleading with outliers/skew
  - Example: ~~Example:~~ Daily website traffic variance

- IQR (Q3 - Q1) → middle 50% spread
  - Best: detect outliers
  - Bad: when data is normal
  - Example: Exam scores spread

- Skewness → data leaning left/right
  - Best: check if transformation needed
  - Bad: if already symentic
  - Example: Monthly sales with holiday spikes

- Kurtosis → tail heavyness (outlier)
  - Best: check for extreme values
  - Bad: normal well-behaved data
  - Example: Stock returns with extreme losses

- Visualizing Distributions
- Bar Plot → frequency of category
  - Compare categories: Too many = clutter
  - Example: Sales by product Type

Date: __/__/20__ Day: _____

- Stacked Bar Graph → categories + subcategories
  - see breakdown: Too complex with many
  - Example: Product sales by region

- Histogram → distribution of continuous data
  - See frequency bins. Misleading with outliers
  - Example: Visitor time on website

- Box Plot → min, Q1, median, Q3, max, outliers
  - Compare groups & find outliers. Bad with small data
  - Example: Home price ranges by region

- Violin Plot → box + density
  - Compare distribution in detail. Overkill for 2 groups
  - Example: Blood pressure by age group

- Pie Chart → part-to-whole
  - Show proportion. Many categories = messy
  - Example: Product market share

- Correlation Heatmap → correlation matrix
  - Detect multicollinearity. Too many vars = clutter
  - Example: Customer satisfaction survey metrics

- Scatter Plot → relation b/w 2 continuous vars
  - Show trends, correlation. Not good for categorical
  - Example: House price vs size

- Multivariate Data (Feature Interactions)
- Facet Grids → multiple subplots by category
  - Compare subsets. Too many = clutter
  - Example: Sales by season

Date: __/__/20 Day: _____

- Pair Plots → grid of scatterplots
  - Show relations across variables. Overwhelming with
  - many vars.
  - Example: Age vs income vs spending

- Identifying Outliers
- Z-Score → how far point is from mean (in SDs)
  - Best: normal data
  - Bad: for skewed data
  - Example: Spike sales days

- Isolation Forest / LOF → ML-based outliers detection
  - Best: Large / high-dimensional data
  - Bad: small / simple data
  - Example: Fraud detection in transaction

- Feature Engineering
- Log Transformation → reduce skew, compress extremes
  - Best: skewed / exponential data
  - Bad: already normal
  - Example: Income distribution

- Polynomial Features → add powers (x^2, x^3)
  - Best: Non-linear relationships
  - Bad: Risk of overfitting
  - Example: House price vs $area^2$

- Interaction Features → combine variables
  - Best: Joint effect of features
  - Bad: Too many = overfit
  - Example: Age × income → luxury purchases

Date: __/__/20__ Day: _____

- Dimensionality Reduction
- PCA (Principal Component Analysis) → Linear reduction
  - Best: reduce correlated features
  - Bad: non-linear data, categorical
  - Example: Stock returns → few principal components

- t-SNE → non-linear, for visualization (2D/3D)
  - Best: clustering, patterns
  - Bad: Slow, bad for large data, distorts distances
  - Example: Customer clusters from purchase history

- UMAP → non-linear, faster than t-SNE, keeps local+global
  - Best: Large Data visualization
  - Bad: Not for exact distances, needs tuning
  - Example: Customer Interaction features for segmentation

Time Series Analysis & Visualization

Time Series = data collected over time in order (e.g stock prices, temperature)

Goal = find trends, seasonality, noise, stationarity → support forcasting & decisions

Key Concepts

- Trend → long-term direction (up, down, flat)
- Seasonality → repeating patterns (daily, monthly, yearly)
- Moving average → Smooth short-term fluctuations
- Noise → random changes with no pattern
- Differencing → subtract past value to remove trend
- Stationarity → mean/variance constant over time
- Order → how many times differencing needed
- Auto correlation (ACF) → Similarity with lagged version
- Resampling → change frequently (daily → monthly)

Date: __/__/20__ Day: _____

Types of Time Series

- Contineous → measured regularity (temp, Stock price)
- Discrete → counts/events (sales unit, categories)

Visualization Approaches

- Continuous Data → line plots, area charts
- Discrete Data → bar charts, histograms

Practical Steps in Python

- Step 1: Import Libraries

```
import pandas as pd, numpy as np, seaborn as sns
matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import adfuller
```

- Step 2: Load data

```
df = pd.read_csv("stock_data.csv", parse_dates=True,
                 index_col= "Date")
```

Step 3: Clean Data

```
df.drop(columns = "unnamed: 0", inplace = True)
```

Step 4: Plot high Stock prices (Line Plot)

```
sns.lineplot (data = df, x = df.index, y = 'High')
```

Step 5: Resampling

```
df_resampled = df.resample('M').mean()
sns.lineplot (data = df_resampled, x = df_resampled.index,
              y = 'High')
```

Date: __/__/20__ Day: _____

Step 6 : Seansonlity (ACF Plot)

```
plot_acf(df['High'], lags = 40)
```

Step 7: Stationarity Test (ADF Test)

```
result = adfuller(df['High'])
print (result[0], result[1])
```

- High p-value > 0.05 → Not stationary
- low p-value < 0.05 → Stationary

Step 8: Differencing

```
df['high_diff'] = df['High'].diff()
```

Step 9: Moving average (smoothing)

```
df['high_smoothed'] = df['High'].rolling(window=120).mean()
```

Step 10: Original Vs Differenced

```
pd.concat([df['High'], df['high_diff']], axis=1).head()
```

Step 11: ADF Test After Differencing

```
result = adfuller(df['High_diff'].dropna())
```

Model Evaluation

Regularization in Machine learning

Regularization = method to pervent overfitting by adding a penalty for complexity. It makes models simpler, more general, stable, and accurate on unseen data.

Why Use Regularization?

- Prevent Overfitting → avoids memorizing noise
- Improves Generalization → better on unseen data
- Simplifies Model → reduces unnecessary complexity

Date: __/__/20 Day: _____

Types of Regularization

- Lasso Regression (L1)
  - Adds absolute values of coefficients as penalty
  - Shrinks some coefficient to zero → feature selection

Formula:

$$Cost = \frac{1}{n}\sum(y_i - \hat{y}_i)^2 + \lambda\sum|w_i|$$

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha = 0.1).fit(X_train, y_train)
print(lasso.coef_)
```

- Ridge Regression (L2)
  - Adds squared values of coefficients as penalty
  - Shrinks coefficients but doesn't make them zero
  - Best when features are correlated (multicollinearity)

Formula:

$$cost = \frac{1}{n}\sum(y_i - \hat{y}_i)^2 + \lambda\sum w_i^2$$

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha = 1.0).fit(X_train, y_train)
print(ridge.coef_)
```

- Elastic Net (L1 + L2)
  - Combines Lasso + Ridge
  - Has 2 parameters
    - λ (alpha) → strength of regularization
    - l1_ratio (α) → balance b/w L1 & L2
      - $\alpha = 1$ → Lasso, $\alpha = 0$ → Ridge

Formula:

$$cost = \frac{1}{n}\sum(y_i - \hat{y}_i)^2 + \lambda\left((1-\alpha)\sum|w_i| + \alpha\sum w_i^2\right)$$

Date: __/__/20__ Day: _____

```
from sklearn.linear_model import ElasticNet
model = ElasticNet(alpha=1.0, l1_ratio = 0.5).fit(
                X_train, y_train)
print(model.coef_)
```

Benefits of Regularization

- Stops Overfitting
- Improves interpretability (Lasso selects features)
- Enhance accuracy by controlling outliers / noise
- Keeps model stable → less sensitive to data changes
- Handles multicollinearity (Ridge helps)
- Fine-tuning possible via alpha / λ
- Ensures consistent performance across datasets.

Confusion Matrix

Confusion matrix = table that compares predicted vs actual results in classification

It shows where model is right or wrong

4 Key Terms

- TP (True Positive): Predicted positive & actual positive
- TN (True Negative): Predicted negative & actual negative
- FP (False Position): Predicted positive but actual negative
- FN (False Negative): Predicted negative but actual positive

Metrics From Confusion Matrix

- Accuracy (overall correctness)

$$\frac{TP + TN}{TP + TN + FP + FN}$$

- Can be misleading if classes are imbalanced.

Date: __/__/20__ Day: _____

- Precision (quality of Positive predictions)

$$\frac{TP}{TP + FP}$$

- Useful when false positive are costly (e.g fraud)

- Recall (ability to catch positive)

$$\frac{TP}{TP + FN}$$

- Useful when false negatives are costly (e.g medical tests)

- F1-Score (balance of precision & recall)

$$\frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

- Good for imbalanced data

- Specificity (True negative Rate)

$$\frac{TN}{TN + FP}$$

- Ability to correctly identify negatives

Errors

- Type I Error (FP): Model says +ive but actually -neg — Hurts precision
- Type II Error (FN): Model says -neg but actually +ive — Hurts recall
  - Example (medical test)
    - Type I (FP): Healthy predicted as sick
    - Type II (FN): Sick predicted as healthy

Binary Classification Example (Dog Vs Non Dog)

- Actual Dog = 6, not Dog = 4
- Results = TP = 5, TN = 3, FP = 1, FN = 1

Date: __/__/20__ Day: _____

Confusion Matrix

	Predicted Dog	Predicted Not Dog
Actual Dog	TP = 5	FN = 1
Actual Non Dog	FP = 1	TN = 1

Example

```
from sklearn.metrics import confusion_matric, classification_report
import seaborn as sns
matplotlib.pyplot as plt
import numpy as np

actual = np.array(['Dog', 'Dog', 'Not Dog', 'Dog', 'Dog'])
predicted = np.array(['Dog', 'Not Dog', 'Dog', 'Dog', 'Dog'])

cm = confusion_matrix(actual, predicted)
sns.heatmap(cm, annot = True, fmt = 'g', xticklabels
        = ['Dog', 'Not Dog'], yticklables = ['Dog', 'Not Dog'])

plt.show()
print(classification_report(actual, predicted))
```

Multi-Class Confusion Matrix

- Rows = actual classes
- Columns = predicted classes
- Each cell = count of predictions

Example (cat, Dog, Horse - 30 images)

Actual / Pred	Cat	Dog	Horse
Cat	8	1	1
Dog	2	10	0
Horse	0	2	8

Date: __/__/20 Day: _____

Example

```
from sklearn.metrics import confusion_matrix, ConfusionMatrix
                - Display, classification_report
y_true = ['Cat']*10 + ['Dog']*12 + ['Horse']*10
y_pred = ['Cat']*8 + ['Dog'] + ['Horse'] + ['Cat']*2 +
        ['Dog']*10 + ['Horse']*8 + ['Dog']*2

cm = confusion_matrix(y_true, y_pred, labels = classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                display_labels = classes)
disp.plot()
print(classification_report(y_true, y_pred, target_names
                = classes))
```

Precision, Recall & F1-Score

- Precision

Out of all predictions labled Yes (Positive) by the model, how many were actually correct

$$\text{Precision} = \frac{TP}{TP + FP}$$

- TP (True Positive) → correctly predicted "yes"
- FP (False Positive) → wrongly predicted "yes"

Example: Detecting spam emails

- If models marks 10 emails as spam and 8 are real spam → Precision = 8/10 = 80%
- High Precision = fewer false ~~Ex~~ alarms
- Limitation: Model may miss real positives (too strict)

- Recall

Out of all actual Yes (Positive) cases, how many

Date: __/__/20__ Day: _____

did the model find

$$\text{Recall} = \frac{TP}{TP + FN}$$

- TP → correctly predicted "Yes"
- FN (False Negative) → missed real 'Yes'

Example: Medical Test

- If 10 patients are sick but model finds only 8 → Recall = 8/10 = 80%
- High Recall = catches almost everything important
- Limitation: May cause many false alarms (FP)

Perecision Vs Recall

- Precision → How correct the 'Yes' prediction are
- Recall → How many real 'Yes' cases were found

- F1 Score

Balances Precision and Recall into one metric

Harmonic Mean

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Why Harmonic Mean? ensures both Precision and Recall must be high
- If one is very Low, F1 will also be low

Example (Binary Classification)

Dataset: 100 Samples

- TP = 80, FP = 5, FN = 10, TN = 5
- Precision = 80/(80 + 5) = 0.94
- Recall = 80/(80 + 10) = 0.88
- Accuracy = (80 + 5)/100 = 85%
- F1 = 0.91

Date: __/__/20 Day: _____

- F1 is useful when data is imbalanced

Special Cases

- Imbalanced Dataset (more negative)
  - Accuracy looks good, but F1 reveals poor performance
    - Example: Precision = 0.50, Recall = 0.11, F1 = 0.18

- Imbalanced Dataset (more positive)
  - Accuracy is low, but F1 captures strong balance
    - Example: Precision = 0.92, Recall = 0.99, F1 = 0.95

Multi-Class F1 Score

- Treat each class as "one vs rest"
- Calculate F1 for each class separately
- Combine using
  - Micro-average → global counts of TP, FP, FN
  - Macro-average → simple average across all classes
  - Weighted-average → weight by class size

Code (Example)

```
from sklearn.metrics import f1_score
y_true = [0, 1, 2, 2, 2, 2, 1, 0, 2, 1, 0]
y_pred = [0, 0, 2, 2, 1, 2, 1, 0 1, 2, 1]

print("Per class:", f1_score(y_true, y_pred, average = None))
print("Micro:", f1_score(y_true, y_pred, average = 'micro'))
print("Macro:", f1_score(y_tune, y_pred, average = "macro"))
print("Weighted:", f1_score(y_true, y_pred, average='weighted'))
```

Date: __/__/20__ Day: _____

AUC-ROC Curve

AUC-ROC is a graph to check how good a binary classifier is at separating positives (diseases) from negatives (no disease) at different thresholds

It plots:

- True Positive Rate (TPR / Recall / Sensitively) → TP/(TP+FN)
- False Positive Rate (FPR) → FP/(FP + TN)
- Specificity = 1 - FPR (correctly catching negatives)

→ Higher Curve = better model

Confusion Matrix Basics

- TP → Correctly predicted positives
- TN → Correctly predicted negatives
- FP → Predicted positive but was negative
- FN → Predicted negative but was positive

ROC & AUC

- ROC curve → plots TPR vs FPR at different thresholds. Shows trade-off between sensitively & specificity
- AUC (Area Under Curve) → area under ROC
  - 1.0 = perfect model
  - 0.5 = random guess
  - <0.5 = bad model

How AUC-ROC Works

- Take one positive and one negative sample
- Check if positive has higher probability than negative
- Repeat for all pairs
- AUC = probability model ranks positives higher than negatives.

Date: __/__/20__ Day: _____

When To Use

- Balanced datasets
- When FP and FN are equally important
- For highly imbalanced data → use prediction-precision-Recall Curve instead.

Model Performance

- High AUC (~1) → strong separation b/w classes
- Low AUC (~0) → model fails
- AUC ~ 0.5 → random guessing

Implementation (Binary Classification)

1- Import libraries

```
import numpy as np, pandas as pd,
matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
```

2- Generate Data

```
X, y = make_classification(n_samples = 1000, n_features = 20,
                           n_classes = 2, random_state = 42)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                           test_size = 0.2, random_state 42)
```

3- Train Models

```
logistic_model = LogisticRegression(random_state = 42).fit
                           (X_train, y_train)
rf_model = RandomForestClassifier(n_estimators = 100,
                           random_state = 42).fit(X_train, y_train)
```

Date: __/__/20__ Day: _____

4 - Predictions

```
y_pred_logistic = logistic_model.predict_proba(X_test)[:, 1]
y_pred_rf = rf_model.predict_proba(X_test)[:, 1]
```

5. Plot ROC Curves

```
plt.figure(figsize=(7, 5))
for model, preds in {'Logistic': y_pred_logistic,
    'RandomForest': y_pred_rf}.items():
    fpr, tpr, _ = roc_curve(y_test, preds)
    plt.plot(fpr, tpr, label=f"{model} (AUC = {auc
                                (fpr, tpr):.2f})")
plt.plot([0, 1],[0, 1], 'r--', label = "Random Guess")
plt.xlabel("FPR"); plt.ylabel("TPR"); plt.legend();
plt.show()
```

ROC-AUC in Multiclass

- Use One-vs-Rest (OvR) pick one class (+) others (-)
- Train classifier per Class
- Plot ROC curve & AUC per class

Cross-Validation

Cross-Validation = method to check how well a model performs on unseen data and pervent overfitting

It works by:

- Spliting data into parts
- Training on some parts, testing on the best
- Repeating with different splits
- Averaging results = final performance

Date: __/__/20__ Day: _____

Types of Cross-Validation

1- Holdout Validation

- Split once → usually 50% train, 50% test
- Simple & Quick
- High bias (model may miss patterns since only half data is used.)

2- LOOCV (Leave-One-Out CV)

- Train on all data except 1 point → test on that Point
- Repeat for every data point
- Low Bias (uses almost all data for training)
- High variance (testing on 1 sample)
- Very slow for big datasets

3- Stratified Cross-Validation

- Keeps class distribution same in all folds
- Useful for imbalanced datasets
- Each folds → 1 Testset + rest Train
- Balanced folds → better generalization

4- K-Fold Cross Validation

- Split dataset into K equal folds
- Train on (K-1) folds, test on 1 fold
- Repeat K times → each fold used once for testing
- Common choice → K = 10 suggested
- Small K → high bias, large K → closer to LOOCV

→ Example (K = 5, 25 samples)

- Iter 1: Test [1-5], Train [6-25]
- Iter 2: Test [6-10], Train rest
- Iter 3: Test [11-15], Train rest
- Iter 4: Test [16-20], Train rest
- Iter 5: Test [21-25], Train rest

Date: __/__/20__ Day: _____

K-Fold vs Holdout

Feature	K-Fold CV	Holdout
Training	Model trained K times	Model trained once
Split	K-folds	Single split
Bias/variance	Lower bias, stable	Higher bias, unstable
Time	Slower (K trainings)	Faster
Best for	Small/medium datasets	Very large datasets

Python Example (K-Fold with SVM)

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.svm import SVC
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target

svm = SVC(kernel='linear')
kf = KFold(n_splits=5, shuffle=True, random_state=42)

results = cross_val_score(svm, X, y, cv=kf)
print("Accuracy per fold:")
for i, acc in enumerate(results, 1):
    print(f"Fold {i}: {acc*100:.2f}%")
print(f"Mean Accuracy: {results.mean()*100:.2f}%")
```

→ Output shows acc. of each fold + mean acc (~97.33%)

Advantages

- Better performance estimate than single split
- Reduces overfitting
- Uses all data for both training & testing
- Works with many datasets & models

Date: __/__/20 Day: _____

Disadvantages

- Slow (K training)
- Time-consuming for large data. (esp. LOOCV)
- Bias-Variance tradeoff → small K = high bias, large K = high variance

Hyperparameter Tuning

What is Hyperparameter Tuning?

- Choosing the best values for models hyperparameters
- Done before training → controls how model learns
- Affects speed, accuracy, complexity of model

Examples:

- Neural Network → learning rate, neurons
- SVM → kernel size
- High learning rate → fast but may skip best solution
- Low learning rate → slow, needs more time/resources
- Each model → different hyperparameters

Techniques for Hyperparameters Tuning

1- GridSearchCV (Brute Force)

- Tests all combinations of hyperparameters
- Steps:
  - Make grid of values
  - Train model on all combos
  - Use cross-validation
  - Pick best score
- Downside: Very slow, heavy for big data

Example: Logistic Regression with C values → 20 models tested, best chosen

Date: __/__/20__ Day: _____

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import numpy as np
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=20,
        n_informative=10, n_classes=2, random_state=42)

c_space = np.logspace(-5, 8, 15)
param_grid = {'C': c_space}
logreg = LogisticRegression()
logerg_cv = GridSearchCV(logreg, param_grid, cv=5)
logreg_cv.fit(X, y)

print("Best Params:", logreg_cv.best_params)
print("Best Score:", logreg_cv.best_score_)
```

→ Example Output → C = 0.0061, Accuracy = 85.7%

2- Randomized Search CV

- Tries random combos (not all)
- Faster than grid search
- Uses cross-validation to select best

→ Example : Decision Tree

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import randint
param_dist = {
    "max_depth": [3, None],
    "max_features": randint(1, 9),
    "min_samples_leaf": randint(1, 9),
    "criterion": ["gini", "entropy"]
}
```

Date: __/__/20 Day: _____

```
tree = DecisionTree Classifier( )
tree_cv = RandomizedSearchCV (tree, param_dist, cv=5)
tree_cv.fit (X, y)
print( "Best Params:", tree.cv.best_params_)
print( "Best Score:", tree_cv.best_score_)
```

→ Example Output

→ Best Params = {criterion: entropy, max_depth: None, max_features: 6, min_sample: 6 }

Accuracy = 84.2%

3- Bayesian Optimization

- Smarter search → learns from past tries
- Uses a surrogate model to predict best hyperparameters
- Updates after each test, chooses next best to try

Common surrogate models

- Gaussian Processes
- Random Forest Regression
- Tree-structured Parzen Estimators (TPE)

Advantages

- Better model accuracy
- Prevents overfitting/underfitting
- Works well on unseen data (generalization)
- Saves computation time & memory
- Easier to interpret tuned models

Challanges

- Large hyperparameters spaces = expensive & slow
- Evaluating models is costly (big data)
- Needs domain knowledge to guide search
- Adaptive tuning (like learning rate schedules) is still developing

Date: __/__/20__ Day: _____

Module 2 Supervised Learning

Supervised learning algorithms are generally categorized into two main types

- Classification - where goal is to predict discrete labels or categories
- Regression - where aim is to predict continuous numerical values

Classification

What is classification

- Teaching a machine to sort data into categories using labeled examples
- After training, it predicts labels for new, unseen data

Examples:

- Emails → "spam" or "not spam"
- Images → "cat" or "dog"
- Uses features like color, texture, shape, size
- Model learns decision boundaries → separates classes (dogs vs cats)

Types of Classification

1- Binary Classification

- Two categories only
- Examples: spam vs not spam

2- Multiclass Classification

- More than two categories
- Example: animals → cat, dog, bird

3- Multi-label Classification

- One data point can have multiple labels
- Example: Movie → "action" + "comedy"

Date: __/__/20__ Day: _____

How Classification Works

1- Data Collection

- labeled dataset (cat/dog)

2- Feature Extraction

- pick useful features (color, shape)

3- Model Training

- learn mapping b/w features & labels

4- Model Evaluation

- Test on new unseen data

5- Prediction

- Assign class to new data

6. Iteration

- adjust algorithm/hyperparameters if accuracy is low

Applications of Classification

- Email spam filtering
- Credit risk assessment (loan default or not)
- Medical diagonsis (disease detection)
- Image classification (face recognition, autonomous cars)
- Sentiment analysis (Positive / negative / neutral text)
- Fraud detection (credit card fraud)
- Recommendation systems (Netflix, Amazon)

Classification Modeling

- Class Separation → distinguish b/w categories
- Decision Boundaries → line/region separating classes

Date: __/__/20 Day: _____

- Data Quality Sensitive → clean, labeled data = better accuracy
- Imbalanced Data Problem → solve with resampling / weights
- Interpertability → some models (Decision Trees) are easy to understand.

Classification Algorithms

1- Linear Classifiers
- Simple, fast, linear decision boundaries
- Examples:
  - Logistic Regression
  - Linear SVM
  - Perceptron (single layer)
  - SGD Classifier

2- Non-linear Classifiers
- Capture complex patterns, flexible
- Examples:
  - K-Nearest Neighbors (KNN)
  - Kernel SVM
  - Naive Bayes
  - Decision Tree

3- Ensemble Classifiers
- Combine multiple models for better accuracy
- Examples:
  - Random Forest
  - AdaBoost
  - Bagging Classifier
  - Voting Classifier
  - Extra Trees

Date: __/__/20__ Day: _____

4- Neural Networks

- Multi-layer Perceptrons (deep learning models)
- Handle complex, large-scale data

Regression

What is Regression?

- Supervised learning ~~model~~ used to predict a contineous number
- Finds relation b/w dependent variable (target) and Independent variables (features)
- Examples: Predict house price using size, room, location

→ Use when output = real/continuous (salary, weight, sales)

→ Simplest model = Linear Regression

Types of Regression

1- Simple Linear Regression

- One feature → one target
- Example: House price based on size

2- Multiple Linear Regression

- Many features → one target
- Example: House price prediction on size, room, location

3- Polynomial Regression

- For non-linear trends (adds powers of features)
- Example: Population growth over time

4- Ridge & Lasso Regression

- Regularization linear Regression
- Helps reduce overfitting by shrinking large coefficients

Date: __/__/20__ Day: _____

5- Support Vector Regression (SVR)

- Based on SVM
- finds best line/hyperplane to predict contineous values

6- Decision Tree Regression

- Splits data using tree-like rules
- Example: Predict customer behaviour (age, income)

7- Random Forest Regression

- Uses many decision trees and averages results
- Examples: Sales prediction, churn analysis.

Evaluation Metrics

- MAE (Mean Absolute Error): Avg Absolute difference
- MSE (Mean Squared Error): Avg squared difference
- RMSE (Root MSE): Square root of MSE
- Huber Loss: Mix of MAE + MSE (robust to outliers)
- R^2 Score: How well model fit (0 to 1, higher better)

Applications

- Price Prediction (houses, products)
- Trend forcasting (sales, demand)
- Risk factor analysis (medical, finance)
- Decision making (stock recommendation)

Advantages

- Easy to understand
- Works well with linear relationships
- Can be robust to outliers.

Date: __/__/20__ Day: _____

Disadvantages

- Assumes linearity
- Fails with multicollinearity (features highly correlated)
- Not great for very complex patterns.

Linear Regression

- Introduction to linear Regression
- Gradient Descent in linear Regression
- Multiple Linear Regression

What is linear Regression?

linear Regression is supervised machine learning algorithm used to model relationship b/w an independent variate (X) and a depended variable (Y) by fitting a straight line to data

It assume that the change in output (Y) is proportional to the change in input (X)

Example:

Predicting a student's exam score based on study hours

- Independent variable (X): Hours studied
- Dependent variable (Y): Exam Score

Equation:

$$y = mx + b$$

Why Linear Regression is Important?

- Simple & Interpretable → easy to explain results
- Predictive Power - forecast future outcomes

Date: __/__/20__ Day: _____

- Foundation Model - basis for logistic regression, neural nets
- Efficient → computationally inexpensive
- Widely used → finance, healthcare, economics, marketing

Best-Fit line

The goal is minimize error b/w predicted and actual values

$$y = mx + b$$

- m (slope): change in Y for each unit change in X
- b (intercept): value of Y when X = 0)

Error Minimization: Least Squares

Residual

$$\text{Residual} = y_i - \hat{y_i}$$

Sum of squared Errors (SSE)

$$SSE = \sum (y_i - \hat{y_i})^2$$

The best-fit line minimizes SSE

Hypothesis function

For simple regression:

$$h(x) = \beta_0 + \beta_1 x$$

for multiple regression:

$$h(x_1, x_2, \ldots, x_k) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k$$

Assumptions

- linearity — X and Y are linearly related
- Independent of Errors
- Homoscedasticity - constant variance of errors
- Normality of Errors
- No Multicollinearity (in multiple regression)
- No Autocorrelation (important in time Series)
- Additivity - total effect = sum of individual effects.

Date: __/__/20 Day: _____

Types of Linear Regression

1- Simple linear Regression

- One independent variable
- Equation: $\hat{y} = \theta_0 + \theta_1 x$

Example: Predict salary based on years of experience

2- Multiple linear Regression

- Multiple independent variables
- Equation: $\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Example: Predict house price using area, location, and number of rooms

Cost Function

We use Mean Squared Error (MSE):

$$J(\theta) = \frac{1}{n} \sum (y_i - \hat{y}_i)^2$$

Optimization: Gradient Descent

- Initialize slope & intercept randomly
- Calculate Error
- Compute gradient (partial derivative)
- Update parameters iteratively
- Stop when cost function coverages

Update rules

$$\theta_j = \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$$

Evaluation Metrics

- MSE (Mean Squared Error)
- MAE (Mean Absolute Error)
- RMSE (Root Mean Squared Error)
- R^2 (Coefficient of Determination
- Adjusted R^2

Date: __/__/20__ Day: _____

Regularization

To prevent overfitting

- Lasso (L1) - shrinks some coefficients to zero
- Ridge (L2) - reduces large coefficients but keeps all
- Elastic Net - combination of L1 + L2

logistic Regression

logistic Regression is a supervised ML algorithm for classification

- Linear Regression → predicts continuous values
- logistic Regression → predicts probability of class (0/1, Yes/No)
- Uses sigmoid function to map values into 0-1

Types of Logistic Regression

1- Binomial → Only 2 Classes (0/1, Yes/No, Pass/Fail)

2- Multinomial → 3+ unordered classes (cat/Dog/Sheep)

3- Ordinal → 3+ ordered classes (Low/Medium/High)

Assumptions

- Data points are Independent
- Depend variable is binary (or use softmax for >2)
- Independent vars have linear relation with log-odds
- No extreme outliers
- Needs large dataset for stability

Sigmoid Function

- Converts raw output → probability (0-1)
- Formula: $\sigma(z) = 1/(1 + e^{(-z)})$
- Threshold (usually 0.5)
  - ≥ 0.5 → Class 1
  - < 0.5 → Class 0

Date: __/__/20 Day: _____

How Logistic Regression Works

- Linear combination: $z = w \cdot X + b$
- Apply sigmoid: $p = \sigma(z) \rightarrow$ gives probability
- Predict class using threshold

$$p(x) = \frac{1}{1 + e^{-(w \cdot X + b)}}$$

Odds and Log-Odds

- Odds $= \frac{p(x)}{1 - p(x)} = e^z$
- Log-Odds (Logit):

$$\log\left(\frac{p(x)}{1 - p(x)}\right) = w \cdot X + b$$

Likelihood & Gradient

- Find weights (w, b) by maximizing likelihood of data
- Log-likelihood

$$\sum y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))$$

- Update parameters using Gradient Ascent:

$$\Delta w_j = \sum (y_i - p(x_i)) x_{ij}$$

Important Terms

- Independent Variables → input features
- Dependent Variable → target (categorical)
- Logistic Function → sigmoid
- Odds → ratio $p/(1-p)$
- Log-Odds (Logit) → log(odds)
- Co-efficient (weights) → effect of features
- Intercept (bias) → log-odds when features = 0
- MLE → estimates best parameters

Date: __/__/20__ Day: _____

Evaluation Metrics

- Accuracy = (TP + TN) / Total
- Precision = TP / (TP + FP)
- Recall (Sensitivity = TP / (TP + FN)
- F1 Score = 2 × (Precision × Recall) / (Precision + Recall)
- AUC-ROC → performance across thresholds
- AUC-PR → precision-recall trade-off

Linear Vs Logistic Regression

Linear Regression	Logistic Regression
Predicts continuous values	Predict categorical values
Regression problems	Classification problems
Best fit line	Best fit S-curve
Uses MSE	Using Log Loss (MLE)
Needs linear relationship	Doesnt need strict linearity
Output → continuous (price)	Output → 0/1 (pass/fail)

Cost Function in Linear Regression

Logistic Regression uses Log Loss (cross-Entrop Loss) Not MSE.

$$Cost(h\theta(x), y) = -[y\log(h\theta(x)) + (1-y)\log(1-h\theta(x))]$$

for all training examples

$$J(\theta) = -\frac{1}{m}\sum[y\log(h\theta(x)) + (1-y)\log(1-h\theta(x))]$$

Why not MSE?

- MSE → non-convex curve (many local minima)
- Log Loss → convex → gradient descent work better

Date: __/__/20 Day: _____

Decision Trees

- Introduction
- Types of Decision Tree
- Decision Tree → Regression
- Decision Tree → Classification

A Decision Tree is a supervised ML algorithm used for classification & regression
It has tree-like structure with
- Root Node → First split (feature test)
- Internal Node → Feature-based decisions
- Branches → Possible values of features
- Leaf Nodes → Final prediction

Decision Trees are popular because they are easy to interpret, flexible, and need little preprocessing

How it Works
- Dataset is split into subsets based on feature values
- Each internal node asks a question (feature test)
- Each leaf node gives a class label (classification) or a value (regression)
- Process continues until pure subsets are formed or stopping criteria are met
- Example (customer purchase)
  - Q1: Income > 50k? → No → "No Purchase"
  - If Yes → Q2: Age > 30?
    - No → "No purchase"
    - Yes → Q3: Previous Purchases > 0? → Yes → "Purchase", No → "No Purchase"

Date: __/__/20__ Day: _____

Attribute Selection Measures.

- Information Gain (Entropy-based)
  - Measures reduction in uncertainty after split

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$$

  - Entropy: Measures impurity (0 = pure, 1 = uncertain)

Example:

A split that separates all "Yes" and "No" perfectly has high Information Gain

- Gini Index
  - Probability of misclassifying a random sample
  - Formula

$$Gini(D) = 1 - \sum_{i=1}^{n} p_i^2$$

  - 0 = pure, 0.5 = impure
  - Faster to compute than Entropy

Decision Tree Algorithms

1- ID3: Uses Information Gain; works well for classification but overfits and struggle with continuous data.

2- C4.5: Uses Gain Ratio; handles continuous data; reduces overfitting

3- CART: Supports both classification (Gini) and regression (variance); binary splits with pruning

4- CHAID: Uses Chi-Square test; best for categorial data

Date: __/__/20 Day: _____

5. MARS: Handles non-linear regression using splines

6- Conditional Interference Trees → uses stational tests (Chi-square, F-test); less biased but slower

Real-life Example (Weather & Activities)

- Start with dataset → Outlook = Sunny, Cloudy, Rainy
- Split based on Outlook
- For Sunny → split again on Humidity (High → Swimming, Normal → Hiking)
- For Cloudy → Hiking
- For Rainy → Stay Inside

Implementation with Scikit-learn

Decision Tree Regression

```
from sklearn.tree import DecisionTreeRegressor
reg = DecisionTreeRegressor(max_depth=4, random_state=42)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
```

→ Predicts continuous values (e.g house prices)

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=1)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

→ Predicts categorical classes (e.g Iris dataset)

→ Hyperparameter tuning can be done using Grid Search CV.

Date: __/__/20__ Day: _____

Support Vector Machine (SVM)

- Introduction SVM
- SVM Hyperparameter Tuning - GridSearchCV
- Non-Linear SVM

What is SVM?

- SVM = Supervised ML algorithm (classification & regression)
- Finds the best boundary (hyperplane) b/w classes
- Goal: maximize margin → better generalization
- Examples: Spam vs Not Spam, Cat or Dog

Key Points

- Hyperplane: Decision boundary → $wx + b = 0$
- Support vectors: Nearest points to hyperplane
- Margin: Distance b/w hyperplane & support vectors
- Kernel: Maps non-linear data into higher dimension
- Hard Margin: No mistakes allowed
- Soft Margin: Allow small mistakes (better generalization)
- Hinge loss: Loss increases if misclassified or too close to margin
- Dual Problem: Uses Lagrange multipliers & kernel trick.

How SVM works

- Chooses hyperplane that maximizes margin
- Ignores outliers with soft margin
- Objective

Minimize $(1/\text{margin}) + \lambda * \text{penalty}$

- Penalty = Hinge Loss

Date: __/__/20 Day: _____

What if data is Not linearly Separable?

Use kernels

- Linear kernel → straight line
- Polynomial kernel → Polynomial curve
- RBF kernel → works on distances (very common)
- Sigmoid kernel → Similar to neural networks.

Example:

Circles cannot be separated with a line

kernel maps to higher dimension → becomes separable

Math of SVM

- Hyperplane: $w^T x + b = 0$
- Distance of point to hyperplane

$d = (w^T x + b) / ||w||$

- Prediction rule

$\hat{y} = 1$ if $w^T x + b \geq 0$

0 if $w^T x + b < 0$

Optimization Problem (Hard Margin)

minimize $\frac{1}{2} ||w||^2$

subject to $y_i(w^T x_i + b) \geq 1$

Soft Margin (with slack variables)

minimize $\frac{1}{2} ||w||^2 + C \sum \xi_i$

subject to $y_i(w^T x_i + b) \geq 1 - \xi_i$

$\xi_i \geq 0$

Dual Problem

- Maximizes Lagrage multipliers α_i
- Uses kernel $K(x_i, x_j)$
- Only support vectors ($\alpha_i > 0$) matter

$w = \sum \alpha_i * y_i * K(x_i, x) + b$

Date: __/__/20__ Day: _____

Types of SVM

- linear SVM: Works if data is linearly separable
- Non-linear SVM: Uses kernels for complex data

Advantages

- Works in high dimensions
- Handles non-linear data with kernels
- Robust to outliers (soft margin)
- Works for binary & multiclass
- Uses few support vectors → memory efficient

Disadvantages

- Slow for large datasets
- Needs parameters tuning (C, gamma, kernel)
- Sensitive to noise
- Less Interpretable
- Requires feature scaling

SVM Hyperparameters Tuning (GridSearchCV)

- Performance depends on C, gamma, kernel
- GridSearchCV finds best values via cross-validation

Steps:

1- Import libraries

```
import pandas as pd, numpy as np
from sklearn.metrics import classification_report
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC
```

2- Load datasets

```
cancer = load_breast_cancer( )
```

Date: __/__/20 Day: _____

3- Train - Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, test_size=0.3, random_state
    =101)
```

4- Without tuning

```
model = SVC()
model.fit (X_train, y_train)
print( classification_report( y_test, model.predict(X_test)))
```

5- GridSearchCV

```
from sklearn.model_selection import GridSearchCV
param_grid = { 'C' : [0.1, 1, 10, 100],
               'gamma' : [1, 0.1, 0.01, 0.001],
               'kernel' : ['rbf'] }
grid = GridSearchCV(SVC(), param_grid, refit= True, verbose=3)
grid.fit (X_train, y_train)
```

6- Best Model

```
print(grid.best_params_)
print(grid.best_estimator_)
```

7- Evaluate

```
print( classification_report ( y_test, grid.predict(X_test)))
```

- Accuracy improves (examples: 92% → 94%)

Non-Linear SVM

- Linear SVM = Straight line
- Non-Linear SVM = curves using kernels
- Examples: Circles or Moons dataset

Date: __/__/20__ Day: _____

Popular Kernels

- RBF: Best for circular / spherical data
- Linear: For simple separable data
- Polynomial = Handles curved patterns
- Sigmoid: Works like neural networks

Example 1 - RBF Kernel (Circular Data)

```
from sklearn.datasets import make_circles
X, y = make_circles(n_samples = 500, factor = 0.5,
                    noise = 0.05)
svm = SVC(kernel = 'rbf', C=1, gamma = 0.5)
svm.fit(X, y)
```

→ creates circular boundary

Example 2: Polynominal Kernel (Moons Data).

```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples = 500, noise = 0.1)
svm_poly = SVC(kernel = 'poly', degree = 3, C=1, coef0=1)
svm_poly.fit(X, y)
```

→ Smooth Curved Boundary

Linear Vs Non-Linear SVM

Feature	Linear SVM	Non-Linear SVM
Boundary	Straight line/hyperplane	Curved / Complex
Data	works if linear separable	works for non-linear
Kernel	Linear Only	RBF, Poly, Sigmoid
Speed	Faster	Slower
Example	Spam Detection	Image/handwriting

Applications

- Image Classification
- Bioinformatics
- NLP
- Fraud detection
- Speech recognition

Date: __/__/20 Day: _____

K-Nearest Neighbors (K-NN)

- Introduction to KNN
- Decision Boundaries in K-Nearest Neighbors (KNN)

What is KNN?

- Supervised ML Algorithm, most for classification
- KNN also works for regression
- Predicts by looking at K closest point (neighbors)
- Classification: Majority vote of neighbors
- Regression: Average of neighbors
- Lazy Learner: Doesn't train, just store data, work at Prediction time
- Non-parametric: No assumption about data distribution

How KNN works (Steps)

- Choose K (number of neighbors)
- Calculate distance (Euclideon, Manhattan, etc)
- Pick K nearest neighbors
- Classification: Majority class
- Regression: Average value

What is "K"?

- K = number of neighbors to check
- Example: If K = 3 → check 3 closest points → majority wins

Choosing K:

- Small K → complex boundaries → overfitting
- Large K → smoother boundaries → underfitting
- Cross-validation / Elbow Method → best K
- Odd → avoids ties

Date: __/__/20__ Day: _____

Distance Metrics

- Euclidean: Straight line distance

$$d(x, y) = \sqrt{\sum (x_i - y_i)^2}$$

- Manhattan (Taxicab): Grid-like distance

$$d(x, y) = \sum |x_i - y_i|$$

- Minkowski: Generalized distance

$$d(x, y) = \left(\sum |x_i - y_i|^p \right)^{1/p}$$

- $p=2 \rightarrow$ Euclidean, $p=1 \rightarrow$ Manhattan

Application of KNN

- Recommendation systems (movies/products)
- Spam detection
- Customer segmentation
- Speech recognition

Advantages

- Simple, easy to understand
- No training step
- Works for classification & regression
- Only two parameters: K and distance metric

Disadvantages

- Slow with large datasets
- Struggles with many features (curse of dimensionality)
- Sensitive to noisy data
- Can Overfit

Date: __/__/20 Day: _____

Decision Boundaries in KNN

- Decision boundary = line/curve separating different classes.
- Depends on
  - Value of K
  - Data distribution
  - Distance metric

Voronoi Diagram (for K = 1)

- Splits space into regions → each region belongs to nearest training point
- Boundaries = perpendicular bisectors b/w training points
- For 1-NN decision boundaries ≈ Voronoi diagram

Effect of Distance Metric

- Euclidean : Circular / elliptical boundaries
- Manthalton : Axis- aligned boundaries

Factor Affecting Boundaries

- Feature Scaling:
  - KNN is distance based → scaling matters
- Noise / Outliers:
  - Can distort boundaries
- Data Distribution:
  - Affects shape & accuracy

Date: __/__/20__ Day: _____

Naive Bayes

Introduction to Naive Bayes
Gaussian Naive Bayes
Multinomial Naive Bayes
Bernoulli Naive Bayes
Complement Naive Bayes

What is Naive Bayes?

Naive Bayes is a probabilistic ML algorithm for classification. It assumes features are independent give the class
Work very well in spam filtering, sentiment analysis, document classification

Key Idea

- Uses Baye's Theorem

$$P(y|X) = \frac{P(X|y) \cdot P(y)}{P(X)}$$

- Naive → assumes features don't affect each other
- Bayes → based on Bayes' theorem
- Features independently contribute to predictions

Example: Play Golf Dataset

Features: Outlook, Temperature, Humidity, Wind
Target: Play Golf (Yes/No)

- Priors

$$P(Yes) = 9/14, \quad P(No) = 5/14$$

- Example input: (Sunny, Hot, Normal, False)
- After calculation

$$P(Yes) \approx 0.756, \quad P(No) \approx 0.244$$

Prediction → Yes (Play Golf)

Date: __/__/20 Day: _____

How Naive Bayes Works

- Compute prior probabilities of classes
- Compute likelihood of features given classes
- Apply independence assumption

$$P(y|x_1, \dots, x_n) \propto P(y) \cdot \prod_{i=1}^{n} P(x_i|y)$$

- Choose class with highest probability

Assumptions

- Features are independent
- Continuous → Gaussian (Normal) distribution
- Discrete → Multinomial distribution
- All features equally important
- No missing data

Types of Naive Bayes

- Gaussian NB → for continuous values (bell-shaped Normal curve)
- Multinomial NB → for word frequencies (text classification)
- Bernoulli NB → for binary features (word present/absent)

Advantages

- Easy, Fast, efficient
- Works with few data points
- Handles many features well
- Strong with categorical/text data

Disadvantages

- Independence assumption often unrealistic
- Affected by irrelevant features
- Zero probability issue for unseen data (fix → Laplace smoothing)

Application: Spam filtering, Medical diagnosis, Credit Scoring

Date: __/__/20__ Day: _____

Gaussian Naive Bayes

Gaussian Naive Bayes is a type of Naive Bayes for continuous features that follow a Gaussian (Normal) distribution

It is fast, simple, works well with small datasets, and is widely used in practice

Math Behind GNB

It assumes:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Where

- $x_i \rightarrow$ feature value
- $\mu \rightarrow$ mean of feature for class y
- $\sigma \rightarrow$ standard deviation for that class

→ To classify: compute posterior probability for each class, choose the highest one.

Why Works Well for Continuous Data

- Many real-world features (age, income, height, test scores) ≈ normal distribution
- Handles continuous data better than plain Naive Bayes
- Common in spam filtering, medical diagnosis, house price prediction

Example: Petal Length Classification

Dataset

Petal Length	Class
1.4, 1.3, 1.5	0 (Setosa)
4.5, 4.7, 4.6	1 (Versicolor)

Date: __/__/20 Day: _____

Predict $x = 1.6$ cm

1- Mean & Variance

- Class 0 → $\mu = 1.4$, $\sigma^2 = 0.0067$
- Class 1 → $\mu = 4.6$, $\sigma^2 = 0.0067$

2- Likelihood (Gaussian PDF)

- $P(1.6 \mid C=0) \approx 0.247$
- $P(1.6 \mid C=1) \approx 0$

3- Multiply by Priors (0.5 each)

- $P(C=0 \mid x) \propto 0.1235$
- $P(C=1 \mid x) \propto 0$

prediction → class 0 (Setosa)

Advantage of Gaussian NB

Simple, fast, works with small data

Use Cases for GNB

Text classification, medical, Spam, finance

Multinomial Naive Bayes

MNB is a type of Naive Bayes algorithm best for discrete data like text.

It works well for text classification (spam detection, document categorization)

It models work counts and assumes words are independent of each other

How It Works

- Naive → assumes all words/features are independent

Date: __/__/20__ Day: _____

- Multinomial → focuses on how many times a word appears.
- Idea: Count how often words appear in each class (spam / not spam)
- Example: If "free" appears often in spam, it helps predict spam

Multinominal Distribution Formula

$$P(X) = \frac{n!}{n_1! n_2! \cdots n_m!} p_1^{n_1} p_2^{n_2} \cdots p_m^{n_m}$$

- n = total trials
- n_i = count of word i
- p_i = probability of word i

Parameter Estimation (MLE with Laplace smoothing)

$$\theta_{c,i} = \frac{count(w_i, c) + 1}{N + V}$$

- $count(w_i, c)$ = times word i appears in class c
- N = total words in class c
- V = vocabulary size

Bernoulli Naive Bayes

Bernoulli Naive Bayes (BNB) is a Naive Bayes variant used when data is binary (0/1, Yes/No, True/False)

It models features using the Bernoulli distribution and assumes features are independent

Common is spam detection, text classification sentiment Analysis

Date: __/__/20__ Day: _____

Math Behind BNB

for each feature x_i:

$$P(x_i|y) = P(i|y)^{x_i} \cdot (1 - P(i|y))^{(1-x_i)}$$

- x_i = feature (0 or 1 → absent or present)
- $P(i|y)$ = probability feature appears in class y

Bernoulli Distribution

- Only two outcomes: Success (1) or failure (0)

formula

$$f(x) = p^x \cdot (1-p)^{1-x}, \quad x = 0, 1$$

if $x = 1$ → value $= p$

if $x = 0$ → value $= 1-p$

Example Spam Classification

1- Vocabulary

{buy, cheap, now, limited, offer, meet, me} → $v = 10$

2- Binary Feature Matrix (presence = 1, absense = 0)

Each message becomes a row of 0/1 values

Example

M1 = [1, 1, 1, 0, 0, 0, 0, 0, 0, 0] (spam)

3- laplace Smoothing

$$P(w_i = 1|c) = \frac{count(w_i, c) + 1}{N_c + 2}$$

Where

- N_c = 2 (docs per class), so denominator = 4)

4- Word Probabilities

Spam class:

- $P(buy|Spam) = 0.75$

Date: __/__/20__ Day: _____

- P(cheap | Spam) = 0.5
- P(now | Spam) = 0.5
- P(limited | Spam) = 0.5
- P(offer | Spam) = 0.5
- Others = 0.25

Not Spam class

- P(now | Not Spam) = 0.5
- P(meet | Not Spam) = 0.5
- P(me | Not Spam) = 0.5
- P(let's | Not Spam) = 0.5
- P(catch | Not Spam) = 0.5
- P(up | Not Spam) = 0.5
- Others = 0.25

5- Classify "buy now"

Features vector → [buy = 1, now = 1]

- For Spam:

$$P(Spam|d) \propto 0.5 \cdot 0.75 \cdot 0.5 = 0.1875$$

- For Not Spam:

$$P(NotSpam|d) \propto 0.5 \cdot 0.25 \cdot 0.5 = 0.0625$$

Final → Classified as Spam

In Short:

- Gaussian NB → continuous features
- Multinomial NB → word counts
- Bernoulli NB → word presence/absense

Date: __/__/20 Day: _____

Complement Naive Bayes

CNB is a Naive Bayes varient designed for imbalanced datasets and text classification

It fixes bias towards majority classes → often words better than Multinomial Naive Bayes

Challange : Unbalanced Data

Example

- 95% = Not Fraud, 5% = Fraud
- Always predicting "Not Fraud" → 95% accuracy but 0 Fraud detection

→ CNB reduces this bias

How CNB works

- For each class, take complement counts (all other classes)
- Estimate probabilities from complement
- Normalize → valid probability distribution
- Classify → pick class with highest posterior

Formula

$$P(f|c) = \frac{\text{count}(f, \bar{c}) + \alpha}{\sum_{f'} \text{count}(f', \bar{c}) + \alpha |V|}$$

- $\text{count}(f, \bar{c})$ = frequency of feature f in complement of class c
- α = Laplace smoothing
- $|V|$ = vocabulary size

Limitations

- Independence assumption (same as other NB models)
- Best for descrete features (needs preprocessing for continuous)
- Bias in balanced datasets → may hurt accuracy vs MNB

Date: __/__/20__ Day: _____

Random Forest (Bagging Algorithm)

Introduction to Random forest
Random Forest Classifier
Random Forest Regression
Hyperparameter Tuning in Random Forest

What is Random Forest?

Random Forest is an ensemble ML Algorithm using many decision trees to improve predictions. Trees use random parts of data and features, then combine results:

- Classification: majority voting
- Regression: average of predictions

It reduces errors and avoids overfitting

How Random Forest Works

- Create Many Trees: Each tree trains on a random data sample

- Pick Random Forest: Features: Each tree considers only some features when splitting :

- Make predictions: Each tree predicts independently

- Combine Predictions:
  - Classification → majority vote
  - Regression → average of all tree predictions

- Why its Works: Randomness prevents overfitting and improves accuracy.

Date: __/__/20__ Day: _____

Key Features

- Handles Missing Data: Can Work with incomplete datasets
- Feature Importance: Shows which features matter most
- Works with Big Data: Handles large Datasets efficiently
- Multiple Tasks: Works for classification & regression

Assumptions

- Each tree predicts independently
- Trees use random data & features
- Enough data ensures trees learn different patterns
- Combining predictions improves accuracy

Advantages

- Accurate predictions, even on big datasets
- Handles missing data,
- No need for normalization
- Reduces overfitting by combining trees

Limitations

- Computationally expensive with many trees
- Harder to interpret than a single decision tree

Random Forest Classifier

Random Forest combines many dicision trees to make more accurate predictions

- Classification: predicts categories (labels) by majority voting
- Regression: predicts numbers by averaging

Date: __/__/20__ Day: _____

How Random Forest Classifier Works

- Bootstrap Sampling: Random rows (with replacement) train each tree.
- Random Feature Selection: Each tree uses a random set of features
- Built Decision Trees: Tree splits data using the best features
- Make predictions: Each tree predicts independently
- Majority Voting: Final prediction is what most tree choose.

Benefits

- Handles large datasets & high-dimensional data
- Reduces overfitting compared to a single tree
- Works well noisy or categorical data

Important Parameters

Parameter	Meaning
n_estimators	Number of trees
max_depth	Max depth of each tree
max_features	Features considered at each split
criterion	Split quality measure ('gini')
min_samples_split	Min samples to split a node
min_samples_leaf	Min samples at a leaf node
bootstrap	Use bootstrap sampling (True/False)

Random Forest Regression

Random Forest is a supervised learning method using many decision trees to predict values more accurately.

Date: __/__/20__ Day: _____

- Regression: predicts continuous numbers by averaging tree predictions
- Reduces variance & improves stability compared to a single decision tree

How it Works

- Bootstrap Sampling: Random rows (with replacement) for each tree
- Feature Sampling: Each tree uses a random subset of features
- Built Trees: Trees learn patterns independently
- Prediction: Each tree predicts a value
- Aggregation: Final prediction = average of all tree predictions.

Why it Works:

Averaging multiple trees reduces overfitting & variance, giving stable & accurate predictions.

Applications

- Predict continuous values: house prices, salaries etc
- Detect risk factors in health or finance
- Analyze high-dimensional data
- Capture complex relationships b/w features & target

Advantages

- Handles non-linear data well
- Reduces overfitting
- Robust to outliers
- Works well with large datasets
- Handles missing data
- No need for feature scaling

Date: / /20 Day:

Disadvantages

- Computationally expensive for large trees
- Harder to interpret than simple models
- Memory intensive
- Can overfit noisy data
- Sensitive to imbalanced data

Random Forest Hyperparameter

Random Forest is a robust model for classification & regression. Tuning its hyperparameters improves accuracy & reduces overfitting

Key Hyperparamters

- n_estimate
  - Number of trees, More trees → better accuracy, slower training
  - 100 - Default

- max_features
  - Max Features considered per split (sqrt, log2, None)
  - Sqrt - Default

- max_depth
  - Max depth of each tree. Too shallow → underfit, 100 -deep → overfit
  - None - Dedault

- max_leaf_nodes
  - Max number of leaf nodes pre tree
  - None - Default

Date: __/__/20__ Day: _____

- max_samples:
  - How many samples per tree
  - None - Defoult

- min_samples_split
  - Min samples to split a node
  - 2 - Default

Introduction to Ensemble Learning

Ensemble learning uses multiple small models instead of one. Each model may be weak, but combined, they give better, more accurate results.

→ like asking a group for advice instead of one person

Types of Ensemble Learning

- Bagging (Bootstrap Aggregating)
  - Train models independently on random subset of data
  - Combine results: voting (classification) or averging (regres)
  - Reduces variance & pervents overfitting

- Boosting
  - Train models sequentially; each new model corrects errors of previous ones
  - Final prediction = weighted combination
  - Reduces bias, improves accuracy

- Stacking
  - Train different types of models
  - Their predictions are inputs to a meta-model
  - Meta-model combines results for better performance

Date: __/__/20__ Day: _____

Benefits of Ensemble Learning

- Reduces Overfitting
  - Aggregating models reduces individual errors
- Improves Generalization
  - Works better on unseen data
- Higher Accuracy:
  - Multiple models > single model
- Robust to Noise:
  - Averaging reduces effect of noisy data.
- Flexible:
  - Works with trees, neural nets, SVMs etc.
- Bias-Variance Tradeoff:
  - Bagging ↓ variance, Boosting ↓ bias

Popular Ensemble Techniques

- Random Forest → Bagging
- Random Subspace → Bagging
- Gradient Boosting (GBM) → Boosting
- XGBoost → Boosting
- AdaBoost → Boosting
- CatBoost → Boosting

Bagging Classifier

Bagging (Bootstrap → Aggregating) trains multiple base models independently on random subsets of the training data.

- Bootstrap Sampling
  - Randomly select data with replacement → some appear multiple times, some excluded.

- Classification → final result = majority vote

Date: __/__/20 Day: _____

- Regression → final result = average
- Works with decision trees, SVMs, neural networks
- Reduces overfitting & improves accuracy.

How Bagging Works

- Bootstrap Sampling:
  - Create multiple subsets from the original data with replacement
  - Example

original : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sample1: [2, 3, 3, 5, 6, 1, 8, 10, 9, 1]
Sample2: [1, 1, 5, 6, 3, 8, 9, 10, 2, 7]

- Train Base Models
  - Train each subset on an independent base learner
  - Training happens in parallel.

- Aggregation
  - Classification → majority vote
  - Regression → average

- Out-of-Bag (OOB) Evaluation:
  - Use excluded samples as natural validation

Applications

- Fraud Detection: Better detection via multiple models
- Spam Filtering: Combine predictions from different samples
- Credit Scoring: More robust predictions
- Image Classification: Reduce overfitting
- NLP: Improve text classification & sentiment analysis.

Date: / /20 Day:

Advantages

- Better Accuracy
- Reduces variance
- Robust
- Flexible

Disadvantages

- No bias reduction
- Possible overfitting
- Limited gain for stable models
- Hyperparameter sensitive

AdaBoost (Adaptive Boosting)

AdaBoost is a boosting ensemble method that combines multiple weak classifiers sequentially to make a strong classifier. Each new model focuses on correcting errors of the previous one

Think:

Like a teacher focusing on weak students to improve class performance.

How AdaBoost Works

- Initial Model (B1)
  - All data points = equal weight
  - Train first weak classifier
  - Some points misclassified

Date: __/__/20__ Day: _____

- Adjust Weights (B2)
  - Misclassified points → higher weights
  - Train new classifier focusing on errors
  - Fewer misclassified points

- Further Adjustment (B3)
  - Again, misclassified points get higher weight
  - Train next weak model
  - Errors reduce.

- Final Strong Model (B4)
  - Combine all weak models → strong ensemble
  - Higher accuracy than any single model

Key points
- Iteratively focuses on misclassified points
- Reduces bias and variance
- Sensitive to noisy data & outliers
- Best for classification tasks

Popular Boosting Algorithms
- Gradient Boosting
- XGBoost
- CatBoost

Advantages
- Higher Accuracy
- Reduces Overfitting
- Handles Imbalanced Data
- Better Interpretability

Date: __/__/20__ Day: _____

Module 3: Unsupervised learning

- Clustering
- Association Rule Mining
- Dimensionality Reduction

Clustering

Clustering is unsupervised ML that groups similar data points into clusters based on characteristics, without labels

- Goal: Find natural groupings in unlabeled data
- How: Assign points to clusters using similarity/distance (Euclidean, Cosine etc)
- Output: Each point gets a cluster ID

Example: Group customers by shopping habits → use clusters for marketing, recommendation, segmentation

Type of Clustering

- Hard Clustering
  - Each point belongs to only one cluster
  - Use: Market segmentation, document grouping
  - Limitation: Cannot show overlap

Example

Data Point	Hard Cluster	Soft Cluster
A	Cluster 1	0.91/0.09
B	Cluster 2	0.30/0.70

Soft Clustering:

- Points can belong to multiple clusters with probabilities

Date: __/__/20 Day: _____

- Use: Overlapping classes, customer personas, medical data
- Benefits: Shows ambiguity in data

Types of Clustering Methods

- Centroid-based (Partitioning)
  - Clusters around centroids (mean/medoid)
  - Algorithms: K-means, K-medoids
  - Pros: Fast, Scalable, Simple
  - Cons: Need K, sensitive to outliers, not for non-spherical clusters

- Density-based (Model-based)
  - Clusters = high-density regions separated by low-density areas
  - Algorithms: DBSCAN, OPTICS
  - Pros: Handles arbitrary shapes, noise, no cluster count needed
  - Cons: Hard to choose parameters (epsilon, min points)

- Connectivity-Based (Hierarchical)
  - Builts nested clusters → dendrogram
  - Approaches: Agglomerative (bottom-up) Division (top-Down)
  - Pros: Visualise hierarchy, no need for cluster count
  - Cons: Computationally heavy, irreversible merges/splits

- Distribution-Based
  - Data assumed from mixture of distributions
  - Algorithm: Gaussian Mixture Model (GMM)
  - Pros: Flexible shapes, probabilistics membership
  - Cons: Need number of distributions, sensitive to initialization, heavier computation

Date: __/__/20__ Day: _____

- Fuzzy Clustering
  - Points can particaully belong to multiple clusters
  - Algorithm: Fuzzy C-Means
  - Pros: Captures ambiguity, good for overlapping clusters
  - Cons: Fuzziness parameter tricky, heavier computation

Use Cases of Clustering

- Customer Segmentation
- Anomaly Detection
- Image Segmentation
- Recommendation Systems
- Market Basket Analysis

Type of Clustering

- Centeroid - Based Methods
- Distribution - Based Methods
- Connectivity Based Methods
- Density - Based Methods

Centroid - Based Methods

- K-Means Clustering
- Elbow Method for optimal value of K in Kmeans
- K Means ++ Clustering
- K. Mode clustering
- Fuzzy C-Means (FCM) clustering

- K-Mean Clustering

K-Means is an unsupervised algorithm used to group data into K clusters based on similarity

Date: __/__/20__ Day: _____

Working

- Choose number of clusters K
- Randomly initialize K centroids (cluster centers)
- Assign each data point to the nearest centroid
- Recalculate centeroids as the mean of all assigned points
- Repeat until centroids stop changing

Objective

Minimize the Within Cluster Sum of Square (WCSS)

$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

Where:

- x_j = data point
- μ_i = centroid of cluster i

Pros:

- Fast, Simple, scalable
- Works well for numerical data

Cons:

- Must specify K
- Sensitive to Outliers
- Poor results if centroids start badly

- Elbow Method (Finding best K)

The Elbow Method helps decide how many cluster(K) to use in K-Means

Steps:

- Run K-Means for several K values (e.g 1 to 10)
- Record WCSS (inertia) for each K
- Plot K vs WCSS

Date: __/__/20__ Day: _____

- The point where the graph starts to bend ("elbow") is the optimal K.

Idea:

Before elbow → adding clusters reduces WCSS sharply.
After elbow → improvement slows → unnecessary extra clusters

- K-Meons ++ Clustering

K-Meonss ++ improves the initialization of centroids for better accuracy and faster covergence.

Steps:

- Pick the first centroid randomly
- For the next centroids
  - Compute each point's distance from nearest chosen centroid
  - Choose next centroid proportionally to squared distance
- Run normal K-Means

Benefits

- Avoids poor random starts
- Produces more stable and accurate results
- Faster convergence

- K-Modes Clustering

K-Models is a variation of K-Means for categorical data (like colors, gender, etc) since "mean" & "Eudideon distance" don't work with categories.

Modifications

- Replace mean → mode (most frequent value)
- Replace Eudideon distance → Hamming distance (count of mismatches)

Date: __/__/20 Day: _____

Dissimilarity

$$D(X, Q) = \sum_{i=1}^{m} \delta(x_i, q_i)$$

Where:

$$\delta(x_i, q_i) = \begin{cases} 0, & x_i = q_i \\ 1, & x_i \neq q_i \end{cases}$$

Example

If clustering people by Gender, Color, Type
K-Modes groups those with similar categorical traits

Pros:

- Handles categorical features
- Simple & Fast

Cons:

- Doesn't Handle numerical data well
- Needs K defined in advance

- Fuzzy C-Means (FCM) Clustering

Fuzzy C-Means is a soft clustering algorithms each data point can belong partially to multiple clusters with different memberships degees.

Concepts

Instead of hard assignment (like K-Means), FCM assigns a membership value b/w 0 & 1 to each cluster

Date: __/__/20__ Day: _____

Steps:

- Initialize cluster centers
- Calculate membership matrix (U) → how strongly each points belongs to each cluster
- Update cluster centers based on weighted average

$$c_i = \frac{\sum_{j=1}^{n} u_{ij}^{m} x_j}{\sum_{j=1}^{n} u_{ij}^{m}}$$

- Repeat until membership values stabilize

Where:

- u_{ij}: membership of points j in cluster i
- m: fuzziness parameter (controls overlap, usually 2)

Pros

- Handles overlapping clusters
- Models uncertainty in data

Cons:

- More computationally expensive
- Needs tuning of fuziness parameter m.

Distribution - Based Methods

- Gaussian mixture models
- Expectation - Maximization Algorithm
- Dirchlet process mixture models (DPMMs)

- Gaussian Mixture Model (GMM)

What is GMM?

- A probabilistics clustering model that assumes data is made from a mixture of several Gaussian distribution

Date: __/__/20 Day: _____

- Unlike K-Means (which gives hard labels), GMM does soft clustering - each point has a probability of belong to each cluster

Key Formula

For a data point x_n:

$$P(z_n = k | x_n = \frac{\pi_k N(x_n | \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k N(x_n | \mu_k, \Sigma_k)}$$

Where:

- π_k: probability (weight) of cluster k
- μ_k: mean of cluster k
- Σ_k: covariance (shape/spread) of cluster k

Total likelihood of point x_n:

$$P(x_n) = \sum_{k=1}^{K} \pi_k N(x_n | \mu_k, \Sigma_k)$$

How it works (Expectation - Maximization Algorithm)

- Initialize: Randomly guess means, covariances, & weights
- E-step: Calculate probabilities that each point belongs to each cluster
- M-step: Update means, covariances, and weights using those probabilities
- Repeat until parameters stop changing

Cluster Shape

- Mean (μ): Center of cluster
- Covariance (Σ): Shape/orientation
- Unlike K-Means (spherical), GMM allows elliptical clusters it is more flexible

Date: __/__/20__ Day: _____

Use Cases

- Clustering (market segmentation, genetics)
- Anomaly detection (fraud, medical errors)
- Image segmentation
- Density estimation (data distribution modeling)

Pros & Cons

- Flexible shapes
- Probabilistics (soft) assignments
- Interpretable parameters
- Sensitive, to Initialization
- Computationally heavy
- Needs number of clusters pre-set

- Expectation - Maximization (EM) Algorithm

What is EM?

An Iterative optimization method used to find parameters of probabilistic models when some data is missing or hidden.

Two Steps

- E-Step (Expectation)

Estimate hidden data (probabilities of belonging to clusters)

- M-Step (Maximization)

Update parameters (means, variances, weights) to maximize likelihood

∴ Repeat until convergence → when log-likelihood stops improving.

Date: __/__/20__ Day: _____

Important Terms

- Latent Variable: Hidden variable (like cluster label)
- Likelihood: Probability of observed data given model
- Log-likelihood: Easier form of likelihood for computation
- MLE (Maximum Likelihood Estimation): Find parameters that maximize likelihood
- Posterior Probability: Probability of hidden variables given observed data

EM Algorithm Steps

- Start with guessed parameters
- E-Step: Compute expected cluster assignments
- M-Step: Update parameters using those assignments
- Repeat until model stabilizes

Applications

- Gaussian Mixture Models
- Hidden Markov Models
- Image Segmentation
- NLP (topic modeling, translation)
- Missing data imputation

Benifits

- Always improves likelihood
- Works with missing data
- Flexible & general

Disadvantages

- Slow convergence
- Can get stuck in local optima
- Needs good initial guesses

Date: __/__/20__ Day: _____

- Dirichlet Process Mixture Model

What is DPMM?

- An advanced version of GMM that automatically decides the number of clusters
- Based on dirichlet Process (DP) → a nonparametric Bayesian approach (no fixed number of clusters)

Key Concepts

- Beta Distribution

Models probability for two outcomes using parameters α and β

- Dirichlet Distribution

A multi-category version of Beta distribution
Each probability $p_i \geq 0$ & $\sum p_i = 1$

- Dirichlet Process (DP)

$$DP(\alpha, G_0)$$

- α → controls cluster diversity
- G_0 → base distribution (prior for cluster parameters)

Stick-Breaking Process

- Image in breaking a stick of length 1 into random parts using Beta $(1, \alpha)$
- Each piece represents a cluster's weight (probability)
- Continue until total ≈ 1

How DPMM works

- Randomly assign points to clusters
- For each point:
  - Compute probability of joining existing clusters or forming a new one

Date: __/__/20 Day: _____

- Continue until cluster assignments stop changing

It extends GMM using DP to allow infinite possible clusters — model automatically decides how many are needed

Formula (Simplified)

- Probability of joining existing cluster:
  proportional to number of points in that cluster
- Probability of starting new cluster:
  proportional to α (concentration parameter)

Advantages over GMM / K-Means

- Automatically finds cluster count
- Probabilistic and flexible
- Handles overlapping data
- Works well with complex datasets

Applications

- NLP (topic modeling)
- Bioinformatics
- Computer Vision
- Finance & Marketing segmentation

Connectivity based Methods

- Hierarchical clustering
- Agglomerative clustering
- Divisive clustering
- Affinity propagation

Hierarchical Clustering

What is Hierachical clustering?

It is an unsupervised learning technique used to

Date: __/__/20 Day: _____

group similar data points by building a hierarchy (tree-like structure) of clusters

Unlike K-Means, you don't need to predefine K (number of clusters)

It works either by

- Merging small clusters step-by-step (Agglomerative) or
- Splitting one large cluster into smaller ones (Divisive)

A dendogram is used to visualize this hierarchy

Simple Example

Imagine fruits with different weights

apple (100g), banana (120g), cherry (50g), grape (30g)

Steps

- Each fruit starts as its own cluster
- Closest ones merge → grape & cherry fruit
- Then apple & banana merge
- Finally, both clusters merge into one

Dendogram (Cluster Tree)

- It shows how and when clusters merge
- Bottom = each point separately
- Maing up = groups start merging
- Shorter line = more similarity b/w clusters

Type of Hierarchical clustering

- Agglomerative (Bottom-up) → start small → merge
- Divisive (Top-Down) → start Big → split

Hierarchical Agglomerative Clustering (HAC)

Bottom up Approach

- Each data point starts alone

Date: __/__/20 Day: _____

- The algorithm merges closest clusters step-by-step until only one remains
- Distance b/w clusters can be calculated using linkage methods (Word, Average, Complete, Single)

Steps:

- Start with all points as separate clusters
- Compute pairwise distances
- Merge two closest clusters
- Update distance matrix
- Repeat until all are merged
- Plot dendogram to visualize

Hierarchical Divisive Clustering

Top-Down approach

- Start with all data in one cluster, then recursively split into smaller clusters

Steps:

- Start with one big cluster
- Split into two using a method like K-Means
- Repeat until each data point is alone or desired clusters reached

Distance (linkage) Methods

When merging clusters, distances can be measured by

Method	Meaning
Min (Single Link)	Shortest distance b/w clusters
Max (complete Link)	Longest distance b/w clusters
Average Link	Mean distance b/w all pairs
Ward's Method	Minimizes total variance increase

Date: __/__/20 Day: _____

Agglomerative Vs Divisive Clustering

Feature	Agglomerative	Divisive
Approach	Bottom-up	Top-Down
Start	Each point separate	All points together
Process	Merge closest	Split largest
Complexity	$O(n^3)$ (expensive)	Slightly less but recursive
Handling Outliers	Better	Can split inefficiently
Interpretability	Easy (via dendogram)	Harder
Libraries	Available in sklearn	Rarely implemented
Use Case	Common (image, doc	Less Common

Affinity Propagation (AP)

What is it?

- An unsupervised clustering algorithm that finds clusters by message passing b/w data points
- Unlike K-Means, you don't set the number of clusters — it's found automatically

Key Parameters

- Preference
  - Controls how many points become cluster centers
  - Higher = more clusters
- Damping Factor:
  - Prevents oscillation
  - Helps the algorithm coverage

Mathematical Concept

- Similarity Matrix (S)

Measures how similar two points are

$$S(i, j) = -\|x_i - x_j\|^2$$

Diagonal values $S(i,i)$ are the prefrences (how likely a point is to be a center)

Date: __/__/20 Day: _____

• Responsibility (R)

Shows how suitable point (k) is an exemplar for point i

$$r(i,k) = s(i,k) - \max_{k' \neq k} \left[a(i,k') + s(i,k') \right]$$

• Availability (A).

Shows how appropriate it is for i to choose k as its ~~example~~ exemplar

$$a(i,k) = \min\left(0, r(k,k) + \sum_{i' \neq k} \max(0, r(i',k))\right)$$

Diagonal case

$$a(k,k) = \sum_{i' \neq k} \max(0, r(i',k))$$

• Convergence

Update R and A iteratively until stable

if $r(i,i) + a(i,i) > 0 \rightarrow$ point i is an exemplar

Process Visualization

• Responsibility Messages: Indicate how much each point wants another as its center

• Availability Messages: Indicate how suitable it is to be chosen as a center

∴ Both pass back and forth until clusters stabilize

Advantages

• No need to predefine K

• Handles different cluster sizes/shapes

• Cluster centers are actual data points

• Works well for complex data

Limitation: Slow for large datasets, sensitive to parameter tuning

• High memory usages, Might not coverage without damping

Date: __/__/20__ Day: _____

Density Based Methods

- DBSCAN (Density-Based Spatial Clustering of Application with Noise)
- OPTICS (ordering Points To Identify the Clustering Structure)

DBSCAN

- DBSCAN is a density-based clustering algorithm
- It groups together points that are closed (dense regions) and labels isolated points as noise
- Works well for arbitrary-shaped clusters & outliers Unlike K-Means (which assumes round/spherical clusters)

DBSCAN can handle

- Arbitrary shapes (not just circular)
- Noise & outliers (marks them as -1)
- Clusters of Different sizes

Key Parameters

- eps (ε) → Radius of neighbourhood around a point
  - Points within this distance are neighbors
  - Choosing ε:
    - Too small → Many points marked as noise
    - Too large → Clusters may merge
  - Use K-distance graph to pick ε

- MinPts → Minimum number of points required in the ε-radius to form a cluster
  - Rule of thumb: MinPts $\geq D+1$ (D = no. of features)
  - Common default: MinPts = 3 or higher.

Types of Points in DBSCAN

Date: __/__/20 Day: _____

Type	Description
Core Point	has ≥ MinPts points within eps
Border Point	Lies near a core point but doesnt have enough neighbor
Noise Point	Doesnot belong to any cluster

Working of DBSCAN

- Identify core points : Check each points neighbors with in eps
- Form Clusters: expand clusters from core points
- Density Connectivity: If a point can be reached through a chain of core points → belongs to the same cluster.
- Label Noise: Points not in any cluster are labeled as noise (-1)

DBSCAN vs K-Means

Feature	DBSCAN	K-Means
Clusters	Auto-detected	Must predefine K
Shape	Any Shape	Spherical
Noise Handling	Detects noise	Sensitive to noise
Parameters	eps, MinPts	Only K
Work Best For	Complex, irregular data	Well separated, round cluster

OPTICS

- OPTICS is an advanced density-based algorithm, similar to DBSCAN
- Unlike DBSCAN, it handles varying densities and creates a Reachability Plot instead of fixed clusters
- Useful for datasets with uneven densities

Date: __/__/20__ Day: _____

Key terms

Core Distance:

- Minimum distance for a point to be a core

Reachability Distance:

- How far one point is from another

Reachability Plot

- A graph of reachability distances for all points
- Valleys = clusters (dense regions)
- Peaks = noise or sparse regions
- More information than DBSCAN, shows hierarchical density structure

Working Steps

- Pick a random point
- Check if it has $\geq$ MinPts within eps → mark as core
- Compute reachability distance for neighbors
- Process Points in order of increasing reachability
- Create reachability plot — valleys = clusters, peak = noise

When To Use OPTICS

- Datasets with varying densities
- When you need a hierarchical view of clusters
- When DBSCAN fails due to single global eps value

Date: __/__/20 Day: _____

Dimensionality Reduction

- Principal Component Analysis (PCA)
- t-distributed Stochastic Neighbor Embedding (t-SNE)
- Non-negative Matrix Factorization (NMF)
- Independent Component Analysis (ICA)
- Isomap
- Locally Linear Embedding (LLE)

Principal Component Analysis (PCA)

PCA is a dimensionality reduction technique in data analysis and machine learning. It reduces the number of features while keeping the most important information.

PCA creates new features (principal components) that don't overlap, and the first few keep most of the variance (important differences)

Why Use PCA?

- Removes redundancy
- Improves computationally efficiency
- Makes high-dimensional data easier to visualize & analysis

How PCA Work?

PCA transform original data into principal components using linear algebra

- Finds eigenvectors (directions) & eigenvalues (importance) from the covariance matrix
- Selects top components (highest eigenvalues) to simplify the data

→ Prioritizes directions with most variation - most information

Date: __/__/20__ Day: _____

Analogy:

Imagine a messy cloud of data points. PCA finds the best angles to view it, keeping the main patterns

Step-by-Step PCA

Step 1: Standardize Data

Different features have different scales (e.g height vs age). Standardize so each feature has

- Mean = 0
- Std = 1

$$z = \frac{x - \mu}{\sigma}$$

where

- μ = mean of features
- σ = standard deviation

Step2: Convariance Matrix

Shows how Features relate (increase/ decrease together)

$$\text{cov}(x_1, x_2) = \frac{\sum_{i=1}^{n} (x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2)}{n-1}$$

- Covariance can be positive, negative or zero

Step 3 Principal Components

- PC1:
- PC2
- Computed from eigenvectors of covariance matrix
- Eigenvalues measure importance

$$Ax = \lambda x$$

- Eigenvector = stable direction
- Eigenvalues = rank directions by variance

Date: __/__/20__ Day: _____

Step 4: Select Top Components & Transform
- Choose top K components (e.g 95% variance)
- Project original data onto these components
- Reduces features while keeping important patterns

Example: Transform 2D data (Radius, Area) → 1D along PC1 (most variance)

PCA Implementation in Python

Advantages of PCA
- Handles multicollinearity
- Reduces noise
- Data Compression → less storage, faster processing
- Helps detect outliers

Disadvantages of PCA
- Hard to interpret new components
- Sensitive to data scaling
- Possible information loss if too few components
- Assumes linear relationships
- Computationally heavy for very large data
- Risk of overfitting with small datasets

Association Rule
- Apiori algorithm
- Implementing apriori algorithm
- FP-Growth (Frequent Pattern - Growth)
- ECLAT

T-distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a non-linear dimensionality reduction algorithm used to visualize high-dimensional data in

Date: __/__/20 Day: _____

2D or 3D

Unlike PCA (linear), t-SNE focuses on preserving local structure — keeping similar data points close together

Purpose

Dimensionally reduction simplifies complex datasets by combining related features → helps in better visualization, faster computation & easier analysis

Working

- t-SNE finds how similar two data points are in high-dimensional space
- It converts these similarities into conditional probabilities (the likelihood that one point is near another)
- Then it maps them to a low-dimensional space while preserving local relationships

Advantages

- Best for Visualization
- Preserves Local Structure
- Non-linear
- Good Cluster Separation

Disadvantages

- Slow & Computationally Heavy
- Non-determine
- Not Scalable
- Only for Visualization
- Loses Global Structure

Date: __/__/20 Day: _____

Non-Negative Matrix Factorization (NMF)

NMF is a dimensionality reduction technique that breaks a large dataset (matrix) into smaller meaningful parts, keeping all values non-negative.
It helps extract useful hidden features from complex data for easier analysis

Matrix Representation

For a matrix A $(m \times n)$ with all values ≥ 0
NMF breaks it into two matrices

$$A_{m \times n} \approx W_{m \times k} \times H_{k \times n}$$

where

- A: Original data matrix
- W: Features (basis) matrix
- H: Coefficient (weight) matrix
- K: Reduced dimension ($K \leq \min(m, n)$)

Each data point in A is a combination of non-negative features from W

Intuition Behind NMF

- Simplifies complex data into few meaningful patterns
- Each column of $A \approx$ combination of features in W
- Data = sum of meaningful parts (non-negative)
- Example: In face recognition
  - W = eye, nose, mouth (features)
  - H = how much each feature contributes to an image

Working Steps

- Intialization: Start with random non-negative W & H
- Iterative Update: Adjust W and H to minimize the difference b/w A & $W \times H$

Date: __/__/20__ Day: _____

- Stopping Criteria
  - Error stops decreasing or
  - Reaches max iterations

Common Optimization Methods:
- Multiplicative Update Rule
- Alternating Least Square (ALS)

Real life Example
Image Processing
- A : Image pixel matrix
- W : Facial features (eyes, nose, mouth)
- H : Strength / weight of each feature in image

NMF → breaks an image into parts + shows how much each part contributes

Application of NMF
- Image Processing
- Text Mining / NLP
- Spectral Analysis
- Bioinformatics

Independent Component Analysis (ICA)
ICA is a signal separation technique that separates mixed signals into their independent, non-Gaussian components
It finds a linear transformation that makes output signal statistically independent
Used in audio, image, biomedical signal processing to isolate original source (like separating voices in a recording)

Date: __/__/20 Day: _____

Concept of Statistical Independence

Two variables X and Y are independent if knowing one does not affect the probability of the other

$$P(X \cap Y) = P(X) * P(Y)$$

Key Assumptions of ICA

- The source signals are independent of each other
- The sources are non-Gaussian

These assumptions help ICA separate mixed signals something PCA cannot do

Mathematical Representation

Let:

- $X = (x_1, \dots, x_m) \rightarrow$ observed mixed signals
- $S = (s_1, \dots, s_n) \rightarrow$ hidden (true) source signals
- $W \rightarrow$ transformation (unmixing) matrix

$$S = W \times X$$

Goal $\rightarrow$ find W that makes components of S independent

Cocktail Party Problem (Example)

Imagine a room with n speakers and n microphones

- Each mic records a mixture of all voices
- ICA separates these mixed signals into individual voices

$$[X_0, X_2, \dots X_n] \Rightarrow [Y_1, Y_2, \dots, Y_n]$$

Where X = mixed signals & Y = independent separated signals

Date: __/__/20__ Day: _____

PCA vs ICA

Feature	PCA	ICA
Goal	Reduce dimensions	Separate mixed signals
Type of Components	Principal Components	Independent Components
Focus	Maximize variance	Maximize independence
Orthogonality	Yes	No
Independence	No	Yes
Use Case	Dimensionality reduction	Signal separation

Advantage

- Separates mixed Signals
- Non-parametric
- Unsupervised
- Feature extraction

Disadvantages

- Assumes non-Gaussian sources
- Assumes linear mixing
- Computationally expensive

Isomap (Isometric Mapping)

Isomap is a dimensionality reduction techniques that reduces high-dimensional data into a lower dimension while keeping its original shape & structure

Unlike PCA, Isomap can handle non-linear and curved data (like, images, speech etc)

Manifold Learning (Base Concept)

- High-dimensional data lies on a low-dimensional surface called a manifold.
- Example: A 2D sheet twisted in 3D is still 2D at its core

Date: __/__/20__ Day: _____

Types of Distances

- Euclidean Distance:

  Straight-line distance b/w two points

- Geodesic Distance:

  Shortest path along the manifold surface (true distance in curved space)

Steps to Isomap

- Calculate Pairwise Distance

  Find Euclidean distances b/w all data points

- Find Nearest Neighbors

  For each point, pick its closest data points

- Built a Neighborhood Graph

  Connect each point with its neighbors

- Compute Geodesic Distances

  Use algorithms like Floyd-Warshall to find Shortest paths on the graph

- Dimensionality Reduction:

  Move points into lower-dimensional space (e.g 2D) while keeping distance relationships

Advantages

- Captures non-linear patterns (unlike PCA)
- Preserves global structure of Data
- Finds a globally optimal low-dimensional representations

Disadvantages

- Slow for large Datasets
- Sensitive to parameters (like no. of neighbors)
- Struggles with Complex manifolds (holes, disconnected shapes)

Date: __/__/20__ Day: _____

Locally Linear Embedding (LLE)

What is Swiss Roll Reduction?

- Swiss Roll = A 3D non-linear dataset shaped like a rolled-up sheet
- Goal: Project it into 2D or 1D while keeping its true curved structure
- Used to test non-linear dimensionality reduction algorithms
- Linear methods like PCA fail because the data lies on a 2D manifold inside 3D space
- Hence, LLE (a non-linear method) works better

Understanding the LLE Algorithm

Locally Linear Embedding (LLE) is a manifold learning algorithm for non-linear dimensionality reduction.

It assumes:

Each data point & its nearby points lie on a locally linear patch of a curved surface

Steps in LLE

- Built a Neighborhood Graph
  - Connect each point to its nearest neighbors (captures locally geometry)
- Compute Reconstruction Weights
  - Represent each point as a linear combination of its neighbors
- Find Low-Dimensional Embedding
  - Map points to lower dimensions (2D/1D) while preserving local structure

LLE is sensitive to no. of neighbors & may not preserve global shape.

Date: __/__/20__ Day: _____

Association Rule

- Apiori algorithm
- Implementing Apriori algorithm
- FP- Growth
- ECLAT

What is Apriori Algorithm

Apriori is a data mining algorithm used to find frequent itemsets and association rules in large datasets
Apriori shows which items often appear together
Example: If people buy bread, they also buy butter → store can offer combo deals

How it Works (Steps)

- Find Frequent Itemsets
- Count how many times each item appears called 1- itemsets
- Use minimum support (frequency threshold) to keep only common items
  - If an items frequency ≥ support → keep it
  - Else → remove it

- Create Large item Groups
  - Combine frequent 1- itemsets → make 2-item sets (pair)
  - Check their support
  - Keep only pairs that meet the threshold
  - Continue to 3- itemsets, 4-itemsets ... until no larger frequent sets are found

- Remove Infrequent Itemsets
  - Apriori rule:

Date: __/__/20 Day: _____

- If a group is not frequent, any larger group containing it will also be not frequent
- So, ~~kee~~ skip those to save time

- Generate Association Rules
  - From frequent itemsets, create rules like Bread → Butter
  - Evaluate rules using three metrics

Key Metrics

- Support :
  - How often an item/group appears in all transaction
  - Example : Beard in 20% of all transactions

- Confidence :
  - How often Y is bought when X is bought
  - Example: Bread → Butter = 75%

- lift :
  - How much more likely X & Y occur together vs Seportely. Example: lift > 1 → strong relation

Example

Given:

- 5 transactions in a grocery shop
- Min Support: 50% (≥ 3 out of 5)
- Min Confidence: 70%

Step 1: Frequent 1- Itemsets

All items appear ≥ 3 times → all are frequent

Step 2: Candidate 2- Itemsets

Date: __/__/20__ Day: _____

Pairs formed:

- (Bread, Butter), (Bread, Milk), (Butter, Milk)

Only (Bread, Milk) meets 50% support → frequent 2-Itemset

Step 3: Candidate 3-Itemsets

- Only {Bread, Butter, Milk}
- Support < 50% → Not frequent

Step 4: Association Rules

Rule	Confidence	Result
Bread → Butter	50%	Fail
Butter → Bread	100%	Pass
Bread → Milk	75%	Pass

Best Rules

- Butter → Bread
- Bread → Milk

Application

- E-commerce
- Food delivery
- Streaming
- Finance
- Travel
- Health

Frequent Pattern Growth Algorithm

FP-Growth (Frequent Pattern Growth) is a data mining algorithm used to find frequent itemsets in large datasets

Date: __/__/20__ Day: _____

It is faster than Apriori, because:

- It avoids generating candidates sets, and
- It scans the database only twice using a tree structure called FP-Tree (Frequent Pattern Tree)

Why FP-Growth?

- Apriori needs many databases scans & candidate generation (slow)
- FP-Growth compresses data & finds patterns directly from the tree, saving time & memory

How FP-Growth Works

- Data Compression
  - Builds an FP-Tree that stores all transactions in a compact form
  - Similar items are grouped along shared paths

- Mining the tree
  - Finds frequent patterns by exploring the conditional trees of each item
  - Uses minimum support to decide which items to keep

- Generate Patterns
  - Extracts frequent itemsets & builds association rules (like Apriori)

Advantages of FP-Growth

- No need for multiple database scans
- No candidate generation (unlike Apriori)
- Uses tree-based compression → faster & memory-efficient
- Works great for large datasets.

Date: __/__/20 Day: _____

ECLAT Algorithm

ECLAT (Equivalence Class Clustering & bottom-up Lattice Traversal) is a data mining algorithm used to find frequent itemsets (items often bought together). These itemsets are then used to build association rule (like "If Bread → Butter")

It's an improved & faster version of Apriori using vertical data format & depth-first Search (DFS)

Applications

- Market Basket Analysis
- Recommendation Systems
- Medical Diagnosis
- Web Usage Mining
- Fraud Detection

Advantages

- Faster than Apriori
- More memory-efficient (vertical format)
- less Scanning of dataset
- Works well on dense datasets

Disadvantages

- Uses more memory for large TID sets
- Not good for sparse data
- Expensive for very large transactions

Date: __/__/20__ Day: _____

Module 4 Reinforcement Learning

Model-Based Methods

- Markov decision processes (MDPs)
- Bellman equation
- Value iteration algorithm
- Monte Carlo Tree Search

Markov Decision Process

A Markov Decision Process is a mathematical way to describe how an agent (like a robot or game character) makes decisions in an uncertain environment to reach a goal.

It helps answer

- What action should the agent take?
- What happens after the action?
- Was the result good or bad?

Use in AI:

MDPs model situations where:

- Actions happen step-by-step
- Outcomes are uncertain (not always the same)
- The agent learns to maximize rewards over time

Components of MDPs

- States (S):
  - All possible situations the agent can be in
  - Example: Position on grid (e.g cell(1, 1))

Date: __/__/20__ Day: _____

- Actions (A):
  - Moves the agent can take
  - Example: UP, DOWN, LEFT, RIGHT

- Tranition Model (T):
  - Probability of reaching a new state after an action
  - Example: 80% go intended way, 10% slip left, 10% split right

- Reward (R):
  - Score received after each action - good or bad
  - Example: +1(goal), -1(fire), -0.1(per step)

- Policy (π)
  - Agent's plan - what action to take in each state
  - Example: "If in cell(1, 2), move RIGHT"

Application of MDP
- Robotics
- Games
- Healthcare
- Navigation
- Inventory Management

Bellman Equation

What is Bellman Equation?

The Bellman Equation tells how good a state is (its value) in reinforcement learning

It says:

The value of a state = current reward + expected value of the next state

It helps the agent make decisions by considering

Date: __/__/20__ Day: _____

immediate & future rewards

It follows the principle of Optimality — the best value of a state depends on:

- reward from the current action, and
- value of the next state

Bellman Equation for State Value Function

The state value Function $V(s)$ is the expected total reward from state s following a policy π

$$V^{\pi}(s) = E[R(s, a) + \gamma V^{\pi}(s')]$$

Expanding using transition probabilities

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s,a)[R(s,a) + \gamma V^{\pi}(s')]$$

where:

- $V^{\pi}(s)$: Value of state s under policy π
- $P(s'|s,a)$: Probability of moving from $s \rightarrow s'$ after action a
- $R(s, a)$: Reward for taking action a in state s
- γ : Discount factor (importance of future rewards)
- $\pi(a|s)$: Probability of taking action a in state s under policy π

Bellman Equation for Action Value Function (Q-Function)

The Q-Function $Q(s, a)$ = expected return for taking action a in state s and then following policy π

$$Q^{\pi}(s, a) = E[R(s, a) + \gamma V^{\pi}(s')]$$

Expanded form:

$$Q^{\pi}(s,a) = \sum_{s' \in S} P(s'|s,a)[R(s,a) + \gamma \sum_{a'} \pi(a'|s') Q^{\pi}(s',a')]$$

it estimates future rewards for both current & next actions.

Date: __/__/20 Day: _____

Bellman Optimality Equations

For optimal policy (π^*), we use max instead of expectation

- Optimal State Value Function

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a) + \gamma V^*(s')\right]$$

- Optimal Action Value Function (Q-learning)

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)\left[R(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

These are the core formulas for:

- Dynamic Programming
- Temporal Difference (TD) Learning
- Q-Learning

Solving Markov Decision Processes (MDPs)

Bellman equations are used in DP-based methods

- Value Iteration:

  Iteratively update values using Bellman Optimality Equation until Convergence

- Policy Iteration:

  Alternate Between

  - Policy Evaluation (compute V using Bellman Expectation)
  - Policy Improvement (update policy based on new V)

- Q-Learning:

  Learn optimal Q-values using Bellman Optimality

Key Ideas: Bellman Equation helps agents:

- Think ahead
- Balance short-term & long-term rewards
- Choose the best action intelligently

Date: __/__/20__ Day: _____

Value Iteration Algorithm

What is Value Iteration?

Value Iteration is a core algorithm in Reinforcement Learning (RL) used to find the optimal policy in a Markov Decision Process (MDP)
It updates the value of each state repeatedly using the bellman equation until it stabilizes (converges). This helps the agents choose the best strategy to maximize its total future reward

Steps of Value Iteration

- Initialization

Start with value $V(s) = 0$ for all state

- Value Update (Bellman Equation)

For each iteration

$$V_{k+1}(s) = \max_{a \in A} \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V_k(s')\right]$$

→ choose the action a that gives the highest expected value

- Convergence Check

Repeat until the value function changes very little

$$|V_{k+1}(s) - V_k(s)| < \epsilon$$

- Optimal Policy Extraction

After convergence, choose the best action for each state

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

Applications

Robotics, Games, Finance, Operations Research etc

Date: __/__/20 Day: _____

Monte Carlo Tree Search

What is MCTS?

MCTS is an algorithm used for decision problems (like Go or Chess) where exploring all moves is impossible.

It builts a tree step by step using random simulations (rollouts) to make better decisions

It balances

- Exploration: Trying new moves
- Exploitation: Using moves that already seem good

Example

A chess player can either follow a known good move (exploitation) or try a new one (exploration) MCTS automates this idea mathematically

Four Phases of MCTS (Repeat Until time/runs end)

- Selection
  - From the root, pick the best child using UCT (upper Confidence Bound for Trees) → balances explore / exploit

- Expansion
  - Add a new child node (new possible move)

- Simulation (Rollout):
  - From the new node, play random moves till the game ends → quick approximate result

Date: __/__/20__ Day: _____

- Backpropagation:
  - Send the result back up the tree → update each node's visits and win counts

Math Behind It – UCB1 Formula

$$UCB1(i) = \bar{X}_i + c\sqrt{\frac{\ln(N)}{n_i}}$$

Where

- $\bar{X}_i$: Average reward of node i
- c: Exploration constant ($\approx \sqrt{2}$)
- N: Total visits to parent node
- n_i: Visits to child node i

Meaning

- First term → Exploitation (choose high average reward)
- Second term → Exploration (try less-visited nodes)
- Exploration decreases as confidence grows

Core MCTS Methods

- expand() → Add one new child node
- best_child() → Pick best move using UCB1
- rollout() → Randomly play till end
- backpropagate() → Update scores back up

Real-World Example

Game AI, Planning, Neural Architecture Search

Limitations

- Needs many simulations
- High variance
- Tactical blindness
- Parameter tuning

Date: __/__/20__ Day: _____

Model - Free Methods

- Q-Learning
- SARSA
- Monte Carlo Methods
- Reinforce Algorithm
- Actor- Critic Algorithm
- Asynchronous Advantage Actor- Critic (A3C)

Q - Learning

Q-Learning is a model-free RL algorithm that helps an agent learn to take the best actions by interacting with the environment- through trial and error & feedback (rewards)

It doesn't need to know how the environment works

Main Idea

The agent builds a Q-table that stores Q-values which show how good an action is in a given state.

It learns these values by updating them over time as it tries actions & receives rewards

Key Components

- Q-Values (Action- Value)

Each $Q(s, a)$ tells how good it is to take action a in state s based on expected future rewards

- Rewards & Episodes

Date: __/__/20__ Day: _____

Each time the agent acts, it:

- Gets a reward (R)
- Moves to a new state (S')
- Continues until reaching a terminal state

- TD update Formula

Q-values are updated using

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{a} Q(S', a) - Q(S, A)]$$

Where:

- S, A → current state & action
- S' → next state
- R → reward
- α (alpha) → learning rate
- γ (gamma) → discount factor

- ε- Greedy Policy (Exploration Vs Exploitation)

Helps choose action:

- Exploit ($1-\varepsilon$): pick best known action (highest Q-value)
- Explore (ε): pick a random action to try new posibilities

How Q-Learning Works (Steps)

- Start at a State (s).
  - The agent observes the current environment

- Select an Action (A).
  - Choose using ε- greedy policy (explore or exploit)

- Execute the action:
  - The environment gives
    - Next State (S')
    - Reward (R)

Date: __/__/20 Day: _____

- Update the Q-table
  - Use the TD update rule to improve the Q-Value for (S, A)

- Repeat the Process
  - Continue for many episodes until the agent learns the optimal policy

Methods for Q-Value Calculation

- Temporal Difference (TD)

Learns from the difference b/w predicted and actual rewards

- Bellman Equation

$$Q(s, a) = R(s, a) + \gamma \max_{a} Q(s', a)$$

It updates Q-Values based on immediate rewards & best future reward.

What is Q-table?

A table showing Q-values for all state-action pairs

- Rows → states
- Columns → Actions
- Each cell → Expected reward (Q-Value)

Over time, it learns which actions give the best rewards

Advantage

- Learns from trail & error
- No model needed (model-free)

Date: __/__/20__ Day: _____

- Improves automatically over time.
- Works well for discrete problems

Disadvantages

- Slow for large environments
- Needs many episodes to learn
- High memory for large state/action spaces
- Not good for continuous actions

SARSA

State - Action - Reward - State - Action

SARSA is an on-policy reinforcement learning algorithm.

It helps an agent learn from its own actions by interacting with the environment, taking actions, getting rewards, & updating its strategy to maximize long-term rewards

Key Components

Component	Description
S (State)	Current position/situation of agent
A (Action)	Move or decision taken in state
R (Reward)	Feedback after taking action (+, -)
S' (Next State)	New state after action
A' (Next Action)	Next action chosen in next state

SARSA update Q-Values based on both current & next-action pairs

SARSA Update Rule

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$$

Date: __/__/20 Day: _____

Where:

- $Q(S, A) \rightarrow$ current Q-value
- α (alpha) $\rightarrow$ learning rate (0-1)
- γ (gamma) $\rightarrow$ discount factor (0, 1) for future rewards
- $R \rightarrow$ immediate reward
- $Q(S', A') \rightarrow$ next state-action Q-value

Intuition

- Immediate Reward: what agent got now
- Future Reward: expected reward from next action
- Correction: adjusts Q-value based on actual experience

SARSA Algorithm Steps

- Initialize Q-table with random or zero values
- Start at initial state. (S_0)
- Choose action (A_0) using ε-greedy policy
- Repeat for each step:
  - Take action A_0 observe reward (R_{t+1}) & next state (S_{t+1})
  - Choose next action (A_{t+1}) using same policy
  - Update Q-value

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

  - Set $S_t \leftarrow S_{t+1}$, $A_t \leftarrow A_{t+1}$
- End episode when goal or terminal state is reached

Example: Grid World

- Start (0, 0)
- Goal (4, 4)
- Obstacles (2,2), (3,2)
- Action: UP, DOWN, LEFT, RIGHT
- Reward: +10 $\rightarrow$ goal, -10 $\rightarrow$ obstacle, -1 $\rightarrow$ normal move

Date: __/__/20__ Day: _____

Exploration Strategy (ε-Greedy)

- Explore (ε): random actions to discover new states
- Exploit ($1-\varepsilon$): choose best-known action
- ε Decay: gradually reduce exploration as learning improves

Advantages

- On-policy: Learns from its real behaveour
- Stable learning: safer for uncertain environments
- More realistics than Q-learning

Limitations

- Slower convergence than Q-learning
- Sensitive to exploration settings

Monte Carlo Integration

Monte Carlo Integration is a statistical method that uses random sampling to estimate definite integrals.

Useful when the integral is complex or high-dimensional, where normal methods fail.

We estimate

$$I = \int_a^b f(x)\,dx$$

Instead of evaluating at every point, we take N random samples x_i from $[a, b]$

$$I \approx (b-a) \times \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

More Samples → more accurate (Law of large Nos)

Date: __/__/20 Day: _____

Reinforce Algorithm

What is REINFORCE?

REINFORCE is a policy gradient method in reinforcement learning that learns by trying actions and adjusting probabilities based on total rewards. It directly improves the policy (action selection) instead of estimating value functions.

- Good for tasks with many or continuous actions
- Learns by trial & reward

How REINFORCE Works

- Collect Episodes

The agent interacts with the environment → gets a list of (state, action, reward)

- Calculate Returns (G_t)

Total future reward from time t:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} R_k$$

where γ = discount factor

- Policy Gradient Update

Update the policy using

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \log \pi_\theta(a_t | s_t) G_t$$

- α = learning rate
- $\pi_\theta(a_t | s_t)$ = probability of action a_t at state s_t
- G_t = total reward

- Repeat

Keep running episodes & updating policy for better decisions

Date: __/__/20__ Day: _____

Advantages

- Simple & easy
- Direct Policy Update
- Good for Episodic Tasks

Limitations / Challenges

- High variance
- Sample Inefficient
- Slow Convergence

Variants

- Baseline

Reduce variance by subtracting baseline b_t:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \log \pi_\theta(a_t | s_t)(G_t - b_t)$$

b_t = expected reward from that state

- Actor - Critic
  - Actor : Chooses action
  - Critic : Evaluates action → gives feeback
    → More stable & faster learning

Applications

Game AI

Self-driving Cars

Actor-Critic Algorithm

Actor-Critic is a Reinforcement Learning (RL) method that combines two parts

- Actor → decides which action to take
- Critic → Judges how good that action was

Together they balance exploration (trying new actions)

Date: __/__/20__ Day: _____

and exploration (using what is learned) for better performance

Key Terms

- Policy (Actor)
  - Denoted by: $\pi(a|s) \rightarrow$ probability of taking action a in state s
  - Goal: maximize expected reward
  - Represented by actor network with parameters θ

- Value Function (Critic)
  - Denoted by: $V(s) \rightarrow$ expected cumulative reward from state s
  - Represented by critic network with parameters w

How It Works

The Actor-Critic objective combines

- Policy Gradient (Actor): improves policy
- Value Function (critic): evaluates value estimates

1- Actor Objective (Policy Gradient)

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{T=0}^{N} \nabla_\theta \log \pi_\theta(a_i|s_i) \cdot A(s_i, a_i)$$

Where

- $J(\theta)$: expected return
- $A(s,a)$: advantage function
- N: Number of samples

2- Critic Objective (Value Function)

$$\nabla_w J(w) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_w \left(V_w(s_i) - Q_w(s_i, a_i)\right)^2$$

where: $V(s) \rightarrow$ estimate value, $Q(s,a) \rightarrow$ estimate action-value

$w =$ critic parameter

Date: __/__/20__ Day: _____

Update Rules

- Actor Update

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t)$$

→ α: actor learning rate

- Critic Update

$$w_{t+1} = w_t - \beta \nabla_w J(w_t)$$

→ β: critic learning rate

Advantage Function

$$A(s, a) \approx Q(s, a) - V(s)$$

Tells how much better (or worse) an action a is compared to average actions at state s.

Advantages

- Efficient learning
- Faster Convergence
- Handles All Action Spaces
- OFF-policy learning

Varients of Actor-Critic

- A2C (Advantage Actor-Critic)
  - Uses advantage function
  - Reduces variance in updates
  - Formula

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

- A3C (Asynchronous Advantage Actor-Critic)
  - Runs multiple agents in parallel
  - Updates happen asynchronously
  - Faster + more stable learning

Date: __/__/20 Day: _____

Asynchronous Advantage Actor-Critic (A3C) Algorithm

~~A3C~~ A3C is a deep reinforcement learning algorithm that introduced parallel training using multiple agents learning at the same time

Unlike DQN (Deep Q-Network) which uses one agent sequentially, A3C uses many agents (workers), each with its own environment

These workers update a shared global model, making learning faster, more stable, diverse

A3C Architecture

- Global Network
  - Stores shared parameters for policy ($\pi(s)$) & value ($v(s)$) networks
- Workers (agents)
  - Each interacts with its own environment
  - Has a local copy of the model
  - Collects experiences & updates the global model
  - Syncs updated parameters back to local copies

Core Elements of A3C

- Asynchronous Training
  - Multiple agents work in parallel, collecting data independently
  - Each sends updates to the global network without waiting for others
- Benefits
  - Faster Training
  - Diverse experiences → avoids overfitting
  - Reduces sample correlation → more stable learning

Date: __/__/20__ Day: _____

- Actor - Critic Framework
  - Actor: learns policy $\pi(a|s)$ → decides which action to take
  - Critic: Learns value $V(s)$ → judges how good the state is
  - Actor selects actions, Critic evaluates them → better policy improvement

- Advantage Function

$$A(s,a) = Q(s,a) - V(s)$$

Tells how much better an action a is compared to the average

Uses:

- Gives Clearer feedback for learning
- Reduces variance in policy gradient updates

Mathematical ~~Intuition~~ Intuition

- Positive advantage ($A > 0$): Action was better than expected -reinforce it
- Negative advantage ($A < 0$): Action was worse → reduce its chance

A3C use n-step returns to balance bias & variance

- Shorter n: more bias, less variance → faster but less stable
- Longer n: Less bias, more variance → slower but smoother

Goals:

- Actor: increase actions with higher advantage
- Critic: reduce prediction error of value function

Training Pipeline

- Experience Collection:
  - Each worker collects (state, action, reward) sequences

Date: __/__/20 Day: _____

- Advantage Estimation:
  - Compute n-step returns & advantage for each action
- Gradient Calculation:
  - Each worker computes gradients for actor & critic
- Asynchronous Update:
  - Workers independently update the global network

This removes synchronization delays & keeps training continuous

Performace & Scalability
- Faster training
- Better exploration
- Less correlation
- Stable learning

Applications
- Game playing
- Robotics
- Financial Trading

Limitations
- Stable Gradients
- Exploration Redundancy
- Hardware Dependence

Date: __/__/20__ Day: _____

Module 5: Semi Supervised Learning

- Semi Supervised Classification
- Self-Training in Semi-Supervised Learning
- Few-shot learning in Machine learning

Semi Supervised Classification

Semi-Supervised learning is a mix of supervised & unsupervised learning

It uses both label & unlabeled data - mostly unlabeled - to train the model

Working Steps

- Train on small labeled data using supervised learning until it performs well
- Use Unlabled data and give them pseudo (fake) labels - accuracy is lower here
- Combine the labeled & pseudo-labeled data
- Train again on this combined data to reduce errors & increase accuracy

Advantages

- Easy to understand
- Uses fewer labeled data
- Quite reliable

Disadvantages

- Results may change with each iteration (unstable)
- Does'nt cover data at the network level
- Accuracy is not very high

Date: __/__/20__ Day: _____

Applications

- Speech Recognition
- Web Content Classification
- Text Document Classification

Self-Training in Semi-Supervised learning

Defination

Self-training is a semi-supervised learning method were:

- A model is first trained on small labeled data
- Then it predicts labels for unlabeled data
- The most confident predictions are added back to training data
- The process repeats until no improvement or all data is labeled

It is useful when labeled data is limited or expensive using unlabeled data to boost accuracy

Steps to Self-Training

- Train on labeled Data:
  - Built initial model using small labeled datasets
- Generate Peseudo labels:
  - Model predicts labels for unlabeled data, keep only high-confidence ones
- Augment Data
  - Add pseudo-labeled samples to original labeled data
- Iterative Reinforcement
  - Retrain model & repeat until stable performance or fixed iterations

Date: __/__/20__ Day: _____

Importance

- Simple and effective
- Uses unlabeled data
- Works for any domain
- Reduces manual labeling

Example (Binary Classification)

- 10% data labeled - train a model (logistic Regression)
- Predict labels for rest of data
- Add predictions with confidence > 95%
- Retrain → repeat process

Applications

- NLP : Text classification, QA
- Computer Vision: Image Recognition
- Health : diagnosis
- Speech processing : Voice detection

Benefits

- low cost
- Flexible
- Simple

- Challanges

- Error Amplification
- Confidence Threshold
- Imbalanced Data

Few-Shot Learning (FSL)

Few shot learning is a type of meta-learning where a model learns from very few examples

Date: __/__/20 Day: _____

Instead of thousands
It helps the model quickly recognize new tasks or objects using limited data

Example
To recognize a new car type, show only a few car images — the model then learns to detect simplar cars on its own

Key Terminologies
- K-ways : No. of classes (e.g 2-way → 2 → classes)
- K-shots: No. of samples per class (e.g 1-shot → one example per class)
- Query Set: Additional unseen samples (use for testing performance)
- Task: Support Set + Query Set
- Training: Model learn on multiple small tasks (support sets)
- Testing: Model predict unseen query example using what it learned

Working (Using Siamese Network)
- Two identical neural Networks (Siamese Networks) compare two inputs
- Each input (e.g image i & j) is converted to embedding : $f(i)$ & $f(j)$
- Compute Similarity

$$\| f(i) - f(j) \|^2$$

- Apply sigmoid function

$$Y = \sigma(w \Sigma |f(i) - f(j)|^2 + b)$$

→ $Y = 1$ → same class, $Y = 0$ → different class
- Uses a threshold t to decide similarity

Variations
- One-shot learning
  → 1 example per class Hard to generalize, higher error chance

Date: __/__/20__ Day: _____

- Zero-Shot learning

→ model recognizes unseen classes using semantic or text-based relations

- ~~Z~~ N-Shot learning

→ N examples per class (more than one but fewer than supervised)

→ More reliable & stable

Real World Applications

- Medical Imaging
- Robotics
- Image Recognition

Advantages

- Need less data for training
- Adapt fast to new unseen tasks
- Flexible & generalization model
- less training time due to small datasets
- Low Computation cost
- Work well for specialization or rare data

Disadvantages

- less Diversity
- Overfitting risk
- Hard for complex task
- Sensitive to Noise
- Rare Class issue

Date: __/__/20 Day: _____

Module 6: Forecasting Models

- ARIMA
- SARIMA
- Exponential Smoothing (Holt-Winters)

ARIMA

Auto-regressive Integrated Moving Average

Used for:

Time Series forcasting (predicting future values using past data)

ARIMA = AR (AutoRegression) + I (Integrated/Differencing) + MA (Moving Average)

Components of ARIMA

(A) Autoregression (AR → p)

Shows relation b/w current & past values

Formula

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \dots + \phi_p Y_{t-p} + \varepsilon_t$$

Where

- Y_t: current value
- $\phi_1, \phi_2 \dots \phi_p$: AR coefficients
- ε_t: error term
- p: number of lag observation

The current value depends on its own past values

Differencing (I → d)

Used to make data stationary (constant mean & variance)

Formula

$$Y_t' = Y_t - Y_{t-1}$$

where:

Date: __/__/20__ Day: _____

- Y_t': differenced series
- d: no. of times differencing applied

Repeated until the data becomes stationary

Moving Average (MA → q)

Shows relations b/w current value & past errors

Formula

$$Y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q}$$

where

- $\theta_1, \theta_2 \dots \theta_q$: MA coefficients
- q: no. of lagged errors terms

The model uses past forecast errors to predict the next value

Working Steps

Step 1 - Check Stationarity

- ARIMA needs stationary data (constant mean, variance)
- Use ADF Test to check
  - p-value > 0.05 → non-stationary (need differencing)
  - p-value ≤ 0.05 → stationary

Step 2 → Differencing

- Apply .diff() to remove trend & achieve stationity
- Recheck stationarity using ADF test again

Step 3 - Identify p & q (ACF/PACF plots)

- ACF plots: finds q (MA order)
- PACF plot: finds p (AR order)

Step 4 Model selection (Grid Search)

Try multiple (p, d, q) combinations & choose the one

Date: __/__/20__ Day: _____

with lowest AIC

AIC (Akaike Information Criterion)

Lower AIC = better model

Use python itertools.product() to test all (p, d, q)

Step 5 Model Fitting

Fit the ARIMA model with best parameters

```
model = ARIMA(ts, order = (p, d, q))
results = model.fit()
```

Step 6 - Forecasting

Predict future values

```
forecast = results.forecast(steps = 10)
```

→ Forcasts next 10 time steps

Step 7 → Evaluation

Compare actual vs forecasted values visually

Plot:

```
plt.plot(ts[-10:], label = 'Actual')
plt.plot(forecast, label = 'forecast')
plt.legend()
plt.show()
```

If results are poor

- Try different (p, d, q)
- Use SARIMA
- USA ARIMAX

SARIMA

(Seasonal Autoregressive Integrated Moving Average)

Date: __/__/20__ Day: _____

SARIMA is an advanced version of ARIMA that handles seasonal (repeating) patterns in time series data.

It's used when data shows trends that repeat monthy, yearly, weekly etc

Why SRRIMA?

- ARIMA → handles non-seasonal data
- SARIMA adds seasonal terms → perfect for seasonal cycles (sales, temperature, demand etc)
- Captures short-term & long-term patterns

Components of SARIMA

Component	Purpose
Seasonal	Captures repeating patterns (monthly/yearly cycles)
AR (p)	Relates current value to past values (autocorrelation)
I(d)	Makes data stationary (remove trends)
MA(q)	Relates current values to past forecast errors

SARIMA Notation

SARIMA (p, d, q) (P, D, Q, s)

Parameter	Meaning
p	AR order (non-seasonal lags)
d	Differencing order (non-seasonal)
q	MA order (non-seasonal)
P	Seasonal AR order
D	Seasonal differencing order
Q	Seasonal MA Order
s	Seasonal period

Example For monthly data with yearly seasonality → s = 12

Date: __/__/20__ Day: _____

Seasonal Differencing

To make data stationary, substract each value from the value in the previous season

$$Y_t' = Y_t - Y_{t-s}$$

Removes seasonal effect → helps the model focus on actual change

Mathematical Formula

$$(1-\Phi_1 B)(1-\phi_1 B^s)(1-B)(1-B^s)y_t = (1+\theta B)(1+\Theta_1 B^s)\varepsilon_t$$

where

y_t: observed value

B: backshift (lag) operator)

Φ, ϕ: AR coefficents

θ, Θ: MA coefficents

ε_t: error term

s: seasonal length

Exponential Smoothing (Time Series Forecasting)

Exponential Smoothing is a simple yet powerful forcasting method that predicts future trends using past data.

It gives more weight to recent data & less weight to older data (weights decrease exponentially)

Main Idea

- Future ≈ recent past
- Good for capturing overall patterns (level, trend, seasonality)
- Not ideal for long-term forecasting or sudden changes

Date: __/__/20 Day: _____

Types of Exponential Smoothing

- Simple (Single) Exponential Smoothing

Data with no trend & no seasonality

Parameter : α (alpha) = smoothing factor ($0 < \alpha < 1$)

- High α → gives more weight to recent values
- Low α → gives more weight to older values

Formula

$$S_t = \alpha X_t + (1-\alpha) S_{t-1}$$

Where

S_t = smoothed value

X_t = current observation

→ Balance past & present for stable forecasts

- Double Exponential Smoothing (Holt's Method)

Data with trend but no seasonality

Adds a trend component to the model

Parameters

- α → level smoothing
- β → trend smoothing

Formula

$$s_t = \alpha x_t + (1-\alpha)(s_{t-1} + b_{t-1})$$

$$b_t = \beta (s_t - s_{t-1}) + (1-\beta) b_{t-1}$$

Where

b_t = estimated trend at time t

→ Better than simple Smoothing when data shows an upward or downward trend

- Triple Exponential Smoothing (Holt Winters)

Data with trend + seasonality

Adds a seasonal component (γ) to level & trend

Date: __/__/20 Day: _____

Parameters

- α → level
- β → trend
- γ → seasonality $(0 < \gamma < 1)$

Can be:

- Additive (for constant seasonality)
- Multiplicative (for changing seasonality)

Formulas

$s_t = \alpha(x_t / c_{t-L}) + (1-\alpha)(s_{t-1} + b_{t-1})$

$b_t = \beta(s_t - s_{t-1}) + (1-\beta) b_{t-1}$

$c_t = \gamma(x_t / s_t) + (1-\gamma) c_{t-L}$

Most accurate but most complex

Advantages

- Simple & fast to compute
- Adapts to data by adjusting α, β, γ
- Better than moving average (more weight to recent data)
- Handles level, trend & seasonality

Limitations

- Struggle with sudden changes or outliers
- Accuracy depends on correct parameter selection
- Needs manual tuning or model optimization

Summary

- SES → constant level
- DES → level + trend
- TES → level + trend + seasonality

Date: __/__/20__ Day: _____

Module 7: Deployment of ML Models

- Machine learning Deployment
- Deploy ML Model Using Streamlit Library
- Deploy ML Web app on Herku
- Create UIs for prototyping ML Model with Gradio
  - Deploy ML Model Using Flask
  - Deploying ML Model as API using FastApi
- MLops
- Continuous Integration & Continuous Deployment (CI/CD)
- End-to-End MLops

Machine learning Deployment

Model Deployment means turning a trained ML model into a real-world tool that people or systems can use

Example → A fraud detection model watching live transaction to catch suspicious ones

→ Purpose: Make ML predictions available for real use.

Step-by-Step Deployment Process

- Step 1: Develop & Create Model
  - Train model in an offline environment using data
  - Many model are built - only best go for deployment
- Step 2: Optimize & Test Code
  - Clean, optimize & test your code
  - Make sure it runs correctly in production
- Prepare for Container Deployment
  - Containerize your model (using Docker)

Date: __/__/20__ Day: _____

- Containers make deployment
- Easy to update or move your model

- Step 4: Continuous Monitoring & Maintainance
  - After deployment, keep checking model performance
  - If accuracy drops or data changes, retain/update model
  - Monitor speed, errors & user impact continuously

Common Deployment Strategies

- Shadow Deployment
  - Run new model alongside old one (no user impact)
    → compare performance safely

- Canary Deployment
  - Release new model to small group of users first, then expand if it performs well

- A/B Testing
  - Show different model verisons to different user groups → pick the better one

Tools & Platforms for Deployment

- Kubernetes
- Kubeflow
- MLflow
- TensorFlow Serving

Best Practice For ML Deployment

- Automated Testing: Always test your model
- Version Control: Keep track of code, data & model
- Security: Protect model & data from attacks

Date: __/__/20 Day: _____

Deploying ML Model Using Streamlit

What is Streamlit?

- It is python library used to build interactive web apps for data Science or ML projects no web development needed.
- It handle backend & Frontend automatically
- Fast, Simple & perfect

Deploy Streamlit ML App on Heroku

What is Heroku?

Heroku is a cloud platform (PaaS) to build, run and manage apps online
Offers free plan - great for beginners
Deploy ML web App (like Streamlit) without coding or CLI.

Steps to Deploy Streamlit App on Heroku

- Create & login to Heroku
- Create Github Repository & Add Required Files
- Upload projects Files
- Create a new App on Heroku
- Name & Create App
- Connect Heroku App to Github
- Start Build Progress
- Wait for Deployment

Gradio - Create UIs For ML Models

What is Gradio?

Gradio is an open-source Python library to make simple, customizable UIs for ML models, APIs, or any function

Date: __/__/20 Day: _____

You can embed the UI in a jupter notebook or share it with anyone using a link

Deploy ML Models Using Flask

You can deploy using Flask
Installation & setup is easy
Use any model ie. Decision Tree Classifier
Dataprocessing
Create Flask App (app.py)

MLops (Machine Learning Operations)

MLops = combine ML + DevOps
Helps to build, deploy, monitor, & manage ML models easily & automatically

Goal

Make ML models scalable, reliable, & continously improving in real-world systems

Why MLops Important
Traditional ML has many Problems
MLops Fixed them

Problem	MLops Solution
lack of team collaboration	Brings Data Science + DevOps + Business
Manual Deployment	Automates model deployment
Poor Version Tracking	Tracks model/data/code
No Monitoring	Adds live montering
Resource Management	Uses automation & cloud tools

Date: __/__/20__ Day: _____

MLOps Workflow

- Data Collection & Preprocessing
- Model Deployment
- Model Training & Validation
- Model Deployment
- Monitoring & Maintenance

How to Implement MLOps?

- Evaluate Current Workflows
- Define Goal
- Form class- Functional Team
- Version Control & Pathing
- Automate Data pipelines
- Containerize Models
- CI/CD For ML
- Mointor Models
- Automate Retraining
- Security & Compliance

CI/CD in MLOps

What is CI/CD in MLOps?

- CI (Continuous Integration):

Regularly merging code changes, testing automatically to ensure everything works together

- CD (Continuous Deployment):

Automatically deploying tested changes to production

- In MLOps: CI/CD automates the ML lifecycle - from code, data, & models to deployment & monitoring

Date: __/__/20__ Day: _____

Includes:

- Code Integration
- Automated Testing
- Model Deployment
- Monitoring & Feedback

Benefits of CI/CD in MLops

- Faster Delivery:
  - Automates testing & Deployment → quicker model updates
- More Reliable:
  - Finds error early → better model quality
- Scalable:
  - Easily deploy & manage many models
- Consistent:
  - Same process across environments → fewer issues
- Better Teamwork:
  - Connects data sciencetists, engineers & Devops in pipeline

Key Components of CI/CD For ML Models

- Source Control Management
- Automated Testing
- Continuous Integration (CI) pipeline
- Continuous Deployment (CD) pipeline
- Monitoring & Feedback

Challenges in CI/CD ML

- Data Management
- Model Complexity
- Infrastructure
- Security & Compliance