Brief Mathematical Overview of Predictive ML Models

Classification

1. **Naive Bayes Classifier**
   It is a probabilistic machine learning model based on Bayes' Theorem, which is used for classification tasks. It assumes that the features (or attributes) used to predict the outcome (class) are independent of each other.

   Working: Suppose we have the following data: We have two features, x_1 = 'word1' and x_2 = 'word2', and two classes: y_1 = 'Spam' and y_2 = 'Not Spam'

   $P(x_1 \mid y_1)$ = Probability of 'word1' appearing in Spam emails
   $P(x_2 \mid y_1)$ = Probability of 'word2' appearing in Spam emails
   $P(x_1 \mid y_2)$ = Probability of 'word1' appearing in Not Spam emails
   $P(x_2 \mid y_2)$ = Probability of 'word2' appearing in Not Spam emails
   $P(y_1)$ = Proportion of Spam records
   $P(y_2)$ = Proportion of Not Spam records
   Now calculate the posterior probabilities for each class:

   $$P(y_1 \mid x_1, x_2) = P(y_1) * P(x_1 \mid y_1) * P(x_2 \mid y_1)$$
   $$P(y_2 \mid x_1, x_2) = P(y_2) * P(x_1 \mid y_2) * P(x_2 \mid y_2)$$

   Result: After calculating the posterior probabilities for all classes, the record is assigned to the class with the highest probability. This is the final prediction of the classifier.

2. **Logistic Regression**
   Logistic Regression estimates the probability that a given input $X=[x_1,x_2,\ldots,x_n]$ belongs to a particular class (e.g., class 1) by calculating a weighted sum of the input features and applying the sigmoid function to this sum.

   Working: Mathematically:

   $$P(Y = 1 \mid X) = \frac{1}{(1 + e^{-(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n)})}$$

   During the training phase, the logistic regression model learns the weights $w_1, w_2, \ldots, w_n$ by minimizing the cost function. This is done through optimization algorithms such as Gradient Descent, which adjusts the weights in the direction that minimizes the error between the predicted probabilities and the actual labels. The cost function used in logistic regression is the log loss or binary cross-entropy function. It measures how well the model's predicted probabilities align with the actual class labels. Once the model has been trained, the weights are set in such a way that the model can predict the probabilities for unseen data points (test data).

   Hyper parameters:

   i. Penalty (regularization): Regularization is used to prevent overfitting by adding a penalty term to the loss function. Here penalty can be l1 (Lasso regularization) which adds the absolute value of coefficients as penalty or l2 (Ridge regularization) which adds the squared magnitude of coefficients as penalty to the loss function.
   ii. Regularization Strength (C): Controls strength of Penalty i.e coefficient of penalty term
   iii. Solver (solver): Optimization algorithm used to minimize the cost function during training which can be liblinear: A solver based on the coordinate descent algorithm,

which is good for smaller datasets and when using l1 regularization, newton-cg: A Newton-based method, useful for larger datasets and l2 regularization, lbfgs: A quasi-Newton method, similar to newton-cg, but typically faster and more memory efficient. It works well with l2 regularization or saga: A variant of stochastic gradient descent that works with both l1 and l2 regularization and is effective with large datasets.

iv. Multi-Class Handling (multi_class): Logistic regression can be extended to multi-class classification (more than two classes). This hyper parameter determines how the model should handle multiple classes. It can be ovr (One-vs-Rest): For each class, a binary classifier is trained to distinguish that class from the others or multinomial: All classes are considered in a single optimization problem (more efficient for problems with many classes) and are passed through softmax function for output probability.

Results: Once we have the probability, we can classify the outcome into two classes:
If $P(Y = 1 \mid X)$ is greater than or equal to 0.5, classify as Y=1.
If $P(Y = 1 \mid X)$ is less than 0.5, classify as Y=0.

3. **Support Vector Classifier(SVC)**

Support Vector Classifier (SVC) is a type of Support Vector Machine (SVM), which is a powerful and flexible algorithm used for classification tasks. The key idea behind SVC is to find a hyperplane (in 2D, it's a line) that best separates the classes while maximizing the margin between them. SVC can be used for both linear and non-linear classification problems.

Terminologies:
Hyperplane: A decision boundary that separates data points of different classes.
Margin: The distance between the hyperplane and the nearest data points (support vectors) from either class. SVC aims to maximize this margin.
Support Vectors: Data points closest to the hyperplane; these points influence the position and orientation of the hyperplane.

Working:

The goal is to find a hyperplane that satisfies:

$$w.x_i + b = 0,$$

where: **w** is the normal vector to the hyperplane, $\mathbf{x_i}$ is a data point, **b** is the bias term. The goal of SVC is to **maximize the margin**, which minimizes the classification error and ensures a robust decision boundary. The margin is defined as $2/\|w\|$, so maximizing the margin translates to minimizing $\|w\|$. The optimization problem becomes:

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

subject to:

$$y_i(w.x_i + b) \geq 1, \qquad \forall i,$$

Where $y_i \in \{-1,+1\}$ are the class labels.

Hyper parameters:

i. C: Regularization Parameter that controls margin mazimization.
ii. kernel: *Linear Kernel (kernel='linear'):* Suitable for linearly separable data. Faster to compute. *RBF Kernel (kernel='rbf'):* Handles non-linear relationships by mapping data to a higher-dimensional space. *Polynomial Kernel (kernel='poly'):* Useful for data with polynomial relationships. Sigmoid Kernel (kernel='sigmoid'): Less commonly used; can behave like a neural network.

Results: A point x is classified as: **Class +1** if $w \cdot x + b > 0$ or **Class -1** if $w \cdot x + b < 0$

Comparison of Classification Models:

Model	**Applications**	**Advantages**	**Disadvantages**
Naive Bayes	Spam detection, Text classification, Sentiment analysis	Fast and simple, Handles large datasets well, Works with probabilistic outputs	Strong independence assumption, Struggles with correlated features, Not ideal for large feature sets
Logistic Regression	Credit scoring, Marketing campaign predictions, Medical diagnosis	Easy to interpret, Works well for linearly separable data, Probabilistic output for better insights	Sensitive to outliers, Can underperform with non-linear data without feature engineering, Assumes linearity
Support Vector Classifier (SVC)	Image classification, Handwriting recognition, Protein structure prediction	Effective in high-dimensional spaces, Can model non-linear boundaries (with kernels), Robust to overfitting with proper regularization	Computationally expensive, Sensitive to hyper parameter tuning, Struggles with large datasets

Regression

1. **Linear Regression**
It is one of the simplest and most widely used algorithms for regression tasks. The goal of linear regression is to model the relationship between a dependent variable (also called the target variable y) and one or more independent variables (features X) by fitting a linear equation to the observed data.

<u>Working:</u>

For multiple features $x_1, x_2, ..., x_d$, the model is generalized as:

$$y = w_1x_1 + w_2x_2 + \ldots + w_dx_d + b$$

or, in vector form:

$$y=Xw+b$$

where: y is the vector of predicted target values, X is the matrix of input feature values, w is the vector of weights (coefficients) for each feature and b is the bias term.

The goal of linear regression is to find the optimal values of the weights and bias b that minimize the difference between the predicted values y_i^ and the actual target values y_i

This is typically done by minimizing the **mean squared error (MSE)** between the actual and predicted values:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - y^{\wedge}{}_i\right)^2$$

Linear regression minimizes this error by adjusting the weights and bias through an optimization method, typically **Gradient Descent**.

Hyper parameters:

i. Learning Rate (η): Determines the size of the steps taken in the gradient descent optimization. If η is too small, the convergence may be slow. If it's too large, the algorithm may overshoot the optimal solution.

Results: Once the weights and bias have converged, the model is trained and can be used for prediction

2. **Ridge Regression (L2 Regularization)**
It is same as the Linear regression with a regularized term (sum of square of weights) added into the loss function. It doesn't drop features so it is used when we expect that all predictors contribute a bit to the prediction and we want to prevent multicollinearity.

$$Loss\ Function = MSE + \alpha\sum_{j=1}^{d} w_j^2$$

Here, alpha is hyper parameter that controls Regularization strength

3. **Lasso Regression (L1 Regularization)**
It is same as the Linear regression with a regularized term (sum of absolute weights) added into the loss function. Lasso can shrink some coefficients exactly to zero, effectively performing feature selection i.e. removing redundant features.

$$Loss\ Function = MSE + \alpha\sum_{j=1}^{d} |w_j|$$

Here, alpha is hyper parameter that controls Regularization strength

4. **Polynomial Regression**
Polynomial regression is an extension of linear regression that allows the model to capture non-linear relationships between the independent variable(s) and the dependent variable. It is particularly useful when the data exhibits a curvilinear trend.

Working: For multiple features $x_1, x_2, \ldots, x_d$, the model is generalized as:

$$y = w_0 + w_1x_1 + w_2x_1^2 + w_3\,x_1\,x_2 + w_4x_2^2 + \ldots + w_dx_d^2$$

This makes polynomial regression highly flexible, allowing it to model more complex relationships between variables. The objective of polynomial regression is the same as linear regression: to find the best-fit model by minimizing the error (typically the Mean Squared Error, MSE) between the predicted and actual target values.

Hyper parameters

i. Degree of the Polynomial (degree): The degree determines the complexity of the model. A higher degree can fit more complex relationships but might over fit the data.

5. **Spline Regression**
Spline regression is an extension of polynomial regression used to model complex, non-linear relationships in data. Unlike polynomial regression, which can suffer from overfitting when dealing with high-degree polynomials, spline regression uses piecewise polynomials (called splines) to model data in a more flexible and efficient manner. It divides the data into intervals and fits a polynomial to each interval, ensuring that the different pieces of the polynomial smoothly connect at certain points known as knots. Basis Functions are the functions that represent the spline segments and we can define them. Common types are linear, cubic splines, B-splines, and natural splines. All other working is same as Polynomial one.

6. **GAM Regression**
The key idea behind GAM is to model the relationship between the features and the target variable as a sum of smooth functions of the individual features. These functions can be chosen to be flexible enough to capture complex, non-linear patterns in the data. They are not restricted to a specific form like polynomials and can capture complex relationships in the data.

*Working: T*he general form of a Generalized Additive Model (GAM) is:

$$E(Y|X) = \beta + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p)$$

Where, Y is the target variable (response), $X_1, X_2, \ldots, X_p$ are the independent variables (predictors) $f_1, f_2, \ldots, f_p$ are smooth functions that represent the non-linear relationships between the predictors and the response and β is the intercept term.
Results: The final decision is sum of output of individual functions.

Comparison of Regression Models:

Model	**Applications**	**Advantages**	**Disadvantages**
Linear Regression	Predicting house prices, sales forecasting, cost estimation	Simple to implement, easy to interpret, works well for linear data	Sensitive to outliers, assumes linearity, struggles with multicollinearity
Ridge Regression	Handling multicollinearity, prediction in high-dimensional data	Reduces overfitting, handles multicollinearity, robust for high-dimensional data	Requires tuning of regularization parameter, less interpretable coefficients
Lasso Regression	Feature selection, sparse data modeling, predictive analytics	Performs feature selection, reduces overfitting, interpretable sparse models	Requires tuning, can shrink coefficients to zero, computationally expensive

Polynomial Regression	Modeling non-linear relationships in housing markets, economic data trends	Captures non-linear relationships, flexible for complex data patterns	Prone to overfitting with high-degree polynomials, requires careful degree selection
Spline Regression	Environmental modeling, time series data, advanced curve fitting	Smooth and flexible fits, handles non-linear relationships efficiently	Requires knot selection, can overfit with too many knots, computationally complex
Generalized Additive Model (GAM)	Healthcare outcomes prediction, complex relationship modeling in economics	Flexible and interpretable, captures non-linear trends without overfitting	Can be computationally expensive, sensitive to smoothness parameter selection

Multi-Purpose

1. **K - Nearest Neighbors (KNN)**
   The fundamental idea behind KNN is simple: given a new data point, the algorithm looks at the K nearest data points (neighbors) in the training set and makes predictions based on them.

   Working:

   There is no training phase in KNN, whether algorithm finds neighbors directly by using a specific distance metric such as Euclidean Distance, Manhattan Distance (sum of absolute differences) or Minkowski Distance (generalized form of Euclidean and Manhattan) which can be specified by hyper parameter. Also how many neighbors (K or n_neighbors) can also be specified and it strongly impacts the model's behavior. Small K makes the model sensitive to noise (overfitting), whereas a larger K helps smooth out decision boundaries. However, we can use hyper parameter tuning or model testing based on specific metric such as accuracy, precision, rmse etc. to find k which normally lies between 1 and 30.

   Results:

   For Classification the class label of the new data point is determined by a majority vote of its K nearest neighbors while for Regression the prediction is typically the average or weighted average of the K nearest neighbors' target values.

2. **Decision Tree**
   A decision tree splits the data into subsets based on the values of input features. At each node of the tree, a decision is made using a certain feature, which partitions the data into smaller groups. These decisions are made based on a criterion that maximizes the homogeneity of the resulting subsets. For Classification the output at the leaf node is the most frequent class in that subset while for Regression the output at the leaf node is the average of the target values in that subset.

   Components:

   Root Node: The topmost node in the tree. It represents the entire dataset and the first feature used for splitting. **Internal Nodes**: Represent the decisions or conditions based on a feature. **Branches**: Show the outcomes of a decision and lead to the next node. **Leaf**

Nodes: Represent the final prediction (either a class label for classification or a numerical value for regression).

Working

i. The Decision Tree splits the dataset into smaller subsets based on **decision rules**.
ii. At each internal node, the algorithm selects a feature to split the data based on a specific **criterion** (e.g., Gini Impurity, Entropy, or Variance Reduction: Lower Gini or Entropy indicates a purer node). The goal is to split the data in a way that the resulting subsets are as **pure** as possible.

$$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

$$Entropy = -\sum_{i=1}^{C} p_i \log_2(p_{i)}$$

Where p_i is the proportion of class i in the node.

$$Variance\ Reduction = Variance_{parent} - \left(\frac{n_{left}}{n} \times Variance_{left} + \frac{n_{right}}{n} \times Variance_{right}\right)$$

Where :
n_{left}: Number of samples in the left subset (after the split).
n_{right}: Number of samples in the right subset (after the split).
n: Total number of samples in the parent node (before the split).
Variance (Parent Node): Variance of the target variable in the parent node (before the split).
Variance (Left Node): Variance of the target variable in the left child node (after the split).
Variance (Right Node): Variance of the target variable in the right child node (after the split).

iii. This process continues until a stopping condition is met (e.g., maximum depth, minimum number of samples per node, etc.).
iv. The leaf nodes represent the final output (class label or value).

Hyper parameters:

i. *Max Depth:* The maximum depth of the tree. A larger depth allows more splits, which can lead to overfitting.
ii. Min Samples Split: The minimum number of samples required to split a node. Higher values reduce overfitting.
iii. Min Samples Leaf: The minimum number of samples required in a leaf node. Prevents very small, overfitted leaf nodes.
iv. Max Features: The maximum number of features considered when looking for the best split.
v. Criterion: The metric used to evaluate the quality of a split (e.g., Gini, Entropy, or Variance Reduction).

Results:

i. For **Classification**: The leaf node assigns the majority class.
ii. For **Regression**: The leaf node predicts the average target value of the data points in that node.

3. Ensemble Modeling

Apply multiple models to data and select the best one

i. Voting Classifier/Regressor

Takes a list of models as inputs and fits the best one

ii. Bagging

Bagging (short for Bootstrap Aggregating) is an ensemble learning technique used to improve the stability and accuracy of machine learning models, especially for high-variance algorithms like decision trees. Bagging reduces variance by creating multiple subsets of data, training a model on each subset, and combining the results of all models. The core idea of bagging is to combine predictions from multiple models to reduce overfitting and improve generalization by averaging the results (for regression) or voting (for classification).

Hyper parameter

i. base_estimator: The base model or algorithm used in bagging (e.g., DecisionTreeClassifier, SVC, etc.). Default is None, which means decision trees are used.
ii. n_estimators: The number of base models (or estimators) in the ensemble. Higher values increase stability but also computation time.
iii. max_samples: The fraction or number of samples to draw from the training dataset for each base estimator (bootstrapping).
iv. max_features: The fraction or number of features to draw for each base estimator when training.
v. Bootstrap: If True, sampling is done with replacement (default). If False, sampling is done without replacement.
vi. bootstrap_features: If True, features are sampled with replacement. If False, all features are used for each bootstrap sample.
vii. oob_score: If True, out-of-bag samples (missed in modeling) are used to evaluate the model's performance.

a. ***Random Forest****:*

A bagging technique with Decision Tree as base model where each tree is trained on a random subset of the training data (using bootstrapping) and each split in a tree considers a random subset of features (introducing additional randomness). Note that here along with hyper parameters of Bagging we can also use hyper parameters of Decision Trees to optimize modeling.

iii. Boosting

Boosting is an ensemble learning technique in machine learning that aims to convert weak learners (models that perform slightly better than random guessing) into strong learners by combining their predictions. Boosting focuses on reducing bias and improving predictive performance by sequentially building models that correct the errors of previous models.

a. ***Ada Boost***

AdaBoost, short for Adaptive Boosting, is one of the first and most popular boosting algorithms. It is designed to improve the performance of weak learners (classifiers that perform slightly better than random guessing) by combining them into a strong classifier. AdaBoost focuses on misclassified instances by assigning them higher weights, making subsequent models pay more attention to these difficult cases.

Hyper parameters:

i. base_estimator: The weak learner or base model used for boosting. Defaults to a decision tree classifier (DecisionTreeClassifier).
ii. n_estimators: Number of weak learners (iterations). Larger values improve accuracy but increase training time.
iii. learning_rate: Shrinks the contribution of each weak learner by multiplying its weight. Smaller values require weaker learners.

b. ***Gradient Boost***

Gradient Boosting is a powerful ensemble learning technique that builds a predictive model in a sequential fashion by combining the predictions of multiple weak learners (usually decision trees). Unlike AdaBoost, which adjusts weights for misclassified samples, Gradient Boosting focuses on minimizing the loss function by sequentially fitting new models to the residuals (errors) of the previous models.

c. ***Extreme Gradient Boost****:*

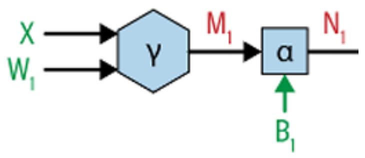
XGBoost is an advanced implementation of the Gradient Boosting algorithm, designed for both speed and performance. It is widely used in machine learning competitions like Kaggle and is highly regarded for its efficiency, accuracy, and scalability. XGBoost builds upon traditional Gradient Boosting by introducing additional features, optimizations, and regularization techniques.

Hyper parameters:

i. Booster: base model
ii. Lambda: L2 regularization strength.
iii. Alpha: L1 regularization strength.
iv. Objective: Loss function to optimize (e.g., "reg:squarederror" for regression, "binary:logistic" for classification).

4. Neural Network

In neural networks we extend our simple linear regression multiple ones. First note that a single Linear Regression involves following Steps:



Forward Pass:
In Neural Networks we first apply a Linear regression based on some weights W1 and then pass the output through a nonlinear function called sigmoid which enables our neural network to model the inherently nonlinear relationship between the features and the target. This whole process is called first layer of a Neural net. Then output of first layer is again feeded to another linear regression with Weights W2 and same process goes on. The second layer and so on. The final layers output then undergoes another linear regression, whose output is not subjected to Sigmoid function rather we use this output to compute Loss Function. This whole process is called Forward Pass:

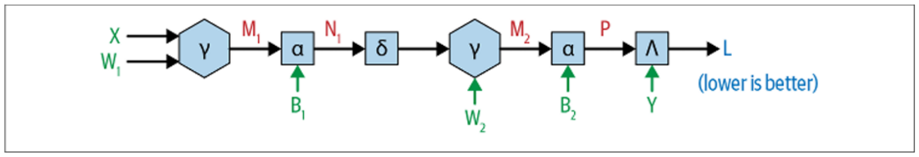


Figure 2-10. Steps 1–3 translated into a computational graph of the kind we saw in Chapter 1

This is double layer NN and here Sigma is Sigmoid function.

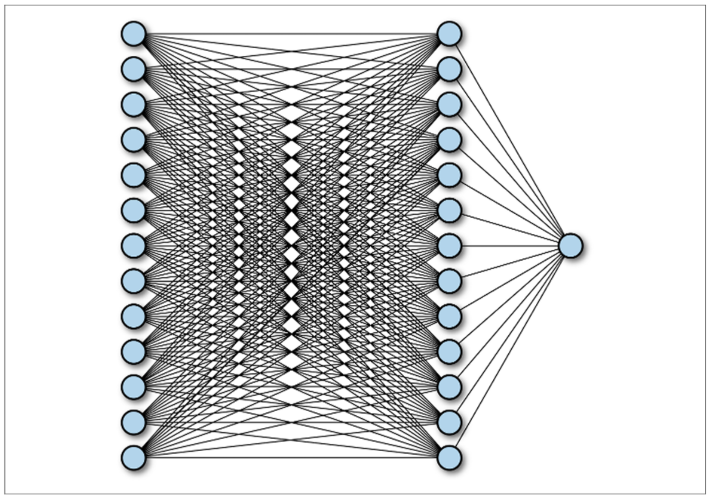


Figure 2-11. A more common (but in many ways less helpful) visual representation of a neural network

Backward Pass:
For optimization of L we use backward pass as we used in Simple LR. We have to calculate:

dL/dW2, dL/dB2 , dL/dW1 and dL/dB1 for Gradient Decent

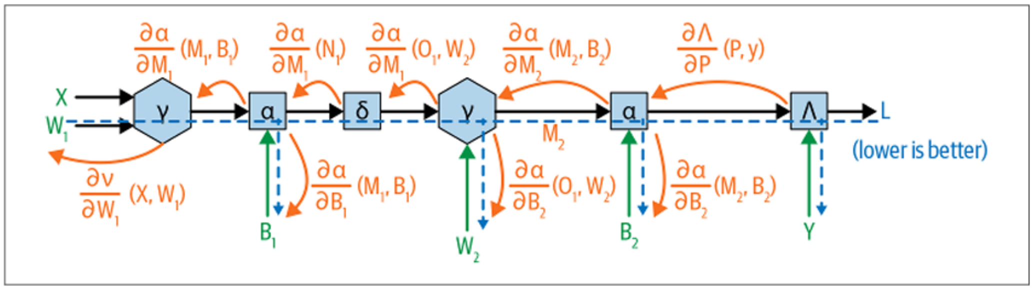


Figure 2-12. The partial derivatives associated with each operation in the neural network that will be multiplied together on the backward pass

Using this gradient, we update the weights and B in gradient descent as follows:

$$W = W - \alpha \cdot \frac{\partial L}{\partial W}$$
$$B = B - \alpha \cdot \frac{\partial L}{\partial B}$$

Here $\boldsymbol{\alpha}$ is a parameter known as learning rate that control how much we will change the values of W and B in each Iteration.

Comparison of Models

Model	**Applications**	**Advantages**	**Disadvantages**
K-Nearest Neighbors (KNN)	Pattern recognition, recommender systems, anomaly detection	Simple to implement, no training phase, effective for non-linear data	Computationally expensive, sensitive to irrelevant features, struggles with large datasets
Decision Tree	Fraud detection, healthcare diagnosis, customer segmentation	Interpretable, handles mixed data types, no data scaling required	Prone to overfitting without regularization, sensitive to small data changes
Bagging (e.g., Random Forest)	Fraud detection, credit scoring, ensemble-based predictions	Reduces overfitting, handles high-dimensional data, robust predictions	Requires more computation than single models, less interpretable
Boosting (e.g., AdaBoost, XGBoost)	Healthcare predictions, financial modeling, anomaly detection	Improves weak learners, handles imbalanced data well, high accuracy	Sensitive to noise, can overfit if not tuned properly, computationally intensive
Neural Networks	Image recognition, speech processing, natural language processing	Highly flexible, can model complex patterns, suitable for large datasets	Requires a large amount of data, computationally intensive, difficult to interpret