Machine Learning Cheatsheet

Machine Learning General

Definiton

We want to learn a target function f that maps input variables X to output variable y, with an error e:

$$y = f(X) + e$$

Linear, Non-linear

Different algorithms make different assumptions about the **shape** and **structure** of f. Any algorithm can be either:

- **Paramteric (or Linear)**: simplify the mapping to a known linear combination form and learn its coefficients.
- **Non-parametric (or Non-linear):** free to learn any functional form from the training data, while maintaining some ability to generalize.

Note: Linear algorithms are usually simpler, faster and requires less data, while Nonlinear can be are more flexible, more powerful and more performant.

Supervised, Unsupervised

• **Supervised** learning methods learn to predict outcomes y $(y^{(1)}, ..., y^{(m)})$ from data points X $(x^{(1)}, ..., x^{(m)})$ given that the **data is labeled**.
→ **Type of prediction**

	Regression	**Classification**
Outcome	Continuous	Class
Examples	Linear Regression	Logistic Regression, SVM, Naive Bayes

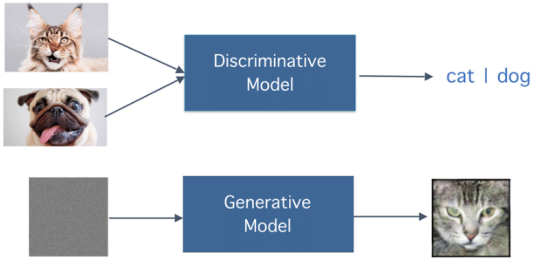
→ **Conditional Estimates**
Regression → **conditional expectation**: $E[y|X = x]$
Classification → **conditional probability**: $P(Y = y|X = x)$

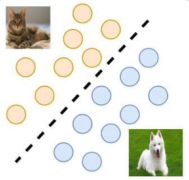
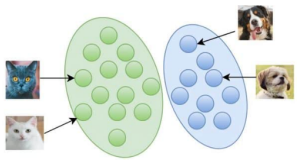
• **Unsupervised** learning methods learn to find the inherent structure or hidden patterns from **unlabeled data** X $(x^{(1)}, ..., x^{(m)})$.

Type of models

→ **Discriminative Model**: It focuses on predicting the data's outputs (for classification or regression) by training a model. It learns parameters by maximizing the **conditional probability** $P(Y|X)$.

→ **Generative Model**: It focuses on learning a probability distribution for the dataset, it can reference this probability distribution to generate new data instances. It learns parameters by maximizing the **joint probability** of $P(X, Y) = P(X \cap Y)$.



Discriminative model	**Generative model**
	
Learns the decision boundary between classes	Learns the input distribution
Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to find likelihood $P(y\|x)$ using **Baye's** rule
Specifically meant for classification tasks	They are used for generating new contents or data
Logistic Regression, Random Forests, SVM, Neural Networks, Decision Tree, kNN	Hidden Markov Models, Naive Bayes, Gaussian Mixture Models, Gaussian Discriminant Analysis, LDA, Bayesian Networks

Bias-Variance trade-off, Underfitting, Overfitting

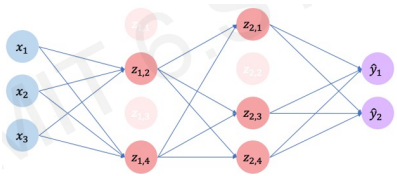
In supervised learning, the **prediction (expected) error** e is composed of the **bias**, the **variance** and the **irreducible** part.

$$\text{Error}(x) = \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2 + \mathbb{E}\left[\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right]^2 + \sigma_e^2$$

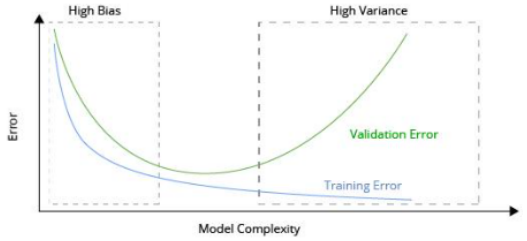
→ **Bias** refers to **erroneous assumptions** made by the **model** about the data to make the **target function** f easier to learn. **Mathematically**, how much do the expected predicted values differ from true values?
→ **Variance** is the **error**, refers to how much the predictions of the model (i.e estimate of the target function f) will vary when trained on **different training data (or possibly with different random seeds)**. It is also known as **Variance Error** or **Error due to Variance**. **Mathematically**, variance is the **expected squared deviation** of the model's prediction from its mean prediction, across different training sets

Note: As the **complexity of the model** (ex: flexibility of the decision boundary) **increases**, the variance will **increase** and bias will **decrease**

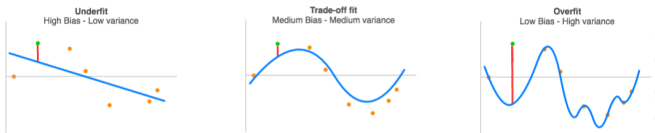
• The **goal of parameterization** is to achieve a **low bias** and **low variance** trade-off through methods such as:
→ **Cross-validation** can be used to tune models so as to optimize the trade-off between **bias** and **variance**
→ **Dimension reduction** and **feature selection**
→ Mixture models (probabilistic models) and **ensemble learning**.
→ **Regularization** - **Dropout** During training, randomly set some activations to 0. This forces network to not rely on any one node.

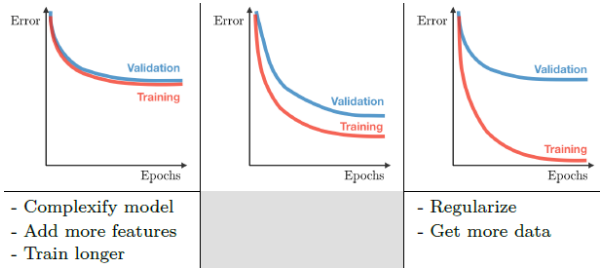


• **Underfitting** or **High bias** means that the model is not able to capture (or learn) the trend (or pattern) in the data.
• **Overfitting** or **High variance** means that the model learns too much from the available data but does not generalize well enough to predict on **new data**.



→ The **training loss** goes down over time, achieving low error values
→ The **validation loss** goes down until a turning point is found, and it starts going up again. That **point** represents the **beginning** of **overfitting**. Therefore, **The training process should be stopped when the validation error trend changes from descending to ascending**.





• Training loss vs. Validation loss:

Training Loss \ Validation Loss	Low	High
Low	?	Overfit
High		Underfit

→ **Epochs:** One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.
→ **Batch:** You can't pass the entire dataset into the neural net at once. So, you divide dataset into No. of Batches or sets or parts.
→ **Iterations** is the No. of Batches needed to complete One Epoch.
Question: If total number of samples in a dataset is 1000 and batch size is 10, how many iterations will be there in one epoch. **Ans**: **100**

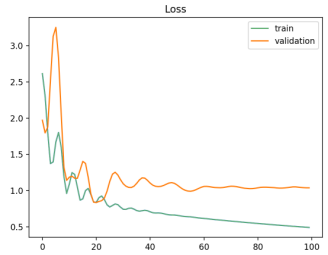
• **How would you identify if your model is overfitting?** By analyzing the **learning curves**, you should be able to spot whether the model is underfitting or overfitting. The y-axis is some metric of learning (ex:, classification accuracy) and the x-axis is experience (time or No. of iteration).

• **What is cross-validation? Why it's important?**
Cross-validation evaluates a model's performance.
→ The idea is to divide the dataset into k subsets or "folds", train the model on $k - 1$ of these folds, and test on the remaining fold to ensure that the model generalizes well to unseen data.
→ After evaluating on all k folds, performance metrics are averaged for a robust estimate of the model's effectiveness.

→ **K-Fold Cross-Validation**: The data is divided into k equally-sized folds.
→ **Stratified K-Fold**: Similar to K-Fold, but it maintains the proportion of classes in each fold, making it ideal for imbalanced datasets.
→ **Leave-One-Out Cross-Validation (LOOCV)**: A special case where k equals the number of data points, so each fold contains just one data point.

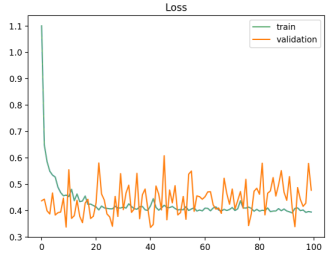
Unrepresentative Training Dataset

When the data available during training is not enough to capture the model, relative to the validation dataset.

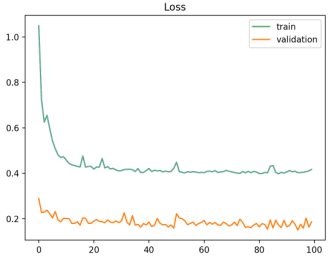


The train and validation curves are improving, but there's a big gap between them, which means they operate like datasets from different distributions.

Unrepresentative Validation Dataset



As we can see, the training curve looks ok, but the validation function moves noisily around the training curve. It could be the case that validation data is scarce and not very representative of the training data, so the model struggles to model these examples.



Here, the validation loss is much better than the training one, which reflects the validation dataset is easier to predict than the training dataset. An explanation could be the validation data is scarce but widely represented by the training dataset, so the model performs extremely well on these few examples.

Data Science General

Inference vs. Classification

→ **Inference**: Given two groups, what is the differences between these groups? t-tests, paired t-test etc.
→ **Classification**: Given a new animal, find whether new animal is cat or dog?

Prediction and Inference

→ **Prediction** uses a model to predict future observations.
- Model does not need to be valid
- Evaluation does need to be valid
- Quality & strength: accuracy of predicting unseen data

→ **Inference** uses the model's structure and parameters to learn or understand an underlying phenomenon.
- Validity depends on assumptions
- Quality & strength: R^2, p-value, assumption checks, coefficients

Preventing Data Leakage

- **Proper Data Splitting**: Split data into training, validation, and test sets **before** performing any data preprocessing. For time series, split data chronologically.
- **Transformation fit on Training Data Only**: Ensure that transformations (e.g., scaling, encoding) are fit only on the training data and then applied to both training and test data
- **Feature Selection**: Ensure that features used for training are available at the time of prediction. For time series data, create lag features that use only past information.
- **Data Augmentation**: Apply augmentation techniques only to the training set.

Handle missing or corrupted data in a dataset

- **Remove Missing Data**: If only a small number of **rows** have missing values, or If an entire **column** has a large percentage of missing values
- **Impute Missing Data**: Mean/Median/Mode Imputation, Imputation Using Algorithms like Forward/Backward Fill/ Interpolate for time series data
- **Use Algorithms That Handle Missing Data**: Decision trees, Random forests
- **Replace Corrupted Data**: Identify Outliers using statistical methods (like z-scores or IQR), Apply manual inspection and correction of corrupted values based on external sources or domain expertise.
- **Data Augmentation**: To create synthetic data based on existing patterns

Notes

→ **Ablation**: An ablation study is turning off components of a model (e.g. features or sub-models) one at a time, to see how much each contributes to the model's performance.

ETL - Extract Transform Load

An ETL workflow is crucial for consolidating and preparing data for analysis and reporting, ensuring accuracy, consistency, and availability for better decision-making.

1. **Extract Phase**: Retrieve raw data from sources like databases, files, or APIs using SQL queries, scripts, or connectors. Extract only the new or updated data since the last extraction to optimize performance and reduce load.
2. **Transform Phase**: Clean, integrate, and format data, applying business rules and calculations. Validate transformations and ensure data quality using tools like Python, SQL, or Apache Spark.
3. **Load Phase**: Insert transformed data into the target system, ensuring correct schema, bulk loading for efficiency, and optimizing with indexing and partitioning.

ETL Workflow Automation
Automation Tools:
→ Apache Airflow: For workflow scheduling and orchestration.
→ AWS Glue: Managed ETL service on AWS.
→ Apache NiFi: For data flow automation.

Model Evaluation

Classification Problems

Confusion Matrix

Type I error: The null hypothesis H_0 is **rejected** when it is **true**.
Type II error: The null hypothesis H_0 is **not rejected** when it is **false**.
→ **False negative** (**Type I error**) — incorrectly decide **no**
→ **False positive** (**Type II error**) — incorrectly decide **yes**

		Truth (**Actual**)	
		Null Hypothesis (**H0**)	Alternative Hypothesis (**H1**)
Decision (**Prediction**)	Do Not Reject **NULL**	OK	Type II Error
	Reject **NULL**	Type I Error	OK

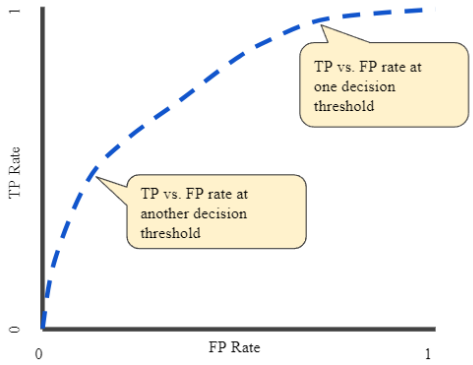
Ex: We assume the null hypothesis H_0 is true.
→ H_0: Person is not guilty
→ H_1: Person is guilty

		Truth (**Actual**)		
		Not Guilty (**H0**) 1	Guilty (**H1**) 0	
Decision (**Prediction**)	Not Guilty (**H0**) 1	OK (**TP**)	Type II Error (**FP**)	**Precision** TP / (TP+FP)
	Guilty (**H1**) 0	Type I Error (**FN**)	OK (**TN**)	
		Sensitivity TP / (TP+FN)	**Specificity** TN / (TN+FP)	**Accuracy** (TP+TN) / (TP+TN + FP +FN)

		Actual			
		C1	**C2**	...	CN
Predicted	C1	**TN**	**FN**	**TN**	
	C2	**FP**	**TP**	**FP**	
	...	**TN**	**FN**	**TN**	
	CN				

1. **Accuracy:** $\dfrac{TP+TN}{TP+TN+FP+FN}$
   → Ratio of **correct predictions** over **total predictions**.
   Estimate of $P[D=Y]$, probability of decision is equal to outcome.
2. **Recall** or **Sensitivity** or **True positive rate**: $\dfrac{TP}{TP+FN}$.
   Completness of model → Out of **total actual positive (1) values**, how often the classifier is correct.
   Probability: $P[D=1|Y=1]$
   Example: "**Fradulent transaction detector**" or "**Person Cancer**" → +ve (1) is "fraud": Optimize for sensitivity because false positive (**FP** normal transactions that are flagged as possible fraud) are more acceptable than false negative (**FN** fradulent transactions that are not detected)
3. **Precision**: $\dfrac{TP}{TP+FP}$
   Exactness of model. → Out of **total predicted positive (1) values**, how often classifier is correct.
   Probability: $P[Y=1|D=1]$, If our model says positive, how likely it is correct in that judgement.
   Example: "**Spam Filter**" +ve (1) class is spam → Optimize for precision or, specificity because false negatives (**FN** spam goes to the inbox) are more acceptable than false positive (**FP** non-spam is caught by the spam filter). **Example:** "**Hotel booking canceled**" +ve (1) class is isCanceled → Optimize for precision or, specificity because false negatives (**FN** isCanceled labeled as "not canceled" 0) are more acceptable than false positive (**FP** isnotCanceled labeled as "canceled" 1).
4. **F1-Score** $= \left[2 \times \dfrac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}\right]$
   False positive (FP) and False negative (FN) are equally important.
5. **False Positive Rate**: $\dfrac{FP}{TN+FP}$
   Fraction of **negatives** wrongly classified positive.
   Probability: $P[D=1|Y=0]$
6. **False Negative Rate**: $\dfrac{FN}{TP+FN}$ = 1 - Recall
   Fraction of **positives** wrongly classified negative.
   Probability: $P[D=0|Y=1]$
7. **Specificity**: $\dfrac{TN}{TN+FP}$ = 1-FPR
   Fraction of **negatives** rightly classified negative.
   Probability: $P[D=0|Y=0]$

• **ROC**-curve: The curve illustrates the trade-off between (TPR) true positive rate **(sensitivity or recall)** and the (FPR) false positive rate using classification thresholds α.
→ Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.



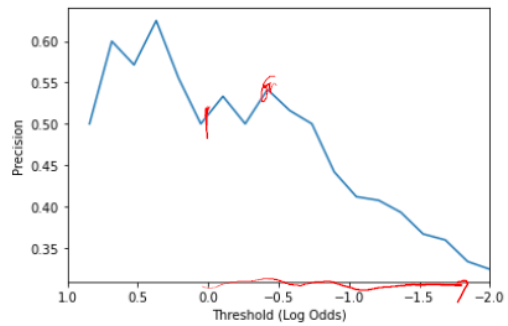
Note: We can think of the plot as the fraction of **correct predictions for the positive class** (y-axis) versus the fraction of **errors for the negative class** (x-axis).

How to choose threshold for the logistic regression? The choice of a threshold depends on the importance of TPR and FPR classification problem. If there is no external concern about low TPR or high FPR, one option is to weight them equally by choosing the threshold that maximizes TPR−FPR.

```
# Get predicted probabilities
y_prob = model.predict_proba(X_test)[:, 1]
# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
# Optimal threshold using Youden's J stat.
youden_index = tpr - fpr
optimal = thresholds[np.argmax(youden_index)]
```

• Area Under the ROC Curve **AUC**: To compute the points in an ROC curve, an efficient, sorting-based algorithm called AUC. AUC ranges in value from 0 to 1. Area Under the Curve measures how likely the model differentiates positives and negatives (perfect AUC = 1, basline = 0.5)

• **Precision-Recall** curve: Focuses on the correct prediction of the **minority** class, useful when data is **imbalanced**. Plot precision at different thresholds.



Regression Problems

1. **Mean Squared Error:** $\text{MSE} = \dfrac{1}{n}\sum_i (y_i - \hat{y}_i)^2$
2. **Root Mean Squared Error:** $\text{RMSE} = \sqrt{\dfrac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}$
3. **Mean Absolute Error:** $\text{MAE} = \dfrac{1}{n}\sum_i |y_i - \hat{y}_i|$
4. **Sum of Squared Error:** $\text{SSE} = \sum_i (y_i - \hat{y}_i)^2$
5. **Total Sum of Squares:** $\text{SST} = \sum_i (y_i - \bar{y}_i)^2$
6. R^2 **Error :**

$$R^2 = 1 - \frac{\text{MSE (model)}}{\text{MSE(baseline)}} \qquad R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

7. **Adjusted R^2** :

$$R_a^2 = 1 - \left[\left(\frac{n-1}{n-k-1}\right)(1-R^2)\right]$$

Variance, R^2 and the Sum of Squares

- The **total sum of squares**: $\text{SS}_{\text{total}} = \sum_i (y_i - \bar{y})^2$
- This scales with variance: $\text{var}(Y) = \frac{1}{n}\sum_i (y_i - \bar{y})^2$
- The **regression** sum of squares:

$\text{SS}_{\text{regression}} = \sum_i (\hat{y}_i - \bar{y})^2$, $\rightarrow$ nVar(predictions)

- The **residual** sum of squares (squared errro):

$\text{SS}_{\text{residual}} = \sum_i (y_i - \hat{y}_i)^2$, $\rightarrow$ nVar(ϵ)

Note: $\bar{\epsilon} = 0, E[\hat{y}] = \bar{y}$

$$\text{SS}_{\text{total}} = \text{SS}_{\text{regression}} + \text{SS}_{\text{residual}}$$

$$R^2 = 1 - \frac{\text{SS}_{\text{residual}}}{\text{SS}_{\text{total}}} = \frac{\text{SS}_{\text{regression}}}{\text{SS}_{\text{total}}} = \frac{nVar(\text{preds})}{nVar(Y)} = \frac{Var(\text{preds})}{Var(Y)}$$

$\rightarrow$ **Explained Variance**: R^2 quantifies how much of the variability in the outcome (**dependent**) variable can be explained by the predictor (**independent**) variables.

$\rightarrow$ **Goodness of Fit**: A higher R^2 value generally suggests a better fit of the model to the data, meaning the model's predictions are closer to the actual values.

$\rightarrow$ An R^2 of 1 indicates a perfect fit, where the model explains all the variability, while an R^2 of 0 indicates that the model explains none of the variability.

$\rightarrow$ R^2 **is not valid for nonlinear models** as

$$\text{SS}_{\text{residual}} + \text{SS}_{\text{error}} \neq \text{SS}_{\text{total}}$$

$\rightarrow$ **Drawback**: R-squared will always increase when a new predictor variable is added to the regression model, that's why **adjusted** R^2 used

Optimization

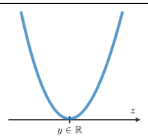
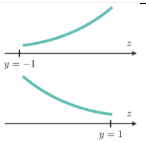
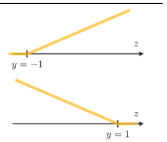
Almost every machine learning method has an optimization algorithm at its core.

$\rightarrow$ **Hypothesis :** The hypothesis h_θ is the model that we choose. For a given input data $x^{(i)}$ the model prediction output is $h_\theta(x^{(i)})$.

$\rightarrow$ **Loss function :** $L : (\hat{y}, y) \in R \times Y \longmapsto L(\hat{y}, y) \in R$ that takes as inputs the **predicted** $\hat{y}$, the **actual** y, and outputs how different they are.

In another way, The **loss function** computes the **distance** or **difference** between the **predicted** output $\hat{y}$ of the algorithm and the **actual** output y.

The common loss functions are summed up in the table below:

Least squared error	**Logistic loss**	**Hinge loss**
$\frac{1}{2}(y-\hat{y})^2$	$\log(1+\exp(-y\hat{y}))$	$\max(0, 1-y\hat{y})$
Linear Regression	Logistic Regression	SVM
		

$\rightarrow$ **Cost function :** The cost function J is commonly **used to know the performance of a model**, and is defined with the loss function L as follows:

$$J(\theta) = \sum_{i=1}^{m} L(h_\theta(x^{(i)}), y^{(i)})$$

- Cost function for **regression**: Mean Squared Error (MSE), Mean Absolute Error (MAE), Huber Loss, Log-Cosh Loss
- Cost function for **classification**: Binary Cross-Entropy Loss, Categorical Cross-Entropy Loss, Sparse Categorical Cross-Entropy Loss, Hinge Loss, Squared Hinge Loss.

Convex & Non-convex

A **convex** function is one where a line drawn between any two points on the graph lies on or above the graph. It has **one minimum**. A **non-convex** function is one where a line drawn between any two points on the graph may intersect other points on the graph. It characterized as "wavy"

$\rightarrow$ When a **cost function is non-convex**, it means that there is a likelihood that the function may find **local minima** instead of the **global minimum**, which is typically undesired in machine learning models from an optimization perspective.

General Optimization Steps

1. Understand data (features and outcome variables) $\rightarrow$ 2. Define loss (or gain/utility) function $\rightarrow$ 3. Define predictive model $\rightarrow$ 4. Search for parameters that minimize the loss function

Gradient Descent

Gradient Descent is used to **find the coefficients** of f that **minimizes a cost function**.

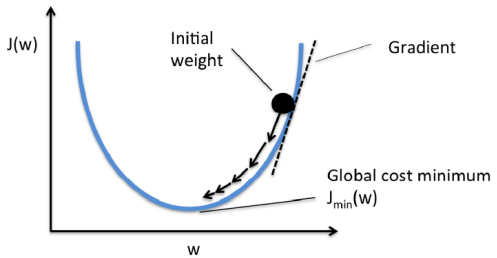
$\rightarrow$ **Time Complexity**: $O(n \cdot m) \rightarrow n$ is no. of data points, m no. of features. If you run for k iterations the total complexity becomes $O(k \cdot n \cdot m)$

$\rightarrow$ It minimizes the average loss by moving iteratively in the direction of steepest descent, controlled by the learning rate γ (step size).

Procedure:

1. Intialization $\quad \theta = 0$ (coefficients to 0 or random)
2. Calculate cost $\quad J(\theta) = \text{evaluate}\big(f(\text{coefficients})\big)$
3. Gradient of cost $\quad \frac{\partial}{\partial \theta_j} J(\theta)$ we knows the uphill direction
4. Update coeff $\quad \theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ we go downhill

The cost updating process is repeated until convergence (minimum found).



Tips : For effective gradient descent, select an optimal learning rate, scale features, initialize parameters wisely, utilize mini-batch processing, monitor convergence, experiment with various optimizers, apply regularization techniques, avoid local minima, visualize loss trends, and tune hyperparameters diligently.

- **Stochastic Gradient Descent** uses a single point to compute gradients, leading to smoother convergence and faster compute speeds.

$\rightarrow$ **Time Complexity**: $O(k \cdot m) \rightarrow m$ is no. of features. In each iteration, SGD computes the gradient using only one data point, leading to $O(k \cdot m)$ for m features.

- **Mini-batch Gradient Descent** trains on small subsets of the data, striking a balance between the approaches. $\rightarrow$ **Time Complexity**: $O(k \cdot b \cdot m)$

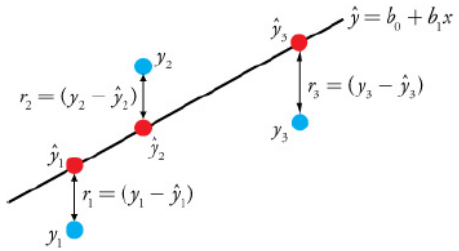
Ordinary Least Squares

General Linear Regression Model: $\hat{y} = \beta_0 + \sum_j \beta_j x_j + \epsilon$

Here, β_j is the j-th coefficient and x_j is the j-th feature.

Ordinary Least Squares - find $\vec{\beta}$ that minimizes **squared error**:

$$\arg\min_{\vec{\beta}} \sum_i (y_i - \hat{y}_i)^2$$



Goal: least-squares solution to : $X\vec{\beta} = \hat{y}$

Solution: solve the normal equations:

$$X^T X\vec{\beta} = X^T\hat{y} \quad \rightarrow \vec{\beta} = (X^T X)^{-1} X^T \hat{y}$$

$$L(\vec{\beta}\,|\vec{X}, \hat{y})$$

$\rightarrow$ Least squares generalizes into **minimizing loss functions**.

$\rightarrow$ This is the heart of machine learning, particularly supervised learning.

Likelihood and Posterior

$$P(\theta|\mathbf{y}) = \frac{P(\mathbf{y}|\theta)\,P(\theta)}{P(\mathbf{y})}$$

- $P(\theta)$ is the prior
- $P(y|\theta)$ is the likelihood – how likely is the data given params θ
- $P(y) = \int P(y|\theta)P(\theta)d\theta$ is a scaling factor (constant for fixed y)
- $P(\theta|y)$ is the posterior.
- We're maximizing likelihood (ML estimator)
- Can also maximize posterior (MAP estimator)
  - When prior is constant, they're the same
  - With lots of data, they're almost the same

$\rightarrow$ Logistic function is trained by **maximizing the log likelihood** of the training data given the model

Maximum Likelihood Estimation (**MLE**)

In MLE, the goal is to estimate the parameters (or coefficients) of a probability distribution by finding the values that **maximize** the likelihood of the observed data.

• **Likelihood Function**: The likelihood function measures the probability of the observed data given the parameters. It helps to evaluate how well different parameters explain the observed data. For a dataset $X = \{x_1, x_2, ...x_n\}$, assumed to be i.i.d.and a parameter θ, the likelihood $L(\theta)$ is:

$$L(\theta|X) = \prod_{i=1}^{n} f(x_i|\theta) \quad \rightarrow L(\theta|X) = P(X|\theta) = \prod_{i=1}^{n} P(x_i|\theta)$$

• **Log-Likelihood**: The natural log of $L(\theta)$ is then taken prior to calculating the maximum because multiplying probabilities can result in very small values and also log is a monotonically increasing function, maximizing the **log-likelihood** log $L(\theta)$ is equivalent to maximizing the likelihood:

$$\log L(\theta|X) = \log P(X|\theta) = \prod_{i=1}^{n} \log P(x_i|\theta)$$

→ MLE is used to find the estimators that **minimized the likelihood function:** $\mathcal{L}(\theta|x) = f_\theta(x)$ density function of the data distribution

In case of **Logistic Regression**:

$$P(Y = 1|X = x) = \hat{y} = \text{logistic}\left(\beta_0 + \sum_j \beta_j x_j\right)$$

The model computes **probability of yes**.

→ What if we want $P(Y = y_i)$, regardless of whether y_i is 1 or 0?

$$P(Y = y_i|X = x_i) = \hat{y_i}^{y_i}(1 - \hat{y_i})^{1-y_i}$$

- $\hat{y_i}$ is model's estimate of $P(Y = 1|X = x_i)$
- $y_i \in \{0, 1\}$ is outcome
- $\hat{y_i}^{y_i}$ is $\hat{y_i}$ if $y_i = 1$, and 1 if $y_i = 0$

Conditioning on Parameters

Fuller definition - condition on parameters $\vec{\beta}$ and write function:

$$P(Y = 1|x, \vec{\beta}) = \hat{y} = m(x, \vec{\beta}) = \text{logistic}(...)$$

Likelihood Function

Given data $\mathbf{X} = \langle x_1, ..., x_n \rangle$, $\mathbf{y} = \langle y_1, ..., y_n \rangle$ and parameters $\hat{\beta}$

$$\text{Likelihood}(\mathbf{y}, \mathbf{X}, \vec{\beta}) = P(\mathbf{y}, \mathbf{X}|\vec{\beta}) = \prod_i P(y_i|x_i, \vec{\beta})$$

By joint conditional probability,

$$P(\mathbf{y}, \mathbf{X}|\vec{\beta}) = P(\mathbf{X}|\vec{\beta}) \cdot P(\mathbf{y}|\mathbf{X}, \vec{\beta})$$

But $\mathbf{X}$ is independent of params, so $P(\mathbf{X}|\vec{\beta}) = P(\mathbf{X})$. And $\mathbf{X}$ is fixed, so $P(\mathbf{X})$ is an (unknown) constant.

$$\log \text{Likelihood}(\mathbf{y}, \mathbf{X}, \vec{\beta}) = \log P(\mathbf{X}) \prod_i P\left(y_i|x_i, \vec{\beta}\right)$$

$$= \log P(\mathbf{X}) + \sum_i \log P\left(y_i|x_i, \vec{\beta}\right)$$

Maximum Likelihood Estimator

$$\arg\max_{\vec{\beta}} \sum_i \log P\left(y_i|x_i, \vec{\beta}\right)$$

$$P(Y = y_i|X = x_i) = \hat{y_i}^{y_i}(1 - \hat{y_i})^{1-y_i}$$

$$\log P(Y = y_i|X = x_i) = y_i \log \hat{y_i} + (1 - y_i) \log(1 - \hat{y_i})$$

Model log likelihood is sum over training data. Applicable to **any** model where $\hat{y} = P(Y = 1|x)$

Bayesian Estimation - Maximum a Posterior (**MAP**)

MAP estimation seeks to find the parameters θ that maximize the posterior distribution, which assumes a "prior distribution $P(\theta)$"

$$\hat{\theta}_{MAP} = \underset{\theta}{\text{argmax}}\ P(\theta|X) = \underset{\theta}{\text{argmax}}\ \frac{P(X|\theta)P(\theta)}{P(X)}$$

Since $P(X)$ does not depend on θ,

$$\hat{\theta}_{MAP} \approx \underset{\theta}{\text{argmax}}\ P(X|\theta)P(\theta)$$

In logistic regression, MAP can be applied by introducing a prior on the model parameters $P(\theta)$

Linear Algorithms

Regression

→ Regression predicts (or estimates) a continuous variable

Dependent variable Y, **Independent** variable(s) X

→ compute estimate $\hat{y} \approx y$

$$\hat{y_i} = \beta_0 + \beta_1 x_i$$

$$y_i = \hat{y_i} + \epsilon_i$$

Here, β_0 is intercept, β_1 is slope and ϵ is residuals. The goal is to learn β_0, β_1 to minimize $\sum \epsilon_i^2$ (least squares)

Linearity: A linear equation of $k + 1$ variables is of the form:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$$

It is the **sum** of **scalar multiples** of the individual variables - aline!

→ Linear models are remarkably capable of transforming many non-linear problems into linear.

Linear Regression

$$\hat{y_i} = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} \cdots + \beta_p x_{ip} + \epsilon$$

$$\hat{y_i} = \beta_0 + \sum_{i=1}^{n} \sum_{j=1}^{p} \beta_j x_{ij}$$

Here, n is total no. of observation, $\hat{y_i}$ is dependent variable, x_{ij} is explanatory variable of j-th features of the i-th observation. β_0 is intercept or usually called **bias** coefficient.

Assumptions:

→ Linear models make **four** key assumptions necessary for inferential validity.

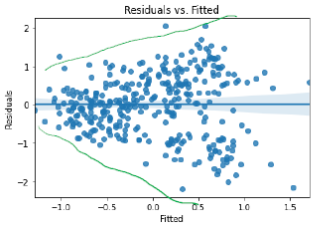
• **Linearity** — outcome y and predictor X have linear relationship.

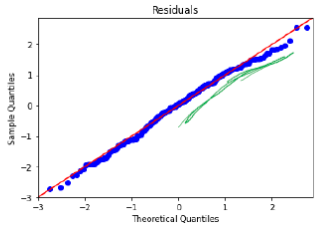
• **Independence** — observations are independent of each other

- Independent variables (features) are not highly correlated with each other → **Low multicollinearity**

• **Normal errors** — residuals are normally distributed - **check** with Q-Q plots. **Violation** means line (in Q-Q plots) still fits but p-value and CIs are unreliable

• **Equal variance** — residuals have constant variance (called homoskedasticity; violation is heteroskedasticity) - **check** scatterplot or regplot between residuals vs. fitted. **Violations** means model is failing to capture a systematic effect. → These violations are problem only for inference not for prediction



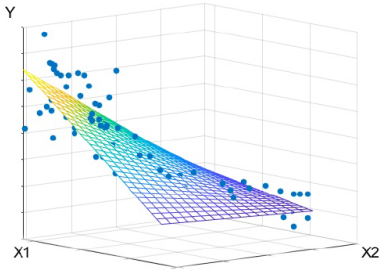


Variance Inflation Factor : Measures the severity if multicollinearity → $\boxed{\frac{1}{1 - R_i^2}}$, where R_i^2 is found by regressing X_i aganist all other variables (a common VIF cutoff is 10)

Learning: Estimating the coefficients β from the training data using the optimization algorithm **Gradient Descent** or **Ordinary Least Squares**.

Ordinary Least Squares - where we find $\vec{\beta}$ that minimizes **squared error**:

$$\arg\min_{\vec{\beta}} \sum_i (y_i - \hat{y_i})^2$$



→ The dimension of the hyperplane of the regression is its **complexity**.

Variations: There are extensions of Linear Regression training called **regularization** methods, that aim to **reduce the complexity** of the models or to address over-fitting in ML. *The regularizer is not dependent on the data.* → In relation to the bias-variance trade-off, regularization aims to decrease complexity in a way that **significantly reduces variances** while only **slightly increasing bias**.

→ Standardize numeric variables when using regularization because to ensure that 0 is a neutral value, so a low coefficient means "little effect when deviating from average". So values, and therefore coefficients, are on the same scale (# of standard deviations), to properly distribute weight between them.

→ **Multicollinearity** → correlated predictors. **Problem**: Which coefficient gets the common effect? To **solve**: Loss and Regularization comes.

- **Ridge Regression** (L2 regularization): where OLS is modified to minimize the **squared** sum of the coefficients

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$

→ Prevents the weights from getting too large (L2 norm). If lambda is **very large then it will add too much weight** and it will lead to **under-fit**.

$$\lambda \propto \frac{1}{\text{model variance}}$$

- **Lasso Regression** (L1 regularization) : where OLS is modified to minimize the sum of the coefficients

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|$$

where p is the no. features (or dimensions), $\lambda \geq 0$ is a tuning parameters to be determined.

→ Lasso shrinks the less important feature's coefficient to **zero** thus, removing some feature altogether. If lambda is **very large value will make coefficients zero** hence it will **under-fit**.

→ L1 is less likely to shrink coefficients to 0. Therefore L1 regularization leads to sparser models.

Logistic Regression

Log-Odds and Logistics

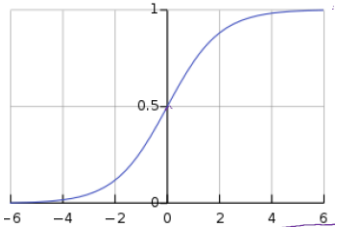
• **Odds**
The probability of success $P(S)$: $0 \leq p \leq 1$
→ The odds of success are defined as the ratio of the **probability of success** over the **probability of failure**.
The **odds** of success: $\text{Odds}(S) = \frac{P(S)}{P(S^c)} = \frac{P(S)}{1-P(S)}$
→ Ex: Odds(failure) = x → means x:1 aganist success

• **Log Odds** or **logit** →

$$\log\ \text{Odds}(A) = \log\frac{P(A)}{1-P(A)} = \log P(A) - \log(1-P(A))$$

• **Logistic**: The inverse of the logit (logit^{-}1):

$$\text{logistic}(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$



sigmoid or logistic curve.

→ Odds are another way of representing probabilities.
→ The logistic and logit functions convert between probabilities and log-odds.

• **General Linear Models (GLMs)**:

$$\hat{y}_i = g^{-1}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} \cdots + \beta_p x_{ip})$$

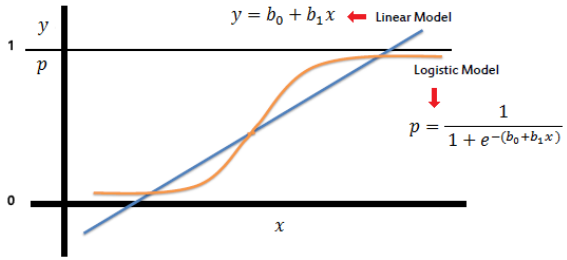
$$\hat{y}_i = g^{-1}\left(\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}\right)$$

Here, g is a link function
• **Counts**: Poision regression, log link func
• **Binary**: Logistic regression, logit link func and g^{-1} is logistic func
→ In logistic regression, a linear output is converted into a probability between 0 and 1 using the sigmoid or logistic function.

$$P(y_i = 1|X) = \hat{y}_i = \text{logistic}\left(\beta_0 + \sum_j \beta_j x_{ij}\right)$$

$$p(X) = \frac{e^{\beta_0+\beta_1 x_1+\cdots+\beta_i x_i}}{1+e^{\beta_0+\beta_1 x_1+\cdots+\beta_i x_i}} = p(y=1\,|\,X)$$

Note : **Coefficients** are linearly related to odds, such that a one unit increase in x_1 affects odds by e^{β_1}.
Note : The **coefficients in logistic regression are interpreted** in terms of their effect on the log-odds of the outcome, and the exponentiated coefficients (odds ratios) provide a clearer understanding of the change in odds associated with each predictor.



The representation below is an equation with binary output, which actually models the probability of default class:

Assumptions:
- Linear relationship between X and log-odds of Y
- Observations must be independent to each other
- Low multicollinearity

Learning: Learning the logistic regression coefficients is done by:
→ **Minimizing** the **logistic loss function**

$$\arg\min_{\vec{\beta}} \sum_i \log(1+\exp(-y_i\vec{\beta}x_i))$$

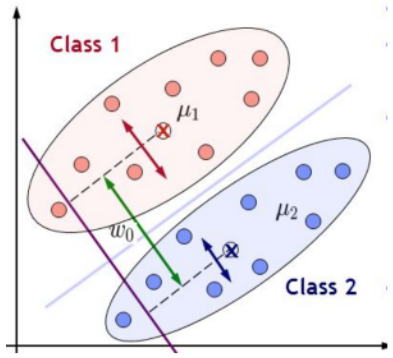
→ **Maximizing** the **log likelihood** of the training data given the model

$$\arg\max_{\vec{\beta}} \sum_i \log P\left(y_i|x_i,\vec{\beta}\right)$$

Linear Discriminant Analysis

For **multiclass classification**, LDA is the preferred linear technique.
Representation: LDA representation consists of statistical properties calculated for **each class**: **means** and the **covariance** matrix:

$$\mu_k = \frac{1}{n_k}\sum_{i=1}^{n} x_i \qquad \sigma^2 = \frac{1}{n-k}\sum_{i=1}^{n}(x_i-\mu_k)^2$$



LDA assumes **Gaussian** data and attributes of **same** σ^2. **Predictions** are made using **Bayes Theorem**:

$$P(y=k\,|\,X=x) = \frac{P(k)\times P(x|k)}{\sum_{l=1}^{k} P(l)\times P(x|l)}$$

to obtain a discriminate function (latent variable) for each class k, estimating $P(x|k)$ with a **Gaussian** distribution:

$$D_k(x) = x\times\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + ln(P(k))$$

The class with largest **discriminant value** is the **output class**.

Variations:

1. **Quadratic DA:** Each class uses its own variance estimate
2. **Regularized DA:** Regularization into the variance estimate.

Data preparation for Linear Algorithm

1. **Data Transformation:** Linear algorithms require data to have a linear relationship between features and the target variable. Often, transformations like log or polynomial are applied to make the data fit a linear pattern.
2. **Feature Engineering:** Feature engineering is crucial. Polynomial features or interaction terms may need to be added manually to capture relationships in the data.
3. **Handling Outliers:** Linear models are sensitive to outliers. Detecting and either removing or transforming outliers is important, as they can significantly influence the results.
4. **Rescaling:** Rescaling features (standardization or normalization) is required to ensure that all features contribute equally to the model. Algorithms such as linear regression perform better with rescaled data.
5. **Assumptions:** Linear algorithms assume a linear relationship between the input features and the output variable. Violating this assumption can lead to suboptimal performance.

Advantages of Linear Algorithms

1. **Simplicity and Interpretability:** Easy to understand and interpret results (e.g., coefficients indicate feature importance).
2. **Computational Efficiency:** Faster to train and predict, especially with large datasets.
3. **Less Prone to Overfitting:** With proper regularization (like Lasso or Ridge), linear models can generalize well on unseen data.
4. **Strong Theoretical Foundation:** Well-established statistical properties and a solid theoretical framework.
5. **Works Well with Linearly Separable Data:** Performs well when the relationship between features and target variable is linear.

Nonlinear Algorithms

All Nonlinear Algorithms are non-parametric and more flexible. They are not sensible to outliers and do not require any shape of distribution.

Naive Bayes Classifier

Naive Bayes is a **classification** algorithm interested in selecting the **best hypothesis** h given data X **assuming that the features of each data point are all independent**

Representation: The representation is based on Bayes Theorem:

$$P(Y|X) = \frac{P(X|Y) \times P(Y)}{P(X)}$$

With naive hypothesis,
$P(Y|X) = P(x_1, x_2, \cdots, x_i|\, Y) = P(x_1|Y) \times P(x_1|Y) \times \cdots P(x_i|Y)$

$$P(X|Y) = \prod_{i=1}^{n} P(x_i|\, Y)$$

The prediction is the maximum a **posterior hypothesis**:

$$\max\left(P(Y|X)\right) = \max\left(P(X|Y) \times P(Y)\right)$$

here, the denominator is not kept as it is only for normalization.

Learning: Training is **fast** because only **probabilities** need to be calculated:

$$P(Y) = \frac{\text{instances}_Y}{\text{all instances}} \qquad P(x|Y) = \frac{\text{count}(x \wedge Y)}{\text{instances}_Y}$$

Variations: Gaussian Naive Bayes can extend to numerical attributes by assuming a Gaussian distribution. Instead of $P(x|h)$ are calculated with $P(h)$ during **learning**, and MAP for **prediction** is calculated using Gaussian PDF

$$f(x\,|\,\mu(x), \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

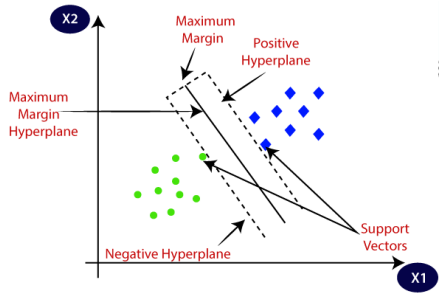
$$\mu(x) = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu(x))^2}$$

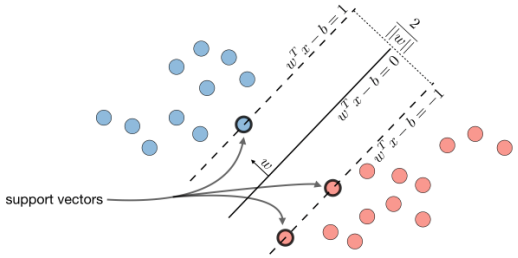
Support Vector Machines

SVM is a go-to for high performance with little tuning. Compares extreme values in your dataset.

In SVM, a **hyperplane** (or decision boundary: $w^T x - b = 0$) is selected to **separate the points** in the input variables space by their class, with the **largest margin**. The closest datapoints (defining the margin) are called the **support vectors**.

→ The **goal** of a support vector machine is to find the optimal separating hyperplane which maximizes the **margin** of the training data.





The **prediction** function is the signed **distance** of the new input x to the separating hyperplane w, with b the bias:

$$f(x) = \langle w, x\rangle + b = w^T x + b$$

→ **Optimal margin classifier:** The optimal margin classifier h is such that:

$$h(x) = \text{sign}(w^T x - b)$$

where $(w, b) \in \mathcal{R}^n \times \mathcal{R}$ is the solution of the following optimization problem:

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

such that

$$y^{(i)}(w^T x^{(i)} - b) \geq 1$$

Learning:

→ **Hinge loss :** The hinge loss is used in the setting of SVMs and is defined as follows:

$$L(\hat{y}, y) = [1 - y\hat{y}]_+ = \max(0,\, 1 - y\hat{y})$$

→ **Lagrangian :** We define the Lagrangian $\mathcal{L}(w, b)$ as follows:

$$\mathcal{L}(w, b) = f(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

Lagrange method is required to convert constrained optimization problem into unconstrained optimization problem. The goal of above equation to get the optimal value for **w** and **b**.

$$\lambda\|\vec{w}\|^2 + \left[\frac{1}{n}\sum_{i=1}^{n} \max\left(0,\, 1 - y_i(\vec{w}\cdot\vec{x_i} - b)\right)\right]$$

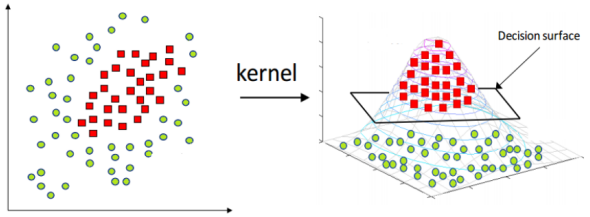
The **first term** is the regularization term, which is a technique to avoid overfitting by penalizing large coefficients in the solution vector. The **second term**, hinge loss, is to penalize misclassifications. It measures the error due to misclassification (or data points being closer to the classification boundary than the margin). The λ is the regularization coefficient, and its major role is to determine the trade-off between increasing the margin size and ensuring that the x_i lies on the correct side of the margin.

→ **Kernel :** A kernel is a way of computing the dot product of two vectors **x**x and **y**y in some (possibly very high dimensional) feature space, which is why kernel functions are sometime called "generalized dot product". The kernel trick is a method of using a linear classifier to solve a non-linear problem by transforming linearly inseparable data to linearly separable ones in a higher dimension.

Given a feature mapping ϕ, we define the kernel K as follows:

$$K(x, z) = \phi(x)^T \phi(z)$$

In practice, the kernel K defined by $K(x, z) = e^{-\frac{\|x-z\|^2}{2\sigma^2}}$ is called the Gaussian kernel and is commonly used.



Note: we say that we use the "kernel trick" to compute the cost function using the kernel because we actually don't need to know the explicit mapping ϕ, which is often very complicated. Instead, only the values $K(x, z)$ are needed.

Variations:
SVM is implemented using various kernels, which define the measure between new data and support vectors:

1. **Linear** (dot-product): $K(x, x_i) = \sum(x \times x_i)$
2. **Polynomial:** $K(x, x_i) = 1 + \sum(x \times x_i)^d$
3. **Radial:** $K(x, x_i) = e^{-}\gamma \sum(x - x_i)^2$

Hyperparameters: regularization parameter (C) and the kernel parameters (such as gamma for the RBF kernel).

K-Nearest Neighbors

If you are similar to your neighbors, you are one of them. KNN uses the **entire training data**, **no training** is required.

Note: Higher $k \rightarrow$ higher the bias, Lower $k \rightarrow$ higher the variance.

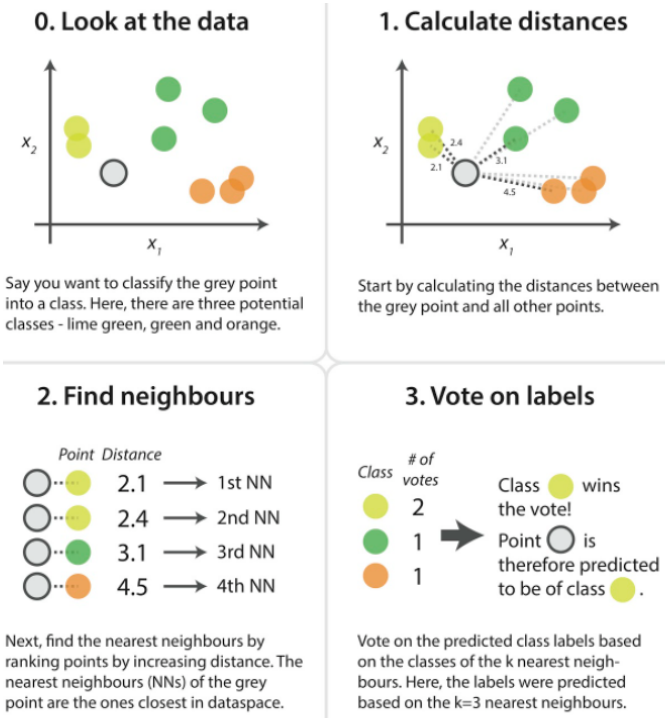
• **Choice of k is very critical** → A small value of **k** means that noise will have a higher influence on the result. → A large value of **k** make everything classified as the most probable class and also computationally expensive.

→ A simple approach to **select k** is set $k = \sqrt{n}$ or **cross-validating** on small subset of training data (validation data) by **varying** values of k and **observing** training - validation error.

→ Minkowski Distance = $\left(\sum |a_i - b_i|^p\right)^{\frac{1}{p}}$

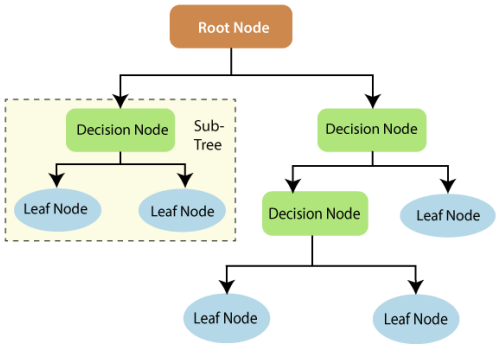
- p=1 gives **Manhattan distance** $\sum |a_i - b_i|$
- p=2 gives Euclidean distance $\sqrt{\sum (a_i - b_i)^2}$

→ Hamming Distance - count of the differences between two vectors, often used to compare categorical variables.



Classification and Regression Trees (CART)

Decision Tree is a **Supervised learning** technique that can be used for both **Classification** and **Regression** problems.



Here, each **node** represents a question about the data, and the **branches** from each node represent the possible answers.
→ **Root Node**: It is the very first node (parent node), and denotes the whole population, and gets split into two or more Decision nodes based on the feature values.
→ **Decision Node**: At each decision node, the algorithm chooses the best feature and threshold to split the data, aiming to create the most homogeneous subsets. They have **multiple** branches.
→ This process continues until a stopping condition is met (like maximum depth or pure leaves).
→ **Leaf Node**: The final predictions are made at the leaf nodes, which represent the outcome of those **decisions**.
→ **Sub-Tree**: A branch is a subdivision of a complete tree.

At each leaf node, CART predicts the most frequent category, assuming false negative and false positive costs are the same.
→ The splitting process handles multicollinearity and outliers.
→ Trees are prone to high variance, so tune through CV.

Note: In decision trees, the **depth of the tree** determines the variance. Decision trees are commonly pruned to control variance

• CART for regression **minimizes SSE** by splitting data into sub-regions and predicting the average value at leaf nodes. The complexity parameter cp only keeps splits that reduce loss by at least cp (small $cp \rightarrow$ deep tree).

• CART for classification **minimizes the sum of region impurity**, where p_i is the probability of a sample being in category i. Possible measures, each with a max impurity of 0.5.

- $\text{Gini Impurity / Gini Index / Gini Coefficient} = 1 - \sum(p_i)^2$
- $\text{Cross Entropy} = \sum(p_i)\log_2(p_i)$

Procedure:

1. Calculate entropy of the **outcome** classes (c)
$$E(T) = \sum_{i=1}^{c} -p_i \log_2 p_i$$
2. The dataset is split on the different attributes. The entropy of each branch is calculated. Then it is added proportionally to get **total entropy for the split**. The resulting entropy is subtracted from the entropy before the split.
$$\text{Gain}(T, X) = \text{Entropy}(T) - \text{Entropy}(T, X)$$
3. Choose attributes with **largest information Gain** as the **decision node**, divide the dataset by its branches and repeat the same process on **every branch**.
4. A branch with **entropy of 0** is a leaf node
5. A branch with **entropy more than 0** needs futher splitting.
6. ID3 algorithm is run recursively on the **non-leaf branches**, until all data is classified

Hyperparameters: The most common **Stopping Criterion** for splitting is a minimum of **training observations per node**, maximum depth of the tree

Ensemble Algorithms

Ensemble methods combine multiple, simpler algorithms (weak learners) to obtain better performance algorithm.

Bagging	Boosting
	AdaBoost
Random Forest	Gradient Boosting
	XGBoost

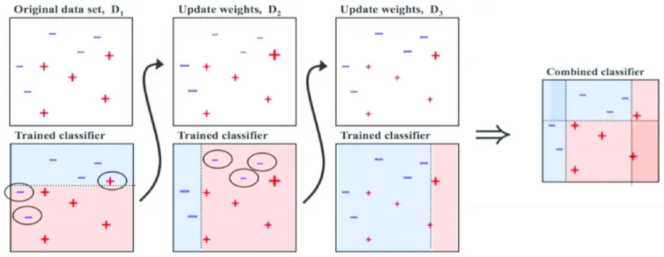
• **Bagging**
→ It involves **parallel** training of multiple models independently on **different subsets** of the data. These subsets of data are drawn using the bootstrap technique.
→ Then averaging their predictions (for regression) or majority voting (for classification).
→ It can **reduce the variance** and prevent overfitting by averaging out the errors of individual models.

→ **Bootstrapping** is drawing random **sub-samples** (sampling **with replacement**) from a large **sample** (available data) to estimate quantity (parameters) of a unknown population by **averaging the estimates** from these **sub-samples**.

Random Forest

→ **Bagged Decision Trees**: Each DT may contain different no. of rows and different no. of features.
→ Individual DTs may face **overfitting** i.e. have **low bias** (complex model) but **high variance**, by ensembling a lot of DTs we are going to **reduce the variance**, while not increasing the bias.
Hyperparameters: number of trees, maximum depth of the trees

• **Boosting**
→ It involves **sequentially** training of multiple models, where each model tries to correct the errors of the previous ones.



AdaBoost

• Uses the **same training samples** at each stage
• "Weakness" = Misclassified data points
• **Learning Focus**: Primarily reduces bias (increases variance) by focusing on misclassified instances **Algorithm**:

1. **Initialize Weights**: Assign **equal weight** to each of the training data
2. **Train weak model and Evaluate**: Provide this as input to the weak model and identify the wrongly classified data points
3. **Adjust Weights**: Increase the weight of wrongly classified data points
4. **Combined Models**: Combine the weak models using a weighted sum, where weights are based on the accuracy of each learner.
5. Repeat steps 2-4 for a predefined number of iterations or until the error is minimized.

• **Limitations**: Sensitive to noisy data and outliers since misclassified points are given more focus.
Hyperparameters: number of estimators, learning rate.

Gradient Boosting

• Uses the **different training samples** at each stage
• "Weakness" = Residuals or Erros
• **Learning Focus**: Instead of adjusting weights, it optimizes the model by minimizing a loss function (e.g., mean squared error for regression).
Algorithm:

1. **Initialize Model**: Start with an initial model (e.g., a constant value). Let's say Avg.
2. **Compute Residuals**: Calculate the residuals (errors) of the current model.
3. **Train Weak Learner**: Train a weak learner on the residuals.
4. **Update Model**: Add the weak learner to the model with a certain learning rate.
5. Repeat steps 2-4 for a fixed number of iterations or until the model converges.

• **Limitations**: Slower to train and more prone to overfitting without careful tuning.
Hyperparameters: learning rate, number of boosting stages, maximum depth of individual trees.

XGBoost
• Enhances gradient boosting by making it faster, more efficient, and more accurate..
→ **Execution speed**: Parallelization (It will use all cores of CPU), Cache optimization, Out of memory (Data size bigger than memory)
→ **Model performance:**
- Adds regularization to balance the trade-off between fitting the training data and maintaining model simplicity.
- **Auto pruning**: Prevents trees from growing too large, improving generalization and reducing the risk of overfitting.
- During training, model learns the optimal way to split data with **missing values** as well as model learns from the patterns of **missing data** and adjusts the decision boundaries accordingly.
- Efficient handling of sparse data
- Flexible: Supports a variety of loss functions and custom objective functions
Hyperparameters: Learning Rate, Number of Trees, Maximum Depth, Min Child Weight, Subsample, Booster Type, Early Stopping Rounds

Data Preparation for Non-Linear Algorithms

1. **Data Transformation:** Non-linear algorithms can naturally model non-linear relationships without the need for data transformations, as they are capable of capturing complex patterns.
2. **Feature Engineering:** Non-linear models are less reliant on manual feature engineering. Algorithms like decision trees or neural networks can automatically capture feature interactions and non-linear relationships.
3. **Handling Outliers:** Non-linear algorithms, such as decision trees and support vector machines, are generally more robust to outliers compared to linear models.
4. **Rescaling:** Some non-linear models (e.g., neural networks, support vector machines) benefit from rescaling, while others (e.g., decision trees, random forests) do not require rescaling.
5. **Assumptions:** Non-linear algorithms do not assume a linear relationship between inputs and outputs, allowing them to model more complex relationships in the data without predefined structures.

Advantages of Non-Linear Algorithms

1. **Captures Complex Relationships:** Can model intricate patterns and interactions in the data that linear algorithms might miss.
2. **Higher Flexibility:** Adaptable to various data types and structures (e.g., images, text).
3. **Improved Accuracy:** Often yields better performance on non-linear datasets by fitting the data more closely.
4. **Handles High Dimensionality:** Suitable for high-dimensional spaces, especially with techniques like kernel methods or neural networks.
5. **No Assumption of Linearity:** Does not require prior knowledge of the relationship between input features and output, making it versatile.
6. **Feature Engineering Not Always Necessary:** Some non-linear algorithms, like tree-based models, automatically capture interactions and non-linearities without explicit feature engineering.

Unsupervised Machine Learning

1. Clustering
2. Dimension Reduction
3. Association Rule Mining
4. Graphical Modelling and Network Analysis

Clustering

Grouping objects into meaningful **subets** or, **clusters**. → Objects within each cluster are similar.
Clustering Algorithms:

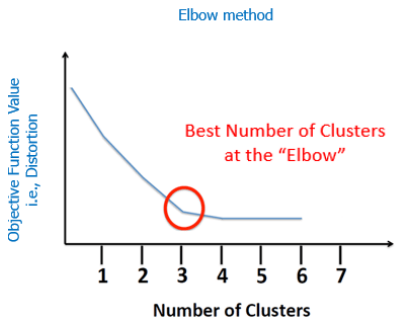
1. Partition-based methods
   (a) K-means clustering
   (b) Fuzzy C-Means
2. Hierarchical methods
   (a) Agglomerative Clustering
   (b) Divisive Clustering
3. Density-based methods
   (a) Density-Based methods (DBSCAN)

K-means clustering

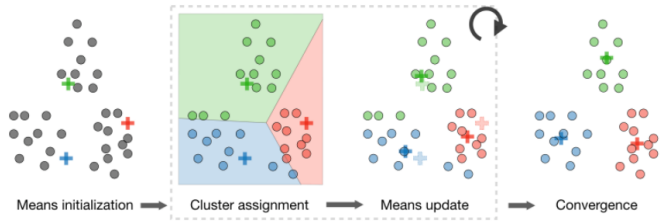
The objective of K-means clustering is to **minimize** total **intra-cluster** or, the **squared error function.**

$$\text{Objective function} \rightarrow \boxed{J = \sum_{j=1}^{K}\sum_{i=1}^{n}\|X_i^{(j)} - C_j\|^2}$$

Here, K is No. of clusters, n is No. of cases, C_j is centroid for cluster j



1. Divide data into K clusters or groups.
2. Randomly select **centroid** for each of these K clusters.
3. Assign data points to their closest cluster **centroid** according to Euclidean/ Square Euclidean/Manhattan/Cosine
4. Calculate the **centroids** of the newly formed clusters.
5. Repeat steps 3 and 4 until the **same centroids** (convergences) are assigned to each cluster.



→ K-means **always converges** (mostly to local minimum not to global minimum)
• **How to choose K number of clusters in K-Means algorithm?**
→ The maximum possible number of clusters will be equal to the number of observations in the dataset.

Hierarchial Clustering

Agglomerative method: "Bottom-up"

1. Compute the distance or, **proximity matrix**
2. **Initialization:** Each observation is a cluster
3. **Iteration:** Merge two clusters which are most similar; until **all observations** are merged into a single cluster.

Divisive method: "Top-down"

1. Compute the distance, or **proximity matrix**
2. **Initialization:** All objects stay in one cluster
3. **Iteration:** Select a cluster and split it into two sub-cluster until each leaf cluster contains only **one observation**.

Proximity (distance) matrix
→ **Single or ward linkage:** Minimize within cluster distance

$$\boxed{L(C_1, C_2) = min\left[D\left(X_i^{C_1}, X_j^{C_2}\right)\right]}$$

→ **Complete linkage:** Longest distance between two points in each cluster. Minimize maximum distance of between cluster pairs

$$\boxed{L(C_1, C_2) = max\left[D\left(X_i^{C_1}, X_j^{C_2}\right)\right]}$$

→ **Average linkage:** Minimize average distance between cluster pairs

$$\boxed{L(C_1, C_2) = \frac{1}{n_{C_1} n_{C_2}} \sum_{i=1}^{n_{C_1}} \sum_{j=1}^{n_{C_2}} \left[D\left(X_i^{C_1}, X_j^{C_2}\right)\right]}$$

DBSCAN

→ Two parameters: ε - distance, minimum points
→ Three classifications of points:
- **Core:** has atleast minimum points within ε - distance including itself
- ε - distance has less than minimum points within ε - distance but can be reached by clusters.
- **Outlier:** point that cannot be reached by cluster

Procedure:

1. Pick a random point that has not been assigned to a cluster or, designated as an **Outlier**. Determine if it is a **Core Point**. If not, label the point as **Outlier**.

2. Once a **Core Point** has been found, add all directly reachable to its cluster. Then do **neighbor jumps** to each reachable point and add them to the cluster. If an **Outlier** has been added, label it as a **Border Point**.
3. Repeat these steps until all points are assigned a cluster or, label as **Outlier**.

Dimensionality Reduction Methods

Reduce the number of input variables (attributes or features) in dataset.

Principle Component Analysis (PCA)

PCA combines highly correlated variables into a new, smaller set of constructs called *principal components*, which capture most of the variance present in the data.

- Dimensionality reduction
- Feature extraction
- Data visualization

Procedure:

1. **Standarize** the data: $\boxed{Z = \frac{X - mean}{SD}}$
2. Calculate **covariance-matrix** of the standarized data
$$\boxed{V = \text{cov}(Z^T)}$$
3. Find **eigen-values** and **eigen-vectors** from the covariance-matrix
$$\boxed{\text{values, vectors} = \text{eig}(V)}$$
4. Feature vectors; It is simply the matrix that has columns, the eigen-vectors of the components that we decide to keep.
5. Project data → $\boxed{Z_{\text{new}} = \text{vectors}^T \cdot Z^T}$

Association Rule Mining

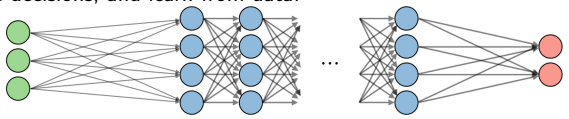
"Market Basket Analysis" → It uses Machine Learning models to analyze data for patterns or, co-occurence in a database.

Graphical Modelling and Network Analysis

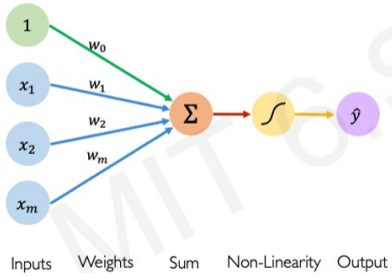
"Bayesian Networks"

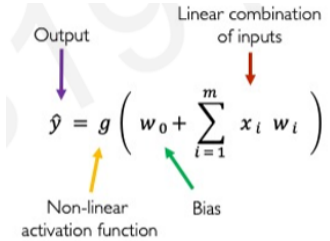
Neural Network

A neural network is a type of machine learning model that mimics the structure and function of the human brain to recognize patterns, make decisions, and learn from data.

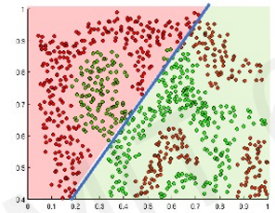


→ **Input Layer**: The first layer that receives the input data. Each neuron in this layer corresponds to a **feature** of the input data.
→ **Hidden Layers**: Layers between the input and output layers where the network learn complex patterns.
→ **Output Layer**: The final layer that produces the network's output, such as a prediction or classification.
• **Perceptron** - the foundation of a neural network, and it is a single-layer neural network. An Artificial Neuron is a basic building block of a neural network.
• **Neural Network** - a multi-layer **perceptron**

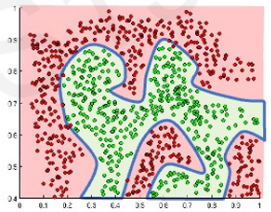




→ **Weights**: are the real values that are attached with each input/feature and they convey the importance of that feature in predicting the final output.
→ **Bias**: is used for shifting the activation function towards left or right.
→ **Summation Function**: used to bind the weights and inputs together and calculate their sum.
→ **Activation Function**: decides whether a neuron should be activated or not, and it introduces **non-linearities** into the network which makes input capable of learning and performing more complex tasks.

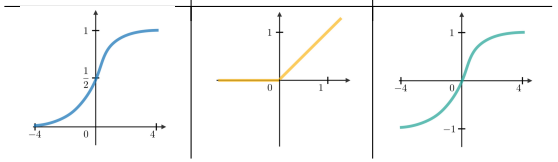


Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Sigmoid	ReLU	Tanh
$\frac{1}{1+e^{-z}}$	$\max(0, z)$	$\frac{e^z - e^{-z}}{e^z + e^{-z}}$



→ **Softmax** - used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes. These probabilities sum to 1 → $\frac{e^z i}{\sum e^z}$

→ If there is more than one 'correct' label, the sigmoid function provides probabilities for all, some, or none of the labels.

How Neural Networks Work?

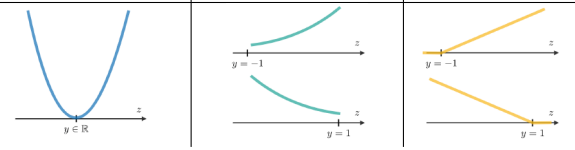
• **Forward Propagation:** The input data is passed through the network, layer by layer, with each neuron applying its weights and bias to the input and passing the result through the activation function. The final layer produces the output.

• **Backpropagation:** Backpropagation is an algorithm used in neural networks to adjust the internal weights and biases to minimize the error calculated by the loss function.

- **Regression Loss**: Mean Squared Error/Squared loss/ L2 loss, Mean Absolute Error/ L1 loss, Huber Loss
- **Classification Loss**: Binary Cross Entropy/log loss, Categorical Cross Entropy

The common loss functions are summed up in the table below:

Least squared error	Logistic loss	Hinge loss
$\frac{1}{2}(y - \hat{y})^2$	$\log(1 + \exp(-y\hat{y}))$	$\max(0, 1 - y\hat{y})$
Linear Regression	Logistic Regression	SVM

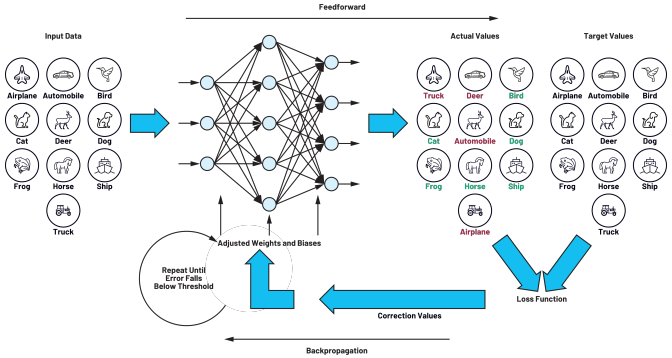


→ During training, the network uses a supervised learning method where the difference (error) between the network's predicted output and the known expected output is calculated. This error is then propagated back through the network via backpropagation to compute the gradient of the loss function with respect to each weight. An optimization algorithm, such as gradient descent, uses these gradients to update the weights and biases, reducing the error and improving the model's accuracy over time.

• **Training:** The process of forward propagation, loss calculation, and backpropagation is repeated over many iterations, allowing the network to learn from the data and improve its accuracy.

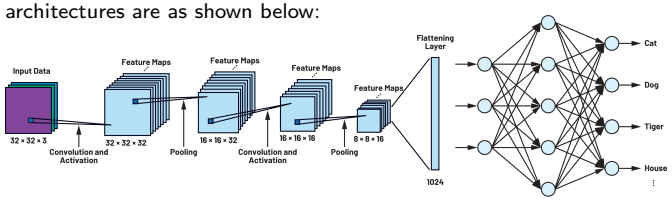
• To prevent **overfitting**, regularization can be applied by:

- Stopping training when validation performance drops
- **Dropout** - randomly drop some nodes during training to prevent over-reliance on a single node
- Embedding weight penalties into the objective function
- **Batch Normalization** - stabilizes learning by normalizing inputs to a layer



Convolutional Neural Network

CNN is a neural network architecture that is well-suited for image classification and object recognition tasks. The general CNN architectures are as shown below:

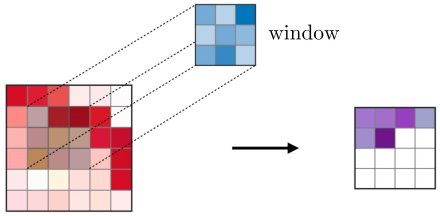


→ A convolutional neural network starts by taking an input image, represented as a matrix of pixel values

→ This input image is passed through **convolutional layers**. Here, a set of filters applies to the input image to detect features like edges, textures, and patterns. Each filter produces a feature map that highlights a specific aspect of the input image.

→ After each convolution, an **activation function** (like ReLU) is applied to introduce non-linearity, enabling the network to learn more complex patterns.

→ This produces **feature maps**. Different weights lead to different feature maps.



→ The feature maps are then passed through **pooling layers**, which downsample the spatial dimensions by taking the maximum or average value in small regions. This reduces the size of the feature maps and retains essential information, making the network more efficient and less sensitive to slight changes in the input.

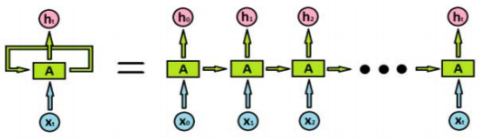
→ Again, the **feature maps** produced by the convolutional layer and pooling layer are then passed through multiple additional convolutional and pooling layers, each layer learning increasingly complex features of the input image.

→ Now, the output obtained from above is fed into a **fully connected layer** for classification, object detection, or other structural analyses. The final output of the network is a predicted class label or probability score for each class, depending on the task.

Question: Describe the difference between **batch normalization** and **layer normalization**.

Recurrent Neural Network

Recurrent Neural Networks (RNNs) are designed to process sequences of data such as time series data, voice, natural language, and other activities.



→ RNN memorize information from previous data with feedback loops inside it which helps to keep data information over time.

→ It has an arrow pointing to itself, indicating that the data inside block "A" will be recursively used. Once expanded, its structure is equivalent to a chain-like structure.

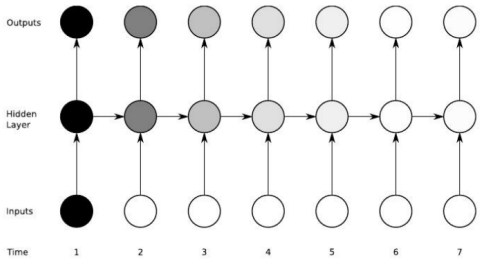
→ Learning to store information or data over long periods of time intervals via recurrent backpropagation takes a very long time. Hence, the gradient gradually vanishes as they propagate to earlier time steps. These downstream gradients relies on parameter (weight) sharing for efficiency, and repeatedly multiplying values greater than or less than 1 leads to:

- **Exploding gradients** - model instability and overflows
- **Vanishing gradients** - loss of learning ability

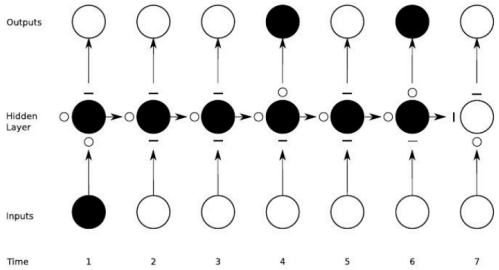
→ This can be solved using:

- **Gradient clipping** - cap the maximum value of gradients
- **ReLU** - its derivative prevents gradient shrinkage for $x > 0$
- **Gated cells** - regulate the flow of information

And, also for the non-convex problem, the RNN model training confuse between local minimum and global minimum. To overcome these problem, LSTM has been introduced as RNN languages modelling learning algorithm based on the feedforward architecture.



• Vanishing gradient problem for RNNs. The sensitivity increases as the network backpropagates through in time. The darker the shade, the greater the sensitivity.



• Preservation of gradient information by LSTM. The sensitivity of the output layer can be switched on and off.

→ LSTM memorize the information for the long period of time. The difference between RNN and LSTM are: RNN cell has only **one** $\tanh$ layer while LSTM cell has **four** layers: forget gate layer, store gate layer, new cell state layer, output layer, and previous cell state as shown in Figure below.

Time Series

Time Series is generally data which is collected over time and is dependent on it.

It is a random sequence $\{X_t\}$ of real values recorded at successive equally spaced points in time.

→ Not every data collected with respect to time represents a time series.

→ Methods of prediction & forecasting, time based data is Time Series Modeling

• **Examples** of time series: Stock Market Price, Passenger Count of airlines, Temperature over time, Monthly Sales Data, Quarterly/Annual Revenue, Hourly Weather Data/Wind Speed, IOT sensors in Industries and Smart Devices, Energy Forecasting

→ Difference between **Time Series** and **Regression**

• **Time Series** is time dependent. However the basic assumption of a linear regression model is that the observations are independent.

• Along with an increasing or decreasing trend, most Time Series have some form of seasonality trends

Note:

→ Predicting a time series using regression techniques is not a good approach.

→ Time series forecasting is the use of a model to predict future values based on previously observed values.

→ A stochastic process is defined as a collection of random variables $X = \{X_t : t \in T\}$ defined on a common probability space, taking values in a common set S (the state space), and indexed by a set T, often either N or $[0, \infty)$ and thought of as time (discrete or continuous respectively) (Oliver, 2009).

Time Series Statistical Models

A **time series model** specifies the joint distribution of the sequence $\{X_t\}$ of random variables; e.g.,

$$P(X_1 \leq x_1, \ldots, X_t \leq x_t) \text{ for all } t \text{ and } x_1, \ldots, x_t$$

Typically, a time series model can be described as

$$X_t = m_t + s_t + Y_t$$

where m_t: trend component; s_t: seasonal component; Y_t: Zero-mean error

Note: The following are some zero-mean models

→ **iid noise**: The simplest time series model is the one with no trend or seasonal component, and the observations X_ts are simply independent and identically distribution random variables with zero mean. Such a sequence of random variable $\{X_t\}$ is referred to as **iid noise**.

$$P(X_1 \leq x_1, \ldots, X_t \leq x_t) = \prod_t P(X_t \leq x_t) = \prod_t F(x_t)$$

where $F(\cdot)$ is the cdf of each X_t. Further $E(X_t) = 0$ for all t. We denote such sequence as $X_t \sim \text{IID}(0, \sigma^2)$. IID noise is not interesting for forecasting since $X_t | X_1, \ldots, X_{t-1} = X_t$.

→ **iid noise** example: A **binary (discrete) process** $\{X_t\}$ is a sequence of iid random variables X_ts with

$$P(X_t = 1) = 0.5, \quad P(X_t = -1) = 0.5$$

→ **Gaussian Noise** example:A **continues process**: Gaussian noise $\{X_t\}$ is a sequence of iid normal random variables with zero mean and σ^2 variance; i.e., $X_t \sim N(0, \sigma^2)$

→ **Random walk**: The random walk $\{S_t, t = 0, 1, 2, \ldots\}$ (starting at zero, $S_0 = 0$) is obtained by cumulatively summing (or "integrating") random variables; i.e., $S_0 = 0$ and
$S_t = X_1 + \cdots + X_t$, for $t = 1, 2, \ldots$
where $\{X_t\}$ is iid noise with zero mean and σ^2 variance. Note that by differencing, we can recover X_t; i.e.,

$$\nabla S_t = S_t - S_{t-1} = X_t$$

Further, we have

$$E(S_t) = E\left(\sum_t X_t\right) = \sum_t E(X_t) = \sum_i 0 = 0$$

$$\text{Var}(S_t) = \text{Var}\left(\sum_t X_t\right) = \sum_t \text{Var}(X_t) = t\sigma^2$$

→ **White Noise**: We say $\{X_t\}$ is a white noise; i.e.,
$X_t \sim \text{WN}(0, \sigma^2)$, if $\{X_t\}$ is uncorrelated, i.e., $\text{Cov}(X_{t_1}, X_{t_2}) = 0$ for any t_1 and t_2 with $\text{E}[X_t] = 0$ and $\text{Var}(X_t = \sigma^2)$.

Note: Every $\text{IID}(0, \sigma^2)$ sequence is $\text{WN}(0, \sigma^2)$ but not conversely.

• **Moving Average Smoother** This is an essentially non-parametric method for trend estimation. It takes averages of observations around t; i.e., it smooths the series. For example, let

$$X_t = \frac{1}{3}(W_{t-1} + W_t + W_{t+1})$$

which is a three-point moving average of the white noise series W_t.

→ **AR(1)** model (**Autoregression** of order 1): Let

$$X_t = 0.6X_{t-1} + W_t$$

where W_t is a white noise series. It represents a regression or prediction of the current value X_t of a time series as a function of the past two values of the series.

Stationary Process

Extracts characteristics from time-sequenced data, which may exhibit the following characteristics:

- **Stationarity** - statistical properties such as mean, variance, auto-correlation are constant over time, an autocovariance that does not depend on time, and no trend or seasonality
- **Non-Stationary** - There are 2 major reasons behind the non-stationary of a Time Series
  - Trend - varying mean over time (mean is not constant)
  - Seasonality - variations at specific time-frames (standard deviation is not constant)
- **Trend** - Trend is a general direction in which something is developing or changing.
- **Seasonality** - Any predictable change or pattern in a time series that recurs or repeats over a specific time period (calendar times) occurring at regular intervals less than a year
- **Cyclicality** - variations without a fixed time length, occurring in periods of greater or less than one year
- **Autocorrelation** - degree of linear similarity between current and lagged values

• CV must account for the time aspect, such as for each fold F_x:
- **Sliding Window** - train F_1, test F_2, then train F_2, test F_3
- **Forward Chain** - train F_1, test F_2, then train F_1, F_2, test F_3

• **Exponential Smoothing** - uses an exponentially decreasing weight to observations over time, and takes a moving average. The time t output is $s_t = \alpha x_t + (1 - \alpha)s_{t-1}$, where $0 < \alpha < 1$.

• **Double Exponential Smoothing** - applies a recursive exponential filter to capture trends within a time series

$$s_t = \alpha x_t + (1 - \alpha)(s_{t-1} + b_{t-1})$$
$$b_t = \beta(s_t - s_{t-1}) + (1 - \beta)b_{t-1}$$

Triple exponential smoothing adds a third variable γ that accounts for seasonality.

• **ARIMA** - models time series using three parameters (p, d, q):
- **Autoregressive** - the past p values affect the next value
- **Integrated** - values are replaced with the difference between current and previous values, using the difference degree d (0 for stationary data, and 1 for non-stationary)
- **Moving Average** - the number of lagged forecast errors and the size of the moving average window q

• **SARIMA** - models seasonality through four additional seasonality-specific parameters: P, D, Q, and the season length s

• **Prophet** - additive model that uses non-linear trends to account for multiple seasonalities such as yearly, weekly, and daily.
→ Robust to missing data and handles outliers well.
→ Can be represented as: $y(t) = g(t) + s(t) + h(t) + \epsilon(t)$, with four distinct components for the growth over time, seasonality, holiday effects, and error. This specification is similar to a generalized additive model.

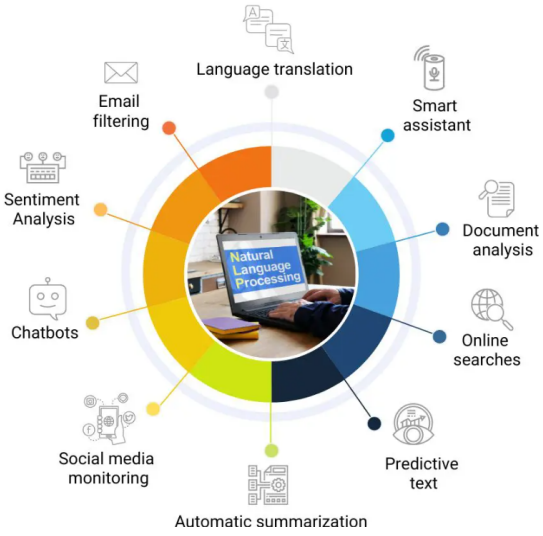
• **Generalized Additive Model** - combine predictive methods while preserving additivity across variables, in a form such as $y = \beta_0 + f_1(x_1) + \cdots + f_m(x_m)$, where functions can be non-linear.
→ GAMs also provide regularized and interpretable solutions for regression and classification problems.

Tutorial: **Complete Guide on Time Series Analysis in Python**

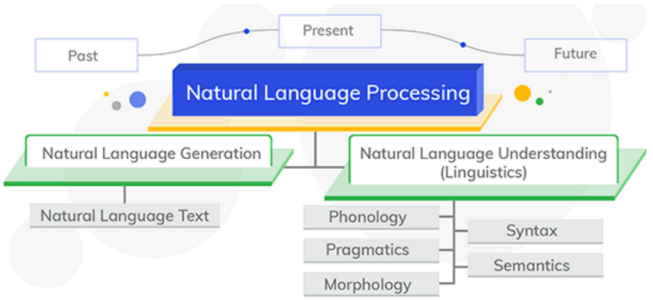
Natural Language Processing

NLP is the discipline of building machines that can manipulate human language — or data that resembles human language — in the way that it is written, spoken, and organized. It evolved from computational linguistics.

NLP Applications



Evolution of NLP



Challenges in NLP

• The 3 stages of an NLP pipeline are: Text Processing → Feature Extraction → Modeling.

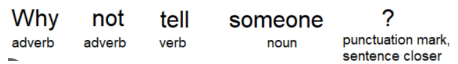
Text Processing

Take raw input text, clean it, normalize it, and convert it into a form that is suitable for feature extraction.
Libraries: nltk, spacy

- **Lower casing**
- Removing other stuff like: punctuations, tags, URLs, etc depends on the problem
- Convert **chat words** used in social media to a normal word
- Spelling correction using libraries like **TextBlob**
- **Stop words** - removes common and irrelevant words (*the, is*)
  Note: Do **not** remove stop words when using **POS Tagging** in text processing.
- **Tokenization** - splits text into individual words (tokens) and word fragments.
  • **Sentence-level** tokenization involves splitting a text into individual sentences.
  • **Word-level** tokenization involves splitting each sentence into individual words or tokens.
- **Lemmatization** - reduces words to its base form based on dictionary definition (*am, are, is* → *be*)
- **Stemming** - reduces words to its base form without context (*ended* → *end*)
- Language Detection

Advance Text Processing

- **POS Tagging**



- **Parse Tree**
- **Coreference Resolution**

Feature Extraction

→ Feature Extraction = Text Representation = Text Vectorization

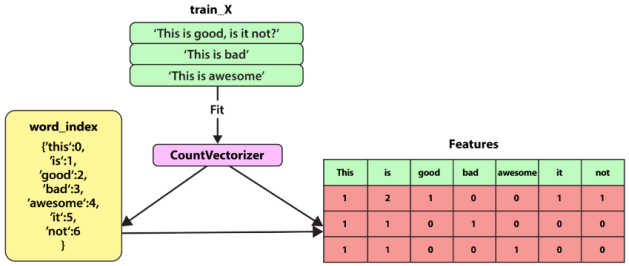
Common Terms:
• Corpus • Vocabulary • Document • Word

Documents	
Doc1	people watch campusx
Doc2	campusx watch campusx
Doc3	people write comment
Doc4	campusx write comment
Corpus	
people watch campusx campusx watch campusx people write comment campusx write comment	
Vocabulary	
people watch campusx write comment	

→ Most conventional machine learning techniques work on the features – generally numbers that describe a document in relation to the corpus that contains it – created by either Bag-of-Words, TF-IDF, or generic (custom) feature engineerings such as **document length**, **word polarity**, and metadata (for instance, if the text has associated **tags** or **scores**).

Note: Deep learning does **not** require to do **feature engineering**

– **Bag-of-words** - counts the number of times each word or n-gram (combination of n words) appears in a document.



Bag-of-Words (through the CountVectorizer method) encodes the total number of times a document uses each word in the associated corpus.

– **n-gram** - predicts the next term in a sequence of n terms based on Markov chains
→ **Markov Chain** - stochastic and memoryless process that predicts future events based only on the current state

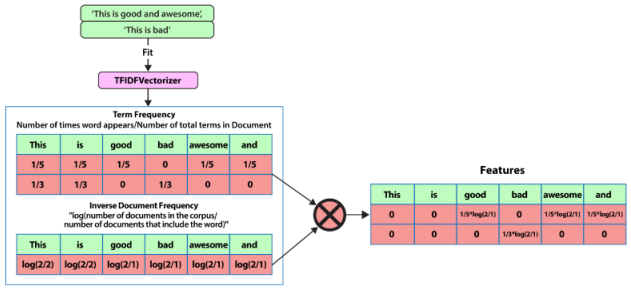
text = "The Margherita pizza is not bad taste"		
1-Gram	**2-Gram**	**3-Gram**
The	The Margherita	The Margherita pizza
Margherita	Margherita pizza	Margherita pizza is
pizza	pizza is	pizza is not
is	is not	is not bad
not	not bad	not bad taste
bad	bad taste	
taste		

– **tf-idf** - In contrast, with TF-IDF, we weight each word by its importance. To evaluate a word's significance, we consider two things:

1. **Term Frequency**: How important is the word in the document? TF(word in a document) = $\frac{\text{Number of occurrences of that word in document}}{\text{Number of words in document}}$
2. **Inverse Document Frequency**: How important is the word in the whole corpus (a collection of documents)? IDF(word in a corpus) = $\log(\frac{\text{number of documents in the corpus}}{\text{number of documents that include the word}})$

Note: A word is important if it occurs many times in a document. But that creates a problem. Words like "a" and "the" appear often. And as such, their TF score will always be high. We resolve this issue by using Inverse Document Frequency, which is high if the word is rare and low if the word is common across the corpus. The TF-IDF score of a term is the product of TF and IDF.

Cosine Similarity - measures similarity between vectors, calculated as $\cos(\theta) = \frac{A \cdot B}{||A|| ||B||}$, which ranges from 0 to 1



TF-IDF creates features for each document based on how often each word shows up in a document versus the entire corpus.

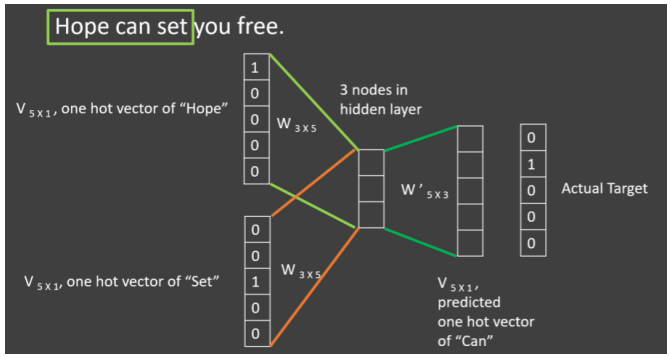
→ **CountVectorizer** - Bag of Words
→ **TfidfTransformer** - TF-IDF values
→ **TfidfVectorizer** - Bag of Words AND TF-IDF values

Word Embedding

Word embeddings are often based on neural network models in deep learning.
→ Based on CBOW, Skip gram: Word2vec, GloVe, fastText
– **Continuous bag-of-words** (CBOW) - predicts the word given its context
– **skip-gram** - predicts the context given a word

• **word2vec** - trains iteratively over a corpus of text to learn the association between the words, and preserve the **semantic** information as well as **contextual** meanings of words within a given corpus of text.
→ They are numerical representations of words and phrases allowing **similar words** to have **similar vector** representations.
→ It uses the cosine similarity metric to measure **semantic** similarity. If the cosine angle is one, it means that the words are overlapping. , such that $king - man + woman \approx queen$

Note: According to research **CBOW** is used when small dataset is available.



• **GloVe** (Global Vectors for Word Representation) - GloVe operates on the idea that words that frequently co-occur together, sharing similar contexts, tend to have related meanings. It builds a global co-occurrence matrix that captures the frequency of word co-occurrences within a context window across the entire corpus
→ Based on transformer architecture
• **BERT** - accounts for word order and trains on subwords, and unlike word2vec and GloVe, BERT outputs different vectors for different uses of words ($cell$ phone vs. blood $cell$)

Sentiment Analysis

Extracts the attitudes and emotions from text
• **Polarity** - measures positive, negative, or neutral opinions
– Valence shifters - capture amplifiers or negators such as '$really$ fun' or '$hardly$ fun'
• **Sentiment** - measures emotional states such as happy or sad
• **Subject-Object Identification** - classifies sentences as either subjective or objective

Topic Modelling

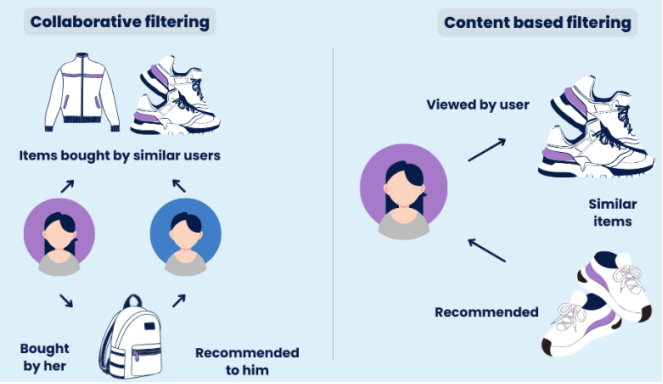
Captures the underlying themes that appear in documents
• **Latent Dirichlet Allocation** (LDA) - generates k topics by first assigning each word to a random topic, then iteratively updating assignments based on parameters α, the mix of topics per document, and β, the distribution of words per topic
• **Latent Semantic Analysis** (LSA) - identifies patterns using tf-idf scores and reduces data to k dimensions through SVD

NLP Tutorial
Duplicate Question Pairs - Quora Questions Pairs: NLP Pipeline

Recommender System

The Recommender System utilizes machine learning and data analysis to provide personalized suggestions to users.
→ It operates by collecting and analyzing user behavior, user preferences, and historical user-item interactions.

- Two main types of recommender system:
  - **Collaborative Filtering** - recommends what similar users like
  - **Content Filtering** - recommends similar items



Collaborative filtering is more common and includes methods such as:
- **Memory-based Approaches** - finds neighborhoods by using **rating data** to compute user and item similarity, measured using correlation or cosine similarity
  - **User-User** - similar users also liked...
    - Leads to more diverse recommendations, as opposed to just recommending popular items
    - Suffers from sparsity, as the number of users who rate items is often low
  - **Item-Item** - similar users who liked this item also liked...
    - Efficient when there are more users than items, since the item neighborhoods update less frequently than users
    - Similarity between items is often more reliable than similarity between users
- **Model-based Approaches** - predict ratings of unrated items, through methods such as Bayesian networks, SVD, and clustering. Handles sparse data better than memory-based approaches.
  - **Matrix Factorization** - decomposes the user-item rating matrix into two lower-dimensional matrices representing the users and items, each with k latent factors

→ Recommender systems can also be combined through ensemble methods to improve performance.

References

[1] Rahul Beakta. "Big data and hadoop: A review paper". In: *International Journal of Computer Science & Information Technology* 2.2 (2015), pp. 13–15.

[2] M. Sundermeyer, H. Ney, and R. Schlüter. "From Feedforward to Recurrent LSTM Neural Networks for Language Modeling". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23.3 (Mar. 2015), pp. 517–529. ISSN: 2329-9290. DOI: 10.1109/TASLP.2015.2400218.

[3] Varsha B Bobade. "Survey paper on big data and Hadoop". In: *Int. Res. J. Eng. Technol* 3.1 (2016), pp. 861–863.

[4] D. Dong, Z. Sheng, and T. Yang. "Wind Power Prediction Based on Recurrent Neural Network with Long Short-Term Memory Units". In: *2018 International Conference on Renewable Energy and Power Engineering (REPE)*. Nov. 2018, pp. 34–38. DOI: 10.1109/REPE.2018.8657666.

[5] Analog Devices. *Training Convolutional Neural Networks: What is Machine Learning? Part 2*. Analog Dialogue. URL: https://www.analog.com/en/analog-dialogue/articles/training-convolutional-neural-networks-what-is-machine-learning-part-2.html.

[6] deeplearning.ai. *Natural Language Processing Resources*. deeplearning.ai. URL: https://www.deeplearning.ai/resources/natural-language-processing/.

[7] Edureka. *MapReduce Tutorial*. Edureka. URL: https://www.edureka.co/blog/mapreduce-tutorial/.

[8] Edureka. *Top 50 Hadoop Interview Questions (2016)*. Edureka. URL: https://www.edureka.co/blog/interview-questions/top-50-hadoop-interview-questions-2016/.

[9] Nilay Chauhan. *Getting Started with NLP Pipelines*. Kaggle. URL: https://www.kaggle.com/code/nilaychauhan/getting-started-with-nlp-pipelines.