Machine Learning Cheatsheet

© 2024 Robins Yadav - magic starts here

My github: https://github.com/robinyUArizona

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 learning 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**.

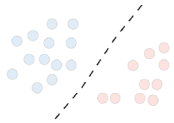
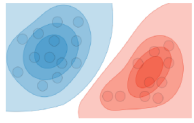
Type of prediction

	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**: $P(Y = y|X = x)$

Type of models

→ **Discriminative Model**: It focuses on predicting the labels of the data. A discriminative machine learning trains a model which is done by learning parameters that maximize the **conditional probability** $P(Y|X)$
→ **Generative Model**: It focuses on explaining how the data was generated. A generative model learns parameters by maximizing the **joint probability** of $P(X, Y)$.

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 deduce $P(y\|x)$ using Baye's rule
Cannot generate new data	Can be used to generate new data
Specifically meant for classification tasks	Typically, their purpose is not classification
Logistic Regression, Random Forests, SVM, Neural Networks, Decision Tree, kNN	Hidden Markov Models, Naive Bayes, Gaussian Mixture Models, Gaussian Discriminant Analysis, LDAm Bayesian Networks

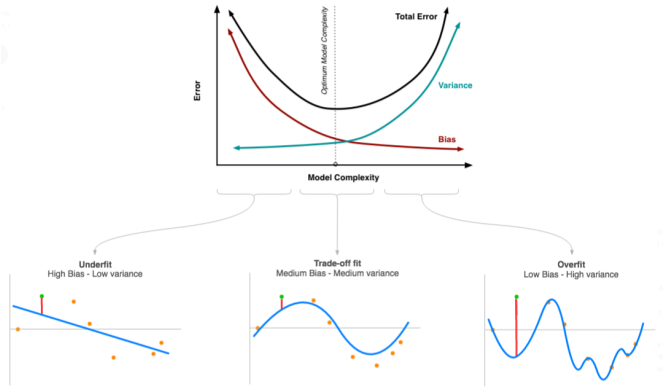
• **Unsupervised** learning methods learn to find the inherent structure or hidden patterns from **unlabeled data** X $(x^{(1)}, ..., x^{(m)})$.

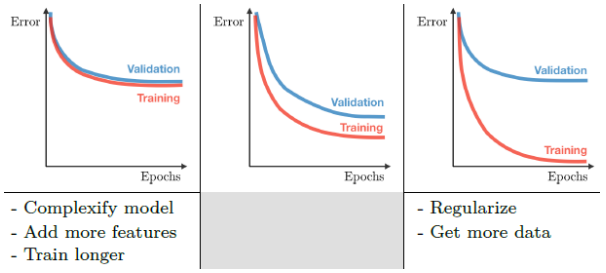
Bias-Variance trade-off, Underfitting, Overfitting

→ Expected test error is the error we expect from predicting new, unobserved data points.
In supervised learning, the **prediction (expected) error** e is composed of the **bias**, the **variance** and the **irreducible** part.

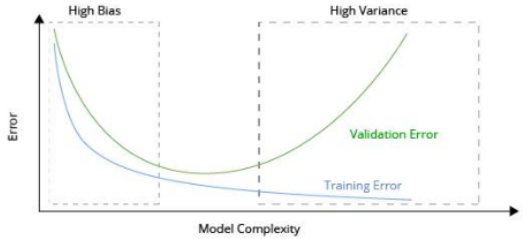
$$\text{Error}(x) = \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2 + \mathbb{E}\left[\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right]^2 + \sigma_e^2$$

→ **Bias** refers to **erroneous assumptions** made by the **model** about the data to make the **target function** easier to learn.
Mathematically, how much predicted values differ from true values?
→ **Variance** is the **error**, amount that the prediction (estimate of the target function) will change if **different training data sets** were used. It measures how scattered (inconsistent) are the predicted values from the correct value due to **different training data (or possibly with different random seeds)** sets. It is also known as **Variance Error** or **Error due to Variance**.
Note: As the **complexity of the model rises**, 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
→ **Dimension reduction** and **feature selection**
→ Mixture models (probabilistic models) and **ensemble learning**.





→ **Underfitting** or **High bias** means that the model is not able to capture or learn the trend or pattern in data.
→ **Overfitting** or **High variance** means that the model fits the available data but does not generalize well 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**.

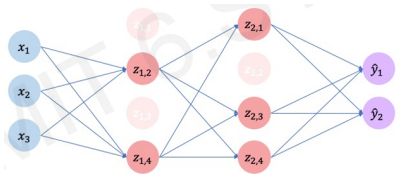
• Training loss vs. Validation loss:

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.

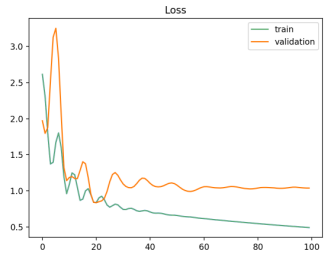
• **How would you identify if your model is overfitting?** By analyzing the **learning curves**, you should be able to spot whether the model is underfitting or overfitting. The y-axis is some metric of learning (ex:, classification accuracy) and the x-axis is experience (time or No. of iteration).

• **Regularization - Dropout** During training, randomly set some activations to 0. This forces network to not rely on any one node.



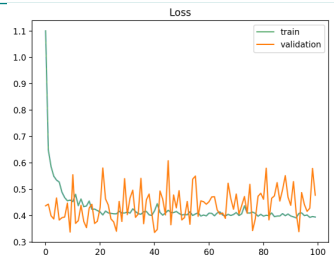
Unrepresentative Training Dataset

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

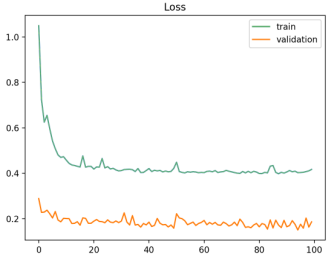


The train and validation curves are improving, but there's a big gap between them, which means they operate like datasets from different distributions.

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.

Model Evaluation

Classification Problems

Confusion Matrix

- The data gives us **outcomes** ("truth") $(y|Y)$
- The model makes **decisions** $(d|D)$ (saving $\hat{y}$ scores)

Then, we compare decisions (d) to outcomes (y)

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**

Decision (Prediction)	Truth (Actual)	
	Null Hypothesis (**H0**)	Alternative Hypothesis (**H1**)
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

Decision (Prediction)	Truth (Actual)		
	Not Guilty (**H0**) 1	Guilty (**H1**) 0	
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)

(1) **Accuracy:** $\boxed{\frac{TP+TN}{TP+TN+FP+FN}}$ → Ratio of **correct predictions** over **total predictions**.
Estimate of $\boxed{P[D=Y]}$, probability of decision is equal to outcome.

(2) **Recall** or **Sensitivity** or **True positive rate**: $\boxed{\frac{TP}{TP+FN}}$.
Completness of model. → Out of **total actual positive (1) values**, how often the classifier is correct. **Probability**: $\boxed{P[D=1|Y=1]}$
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** : $\boxed{\frac{TP}{TP+FP}}$ Exactness of model. → Out of **total predicted positive (1) values**, how often classifier is correct.
Probability: $\boxed{P[Y=1|D=1]}$, 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 cancelled**" +ve (1) class is isCancelled → Optimize for precision or, specificity because false negatives (**FN** isCancelled labeled as "not cancelled" 0) are more acceptable than false positive (**FP** isnotCancelled labeled as "cancelled" 1).

(4) **F1-Score** $= \boxed{2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}}$ → False positive (FP) and False negative (FN) are equally important.

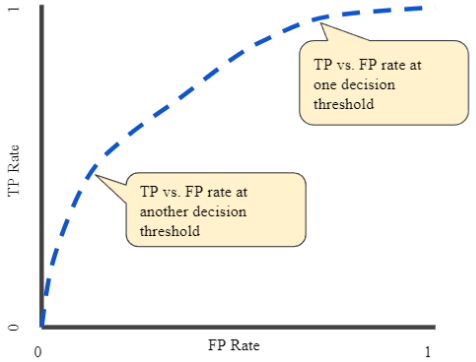
(5) **False Positive Rate**: $\boxed{\frac{FP}{TN+FP}}$ — Fraction of **negatives** wrongly classified positive. **Probability**: $\boxed{P[D=1|Y=0]}$

(6) **False Negative Rate**: $\boxed{\frac{FN}{TP+FN}}$ = 1-Recall — Fraction of **positives** wrongly classified negative. **Probability**: $\boxed{P[D=0|Y=1]}$

(7) **Specificity**: $\boxed{\frac{TN}{TN+FP}}$ = 1-FPR — Fraction of **negatives** rightly classified negative. **Probability**: $\boxed{P[D=0|Y=0]}$

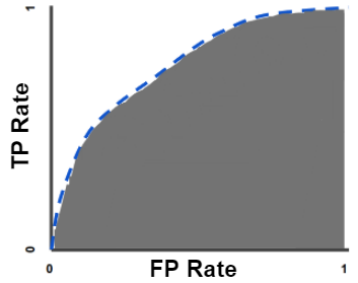
(8) "**Fradulent transaction detector**", $\boxed{\text{FPR} = \frac{FP}{FP+TN}}$ → probability of falsely rejecting "Null Hypothesis" H_0

(9) **ROC**-curve: **What FPR must you tolerate for a certain TPR?** An ROC curve plots TPR vs. FPR at different classification thresholds α.
→ Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.



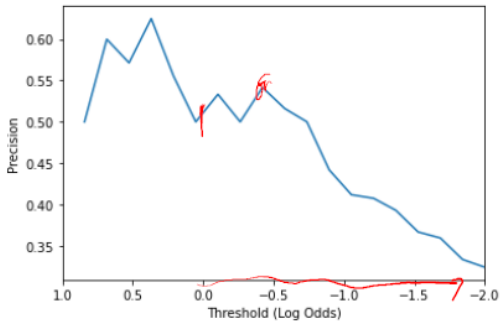
Note: We can think of the plot as the fraction of correct predictions for the positive class (y-axis) versus the fraction of errors for the negative class (x-axis).

(10) AUC: Area Under the ROC Curve: 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)



How to choose threshold for the logistic regression? The choice of a threshold depends on the importance of TPR and FPR classification problem. For **example**: Suppose you are building a model to **predict customer churn**. **False negatives** (not identifying customers who will churn) might lead to loss of revenue, making TPR crucial. In contrast, falsely predicting churn (**false positives**) could lead to unnecessary retention efforts, making FPR important. 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.

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

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

The proportion of explained y-variability. Negative R^2 means the model is worse than just predicting the mean. R^2 is not valid for nonlinear models as $\text{SS}_{\text{residual}} + \text{SS}_{\text{error}} \neq \text{SST}$.

7. **Adjusted** R^2 :

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

which changes only when predictors (features) affect R^2 above what would be expected by chance

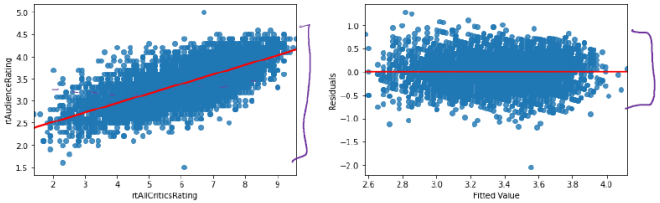
Variance, R^2 and the Sum of Squares

The **total sum of squares**: $\text{SS}_{total} = \sum_i (y_i - \bar{y})^2$
This scales with variance: $\text{var}(Y) = \frac{1}{n}\sum_i (y_i - \bar{y})^2$
The **regression** sum of squares: $\text{SS}_{reg} = \sum_i (\hat{y}_i - \bar{y})^2$
, → nVar(predictions)
The **residual** sum of squares (squared errro):
$\text{SS}_{resid} = \sum_i (y_i - \hat{y}_i)^2$, → nVar($\epsilon$)
Note: $\bar{\epsilon} = 0, E[\hat{y}] = \bar{y}$

$$SS_{total} = SS_{reg} + SS_{resid}$$

$$R^2 = 1 - \frac{SS_{resid}}{SS_{total}} = \frac{SS_{reg}}{SS_{total}} = \frac{nVar(Preds)}{nVar(Y)} = \frac{Var(Preds)}{Var(Y)}$$

The **fraction of variance explained!**

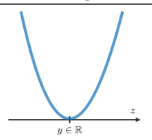
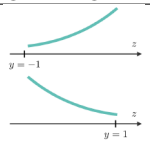
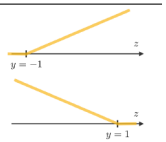


→ The variance in the outcome variable decomposes into regression variance and residual variance.
→ R^2 measures the fraction of total variance explained by the regression.
→ R^2 % of dependent variable variance explained using independent variable(s)

Optimization

Almost every machine learning method has an optimization algorithm at its core.
→ **Hypothesis :** The hypothesis is noted h_θ and is the model that we choose. For a given input data $x^{(i)}$ the model prediction output is $h_\theta(x^{(i)})$.

→ **Loss function :** $L : (z, y) \in R \times Y \longmapsto L(z, y) \in R$ that takes as inputs the **predicted value** z corresponding to the **real data value** y and outputs how different they are. The **loss function** is the function that computes the **distance** or **difference** between the current output z of the algorithm and the expected output y.
The common loss functions are summed up in the table below:

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

→ **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:

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

Convex & Non-convex

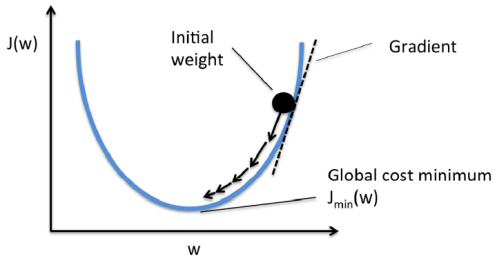
A **convex** function is one where a line drawn between any two points on the graph lies on or above the graph. It has **one minimum**. A **non-convex** function is one where a line drawn between any two points on the graph may intersect other points on the graph. It characterized as "wavy"
→ When a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

Gradient Descent

Gradient Descent is used to **find the coefficients** of f that **minimizes a cost function** (for example MSE, SSR).
→ **Time Complexity**: $O(kn^2)$ → n is no. of data points.
Procedure:

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

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



Tips :

- Change **learning rate** α ("size of jump" at each iteration)
- Plot *Cost* vs. *Time* to assess learning rate performance.
- Rescaling the input variables
- Reduce passes through training set with SGD
- Average over 10 or more updated to observe the learning trend while using SGD

Batch Gradient Descent does summing/average of the cost over all the observations.
Stochastic Gradient Descent apply the procedure of parameter updating for each observation.
→ **Time Complexity**: $O(km^2)$ → m is the sample of data selected randomly from the entire data of size n

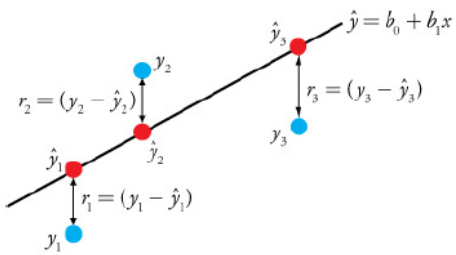
Ordinary Least Squares

Least Squares Regression
We fit linear models:

$$\boxed{\hat{y} = \beta_0 + \sum_j \beta_j x_j}$$

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**:

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



Goal: least-squares solution to : $\boxed{X\vec{\beta} = \hat{y}}$
Solution: solve the normal equations:

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

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

General Optimization:

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

Augmented Loss
We can add more things to the loss function

- Penalize model complexity
- Penalize "strong" beliefs
  - Requires predictive utility to overcome them

→ Least squares generalizes into **minimizing loss functions**.
→ This is the heart of machine learning, particularly supervised learning.

Maximum Likelihood Estimation

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

Log Likelihood

Logistic Regression:

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

The model computes **probability of yes**.

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

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

- $\hat{y_i}$ 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 $y_i = 0$ — multiplicative if

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

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

This is weird:

$$P(\mathbf{y}, \mathbf{x}|\vec{\beta}) \propto P(\mathbf{y}|\mathbf{x}, \vec{\beta})$$
$$P(\mathbf{y}, \mathbf{x}|\vec{\beta}) = P(\mathbf{y}|\mathbf{x}, \vec{\beta})P(\mathbf{x}|\vec{\beta})$$

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.

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

Maximum Likelihood Estimator

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

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

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

Likelihood and Posterior

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

→ Logistic function is trained by **maximizing the log likelihood** of the training data given the model

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!

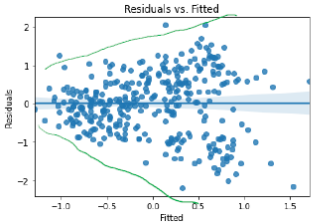
→ Linear models are remarkably capable of transforming many non-linear problems into linear.

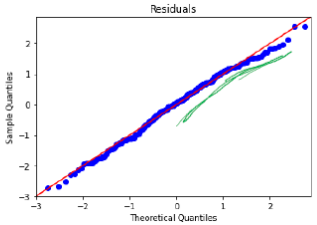
Linear Regression

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

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

Here, n is total no. of observation, 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



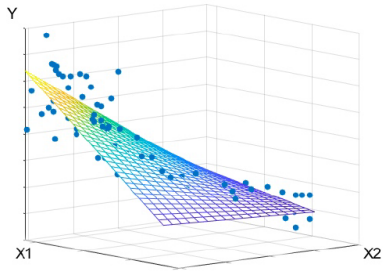


Variance Inflation Factor : Measures the severity if multicollinearity → $\boxed{\frac{1}{1 - R_i^2}}$, where R_i^2 is found by regressing X_i aganist all other variables (a common VIF cutoff is 10)

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.

Data preparation:
- Transform data for linear relationship (ex: log transform for exponential relationship)
- Remove noise such as outliers
- Rescale inputs using standardization or normalization

Advantages:
+ Good regression baseline considering simplicity
+ Lasso/Ridge can be used to avoid overfitting
+ Lasso/Ridge permit feature selection in case of collinearity

Usecase examples:
- Product sales prediction according to prices or promotions
- Call-center waiting-time prediction according to the number of complaints and the number of working agents

Logistic Regression

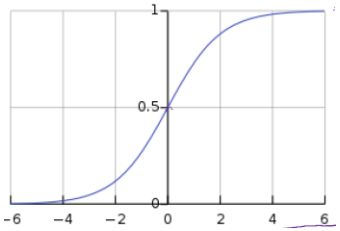
Log-Odds and Logistics

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):

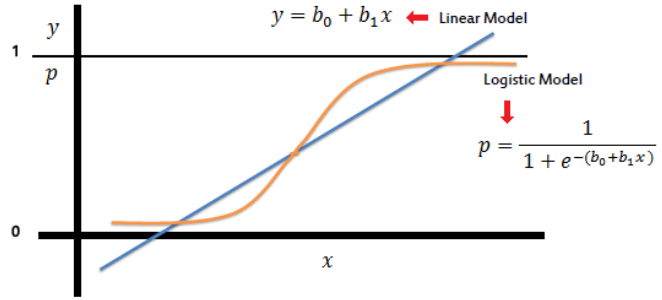
$$\hat{y_i} = g^{-1}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} \cdots + \beta_p x_{ip})$$

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

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. It is the go-to for **binary classification**.

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

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



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

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

→ predict value close to 1 for default class and close to 0 for the other 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:
Note : Coefficients are linearly related to odds, such that a one unit increase in x_1 affects odds by e^{β_1}.
→ **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

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

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

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

Data preparation:
- Probability transformation to binary for classification
- Remove noise such as outliers

Advantages:
+ Good classification baseline considering simplicity
+ Possibility to change cutoff for precision/recall tradeoff
+ Robust to noise/overfitting with L1/L2 regularization
+ Probability output can be used for ranking

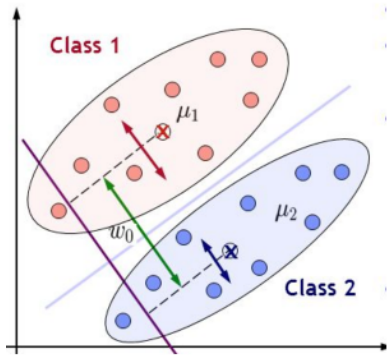
Usecase examples:
- Customer scoring with probability of purchase
- Classification of loan defaults according to profile

How to choose threshold for the logistic regression? The choice of a threshold depends on the importance of TPR and FPR classification problem. For **example**, if your classifier will decide which criminal suspects will receive a death sentence, false positives are very bad (**innocents will be killed!**). Thus you would choose a threshold that yields a low FPR while keeping a reasonable TPR (so you actually catch some true criminals). 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.

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**:

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

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:
- Review and modify univariate distributions to be Gaussian
- Standardize data to $\mu = 0$, $\sigma = 1$ to have same variance
- Remove noise such as outliers

Advantages:
+ Can be used for dimensionality reduction by keeping the latent variables as new variables

Usecase example:
- Prediction of customer churn

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 d **assuming that the features of each data point are all independent**
Representation: The representation is based on Bayes Theorem:

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

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

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

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

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

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

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

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

Data preparation:
- Change numerical inputs to categorical (binning) or near Gaussian inputs (remove outliers, log & boxcox transform)
- Other distributions can be used instead of Gaussian
- Log-transform of the probabilities can avoid overflow
- Probabilities can be updated as data becomes available

Advantages:
+ Fast because of the calculations
+ If the naive assumptions works can converge quicker than other models. Can be used on smaller training data.
+ Good for few categories variables

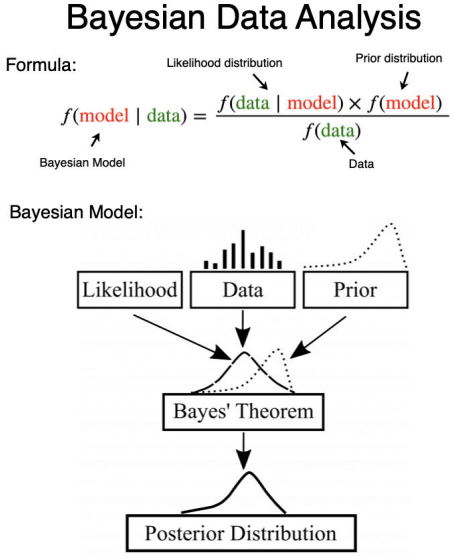
Usecase examples:
- Article classification using binary word presence
- Email spam detection using a similar technique

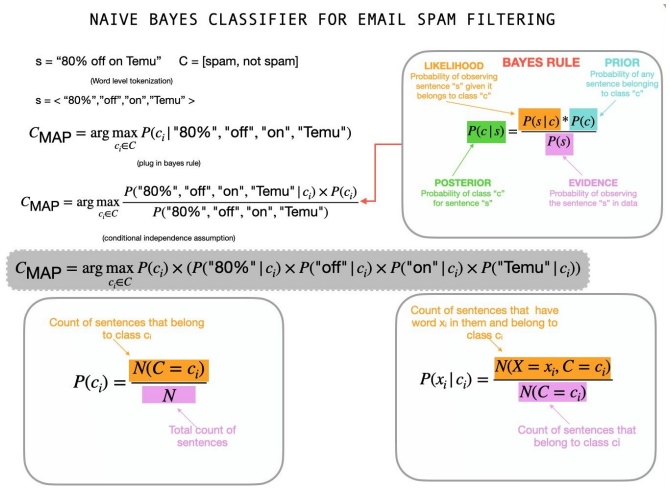
Likelihood and Posterior

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

- $P(\theta)$ 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



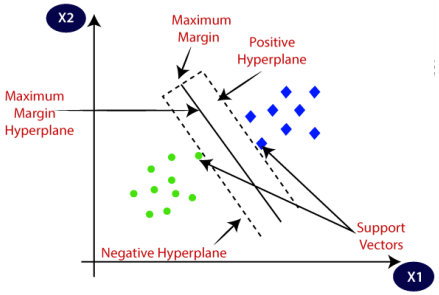


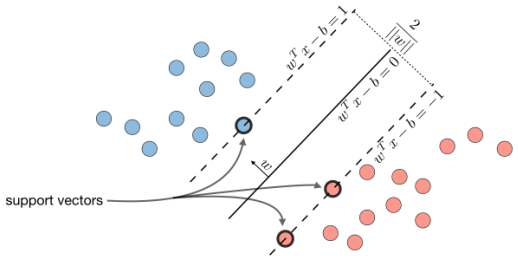
Support Vector Machines

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:

$$L(z, y) = [1 - yz]_+ = \max(0,\ 1 - yz)$$

→ **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**.

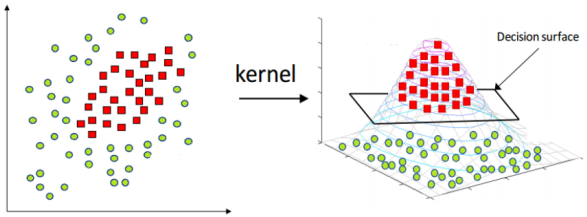
$$\lambda\|\vec{w}\|^2 + \left[\frac{1}{n}\sum_{i=1}^{n} \max\left(0,\ 1 - y_i(\vec{w} \cdot \vec{x_i} - b)\right)\right]$$

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:** $K(x, x_i) = e^{-}\gamma \sum(x - x_i)^2$

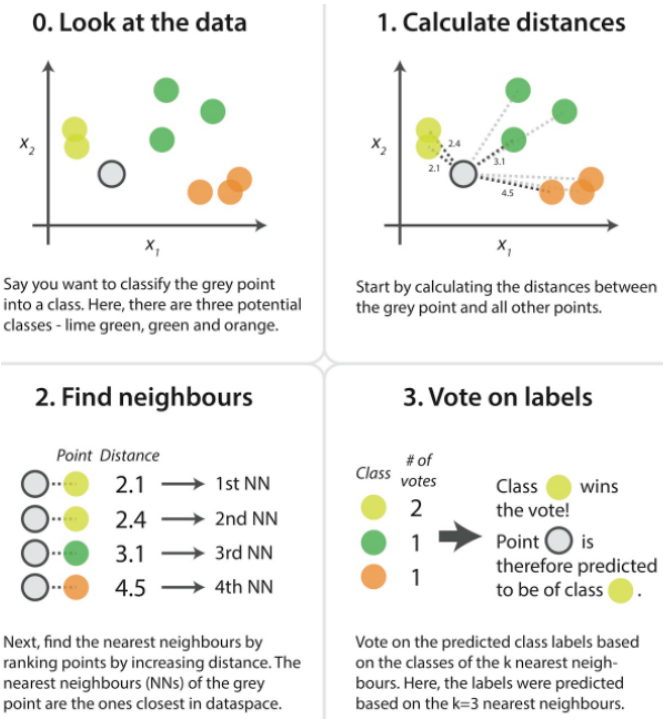
Data preparation:
- SVM assumes numeric inputs, may require dummy transformation of categorical features

Advantages:
+ Allow nonlinear separation with nonlinear Kernels
+ Works good in high dimensional space
+ Robust to multicollinearity and overfitting

Usecase examples:
- Face detection from images
- Target Audience Classification from tweets

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.

Time complexity: The distance calculation step requires quadratic time complexity, and the sorting of the calculated distances requires an $O(N \log N)$ time. Total process is an $O(N^3 \log N)$

Space complexity: Since it stores all the pairwise distances and is sorted in memory on a machine, memory is also the problem. Usually, local machines will crash, if we have very large datasets.

Data preparation:
- Rescale inputs using standardization or normalization
- Address missing data for distance calculations
- Dimensionality reduction or feature selection for COD

Advantages:
+ Effective if the training data is large
+ No learning phase
+ Robust to noisy data, no need to filter outliers

Usecase examples:
- Recommending products based on similar customers
- Anomaly detection in customer behavior

Classification and Regression Trees (CART)

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

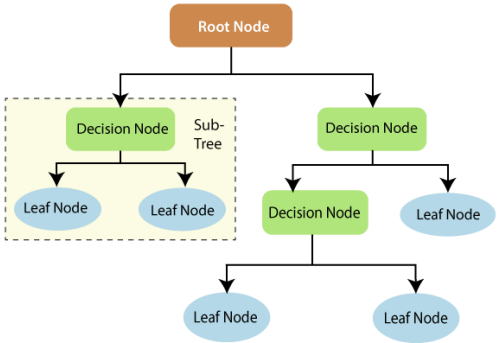
• CART for regression **minimizes SSE** by splitting data into sub-regions and predicting the average value at leaf nodes. The complexity parameter cp only keeps splits that reduce loss by at least cp (small $cp \rightarrow$ deep tree).

• 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.

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

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.

A decision tree is made up of nodes. Each node represents a question about the data. And the branches from each node represents the possible answers.



Root Node: It is the very first node, or we can call it as a parent node. It denotes the whole population and gets split into two or more Decision nodes based on the feature value.
Decision Node: Decision nodes are used to make any **decisions** and have **multiple** branches
Leaf Node: Leaf nodes are the **output** of those **decisions**.
Sub-Tree: A branch is a subdivision of a complete tree.

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

Procedure:

1. Calculate entropy of the **outcome** classes (c)
$$\boxed{E(T) = \sum_{i=1}^{c} -p_i \log_2 p_i}$$
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.
$$\boxed{\text{Gain}(T, X) = \text{Entropy}(T) - \text{Entropy}(T, X)}$$
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

Advantages:
→ Can take any type of variables and do not require any data prepraration
→ Simple to understand, interpret, visualize
→ Non-linear parameters don't effect its performance

Disadvantages:
→ Overfitting (High variance) occurs, when noise data
→ DT can be unstable (use bagging or boosting) because of small variation in data

Note: The most common **Stopping Criterion** for splitting is a minimum of **training observations per node**.

Ensemble Algorithms

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

Bagging	Boosting
Random Forest	AdaBoost
	Gradient Boosting
	XGBoost

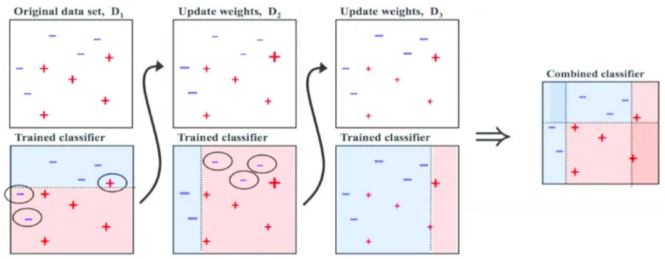
• **Bootstrapping** is drawing random **sub-samples** (sampling **with replacement**) from a large **sample** (available data) to estimate quantity (parameters) of a unknown population by **averaging the estimates** from these **sub-samples**.
• **Bagging**: It uses bootstrap technique, and can **reduce the variance** of high-variance models..

How bagging works? 1. Bootstrapping → 2. Parallel training → 3. Aggregation. Finally, **depending on the type of task** — regression or classification, for example—the average or majority of those predictions yield a more accurate estimate.

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.

• **Boosting**: The idea of boosting methods is to train **weak learners sequentially**, each trying to correct its predecessor.



AdaBoost

• Uses the **same training samples** at each stage
• "Weakness" = Misclassified data points
• Increase the weight of misclassified data points

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.

Gradient Boosting

• Uses the **different training samples** at each stage
• "Weakness" = Residuals or Erros
• A loss function to be optimized, additive model to add weak learners

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.

XGBoost

• Optimized and scalable implementation of gradient boosting.
→ **Execution speed**: Parallelization (It will use all cores of CPU), Cache optimization, Out of memory (Data size bigger than memory)
→ **Model performance:** Adds regularization (prevent overfitting), Auto running (Uses "max depth" to control the growth of trees), Missing values treatment, Efficient handling of sparse data,

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:

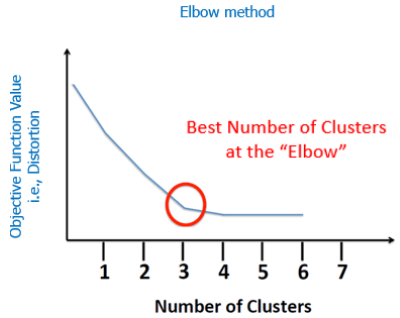
1. Partition-based methods
   (a) K-means clustering
   (b) Fuzzy C-Means
2. Hierarchical methods
   (a) Agglomerative Clustering
   (b) Divisive Clustering
3. Density-based methods
   (a) Density-Based methods (DBSCAN)

K-means clustering

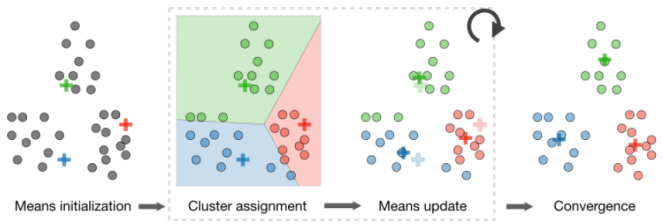
The objective of K-means clustering is to **minimize** total **intra-cluster** or, the **squared error function.**

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

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

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

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

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

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

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

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: $$Z = \frac{X - mean}{SD}$$
2. Calculate **covariance-matrix** of the standarized data
   $$V = \text{cov}(Z^T)$$
3. Find **eigen-values** and **eigen-vectors** from the covariance-matrix
   $$\text{values, vectors} = \text{eig}(V)$$
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 → $$Z_{\text{new}} = \text{vectors}^T \cdot Z^T$$

Association Rule Mining

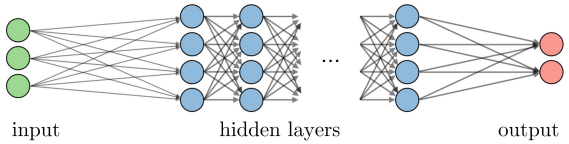
"Market Basket Analysis" → It uses Machine Learning models to analyze data for patterns or, co-occurence in a database.

Graphical Modelling and Network Analysis

"Bayesian Networks"

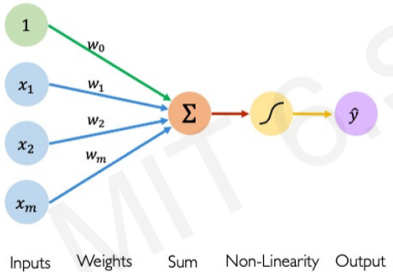
Neural Network

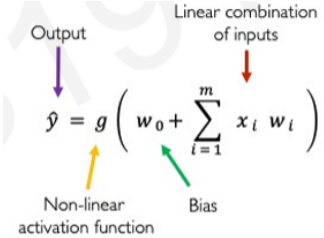
Deep Learning Tutorial by CampusX
→ Feeds inputs through different hidden layers and relies on weights and nonlinear functions (activation functions, convolution, or pooling) to reach an output



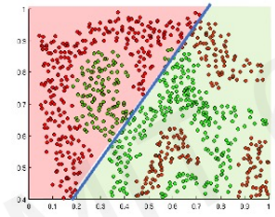
• **Perceptron** - the foundation of a neural network, and it is a single-layer neural network that multiplies inputs by weights, adds bias, and feeds the result to an activation function

Note: Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a **Linear Binary Classifier**.
An Artificial Neuron is a basic building block of a neural network.

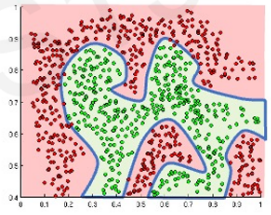




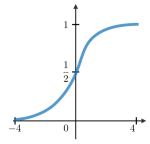
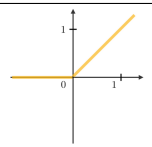
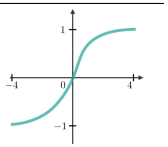
→ **Weights**: are the real values that are attached with each input/feature and they convey the importance of that corresponding feature in predicting the final output.
→ **Bias**: is used for shifting the activation function towards left or right.
→ **Summation Function**: used to bind the weights and inputs together and calculate their sum.
→ **Activation Function**: decides whether a neuron should be activated or not. The activation function 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
$\frac{1}{1+e^{-z}}$	$\max(0, z)$	$\frac{e^z-e^{-z}}{e^z+e^{-z}}$
		

• **Neural Network** - a multi-layer **perceptron**
→ **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.

→ **Loss Function** - The loss function is the function that computes the distance or difference between the **predicted** output $\hat{y}$ of the algorithm and the **expected** output y. → The loss function is used to optimize the model by minimizing the loss.
- **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

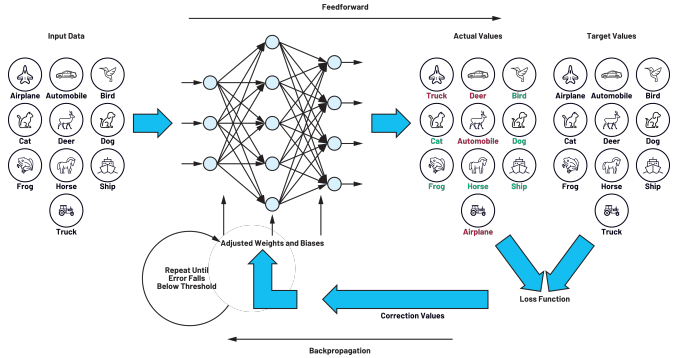
Gradient Descent - minimizes the average loss by moving iteratively in the direction of steepest descent, controlled by the learning rate γ (step size). Note, γ can be updated adaptively for better performance. For neural networks, finding the best set of weights involves:

1. Initialize weights W randomly with near-zero values
2. Loop until convergence:
   - Calculate the average network loss $J(W)$
   - **Backpropagation** - iterate backwards from the last layer, computing the gradient $\frac{\partial J(W)}{\partial W}$ and updating the weight $W \leftarrow W - \gamma \frac{\partial J(W)}{\partial W}$
3. Return the minimum loss weight matrix W

• 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

Stochastic Gradient Descent - only uses a single point to compute gradients, leading to smoother convergence and faster compute speeds. Alternatively, mini-batch gradient descent trains on small subsets of the data, striking a balance between the approaches.

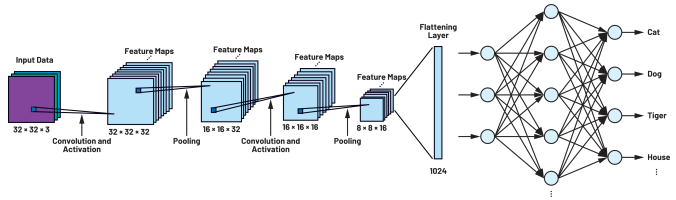
Backpropagation :
Artificial neural networks (ANNs) and deep neural networks use **backpropagation** as a learning algorithm to compute a gradient descent, which is an optimization algorithm that guides the user to the maximum or minimum of a function.



The principle of the backpropagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state.

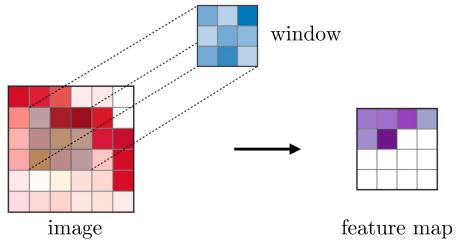
Convolutional Neural Network

CNNs are a type of deep learning neural network architecture that is particularly well suited to image classification and object recognition tasks. The general CNN architectures is as shown below:



A convolutional neural network starts by taking an input image, which is then transformed into a feature map through a series of convolutional and pooling layers.

The **convolutional layer** applies a set of filters to the input image (by applying weights, bias, and an activation function), each filter producing a feature map that highlights a specific aspect of the input image. Different weights lead to different feature maps.



The **pooling layer** then downsamples the feature map to reduce its size, while retaining the most important information even if they have shifted slightly.

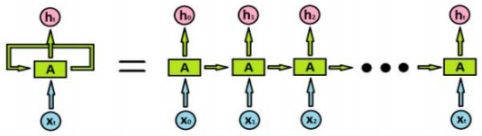
Again, the pooled feature maps produced by the convolutional layer and pooling layer are then passed through multiple additional convolutional and pooling layers, each layer learning increasingly complex features of the input image.

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.

Recurrent Neural Network

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

• RNN works on the principle of saving the output using hidden states of a particular layer and feeding this back to the input in order to predict the output of the layer.



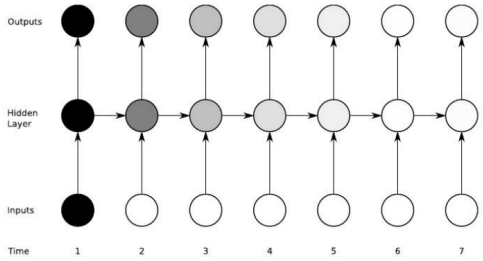
→ RNN memorize information from previous data with feedback loops inside it which helps to keep data information over time.
→ It has an arrow pointing to itself, indicating that the data inside block "A" will be recursively used. Once expanded, its structure is equivalent to a chain-like structure.
→ 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

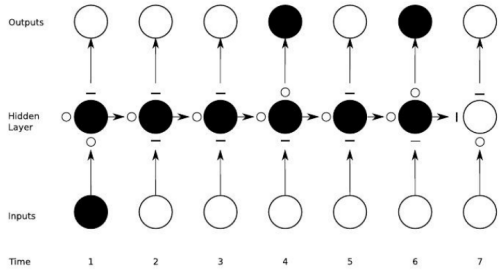
→ This can be solved using:

- **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.

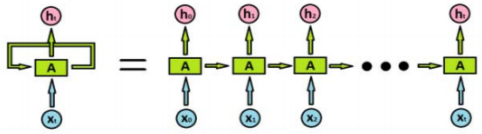


• Vanishing gradient problem for RNNs. The sensitivity increases as the network backpropagates through in time. The darker the shade, the greater the sensitivity.

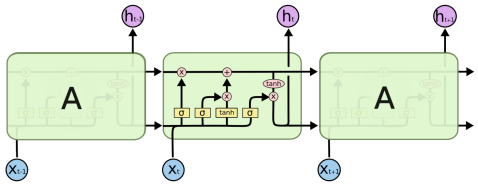


• Preservation of gradient information by LSTM. The sensitivity of the output layer can be switched on and off.

LSTM is based on RNN, therefore, the basic structure of RNN is explained first and then LSTM structure is explained referencing RNN.



LSTM memorize the information for the long period of time, which is important in many applications such as time prediction of the High frequency (HF) spectrum. The basic structure of the RNN and LSTM are similar as shown respectively below.



→ 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.

→ The forget layer is responsible for deciding what information to retain from the previous cell state, and what information is to be forgotten or removed

$$f_t = \sigma(W_t.[h_{t-1}, x_t] + b_f)$$

→ The store gate layer has an input gate using which we calculate another variable called new candidate values. The new candidate values are information which seem relevant are added to the cell state.

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_c.[h_{t-1}, x_t] + b_c)$$

→ The new cell state layer calculates the new cell state by updating the information from last cell. And the new cell state is calculated using the information acquired from the previous two layers.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

→ The output layer makes use of all this information gathered over the last three layers to produce an output.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

→ Also, the cell state at the top of the Figure starting with $c(t-1)$ runs horizontally as it keeps the information integrity from long period of time with some minor linear attractions.

Time Series

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.

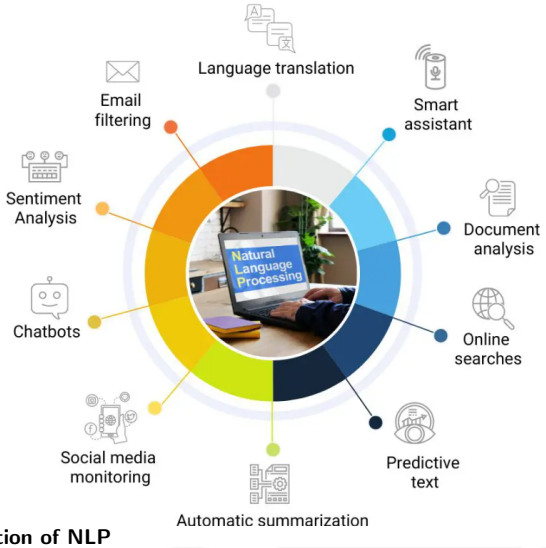
• **Generalized Additive Model** - combine predictive methods while preserving additivity across variables, in a form such as $y = \beta_0 + f_1(x_1) + \cdots + f_m(x_m)$, where functions can be non-linear.
→ GAMs also provide regularized and interpretable solutions for regression and classification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

Tutorial: Complete Guide on Time Series Analysis in Python

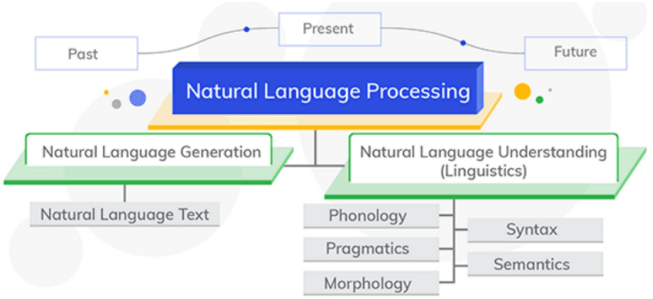
Natural Language Processing

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**

Why	not	tell	someone	?
adverb	adverb	verb	noun	punctuation mark, sentence closer

- **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

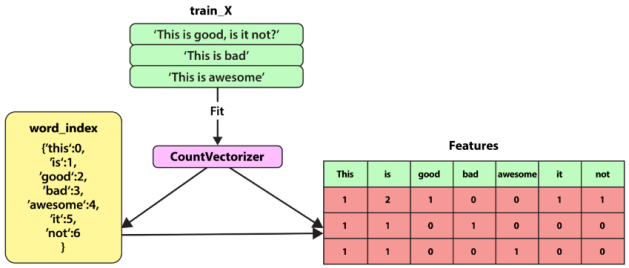
Corpus
people watch campusx campusx watch campusx people write comment campusx write comment

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.

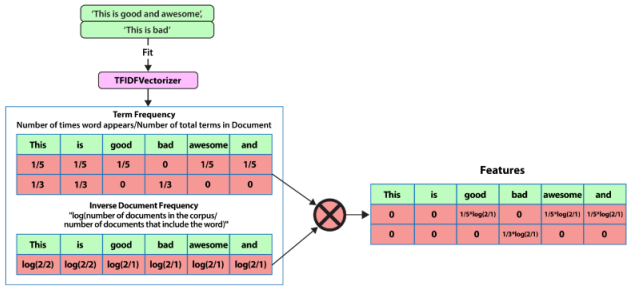
- n-**gram** - predicts the next term in a sequence of n terms based on Markov chains
  → **Markov Chain** - stochastic and memoryless process that predicts future events based only on the current state

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 o to 1



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

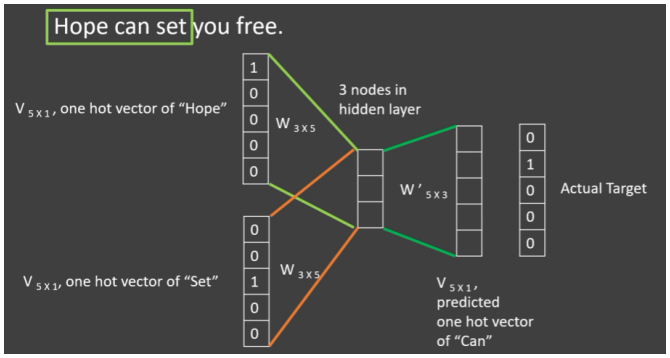
→ **CountVectorizer** - Bag of Words
→ **TfidfTransformer** - TF-IDF values
→ **TfidfVectorizer** - Bag of Words AND TF-IDF values

Word Embedding

Word embeddings are often based on neural network models in deep learning.

• **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 $king - man + woman \approx queen$
• Relies on one of the following:
– **Continuous bag-of-words** (CBOW) - predicts the word given its context
– **skip-gram** - predicts the context given a word

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

• **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