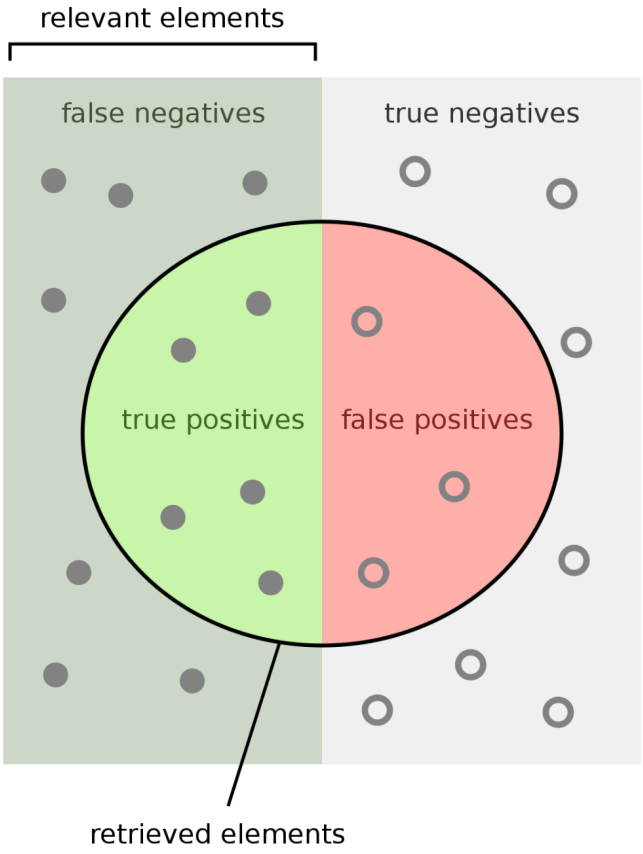
Machine Learning Breadth Interview Preparation Study Guide

Name: John Hodge

Date: 05/13/24

Precision vs. Recall

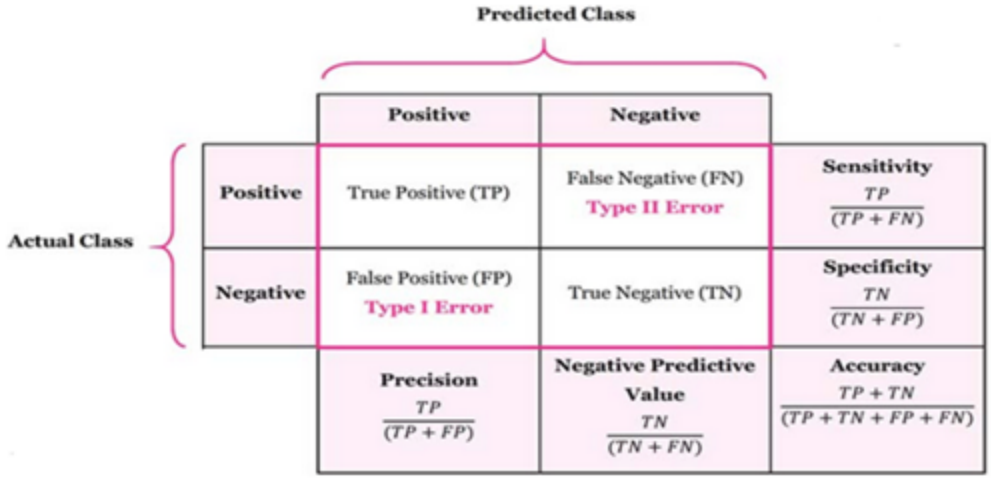


How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{green}}{\text{green + red}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{green}}{\text{green + dark green}}$$



$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP} \\
specificity &= \frac{TN}{TN + FP}
\end{aligned}
$$

Precision and recall are two fundamental metrics used in the evaluation of classification models, especially when dealing with imbalanced datasets. They help provide insights into how well a model is performing with respect to both positive class prediction and the relevance of the results it provides. Here's a breakdown of each:

1. **Precision**: This metric indicates the accuracy of positive predictions. It is defined as the ratio of true positive results to the total number of examples predicted as positive. In simpler terms, precision answers the question: "Of all the examples the model labeled as positive, how many actually belong to the positive class?" Precision is crucial when the cost of a false positive is high.

   Formula:
   \[
   \text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP) + False Positives (FP)}}
   \]

2. **Recall** (also known as Sensitivity or True Positive Rate): This metric measures the model's ability to detect positive instances among the actual positive instances available during testing. It is defined as the ratio of true positive results to the actual number of examples that are positive. Recall addresses the question: "Of all the actual positives, how many did the model successfully identify?" This metric is critical when the cost of missing a positive (false negative) is severe.

   Formula:
   \[
   \text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP) + False Negatives (FN)}}
   \]

In practice, precision and recall are often in a trade-off relationship. Increasing precision typically reduces recall and vice versa, depending on the threshold set for classifying a prediction as positive. This is why another metric, called the F1-score, is often used to find a balance between precision and recall, providing a single measure of a model's accuracy that factors in both metrics.

Would you like to delve deeper into how these metrics are applied or maybe explore some examples?

https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

ROC-AUC

https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

An **ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

AUC represents the probability that a random positive (green) example is positioned to the right of a random negative (red) example.

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases:

- **Scale invariance is not always desirable.** For example, sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that.
- **Classification-threshold invariance is not always desirable.** In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. For example, when doing email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.

Adam Optimizer vs. SGD

The Adam optimizer and stochastic gradient descent (SGD) are both algorithms used for optimizing the training process of neural networks, but they have key differences in how they approach the optimization problem, which often makes Adam more effective in practice.

Stochastic Gradient Descent (SGD)
SGD is one of the simplest and oldest optimization techniques used for training neural networks. It updates the model's weights by taking a step proportional to the negative of the gradient of the objective function with respect to the weights. The basic formula for the weight update is:
$$ w = w - \eta \nabla_w J(w) $$
where $ w $ represents the weights, $ \eta $ is the learning rate, and $ \nabla_w J(w) $ is the gradient of the loss function $ J $ with respect to the weights $ w $.

SGD updates weights using the same learning rate for all parameters and does not take into account the history of gradients. This can lead to several issues such as slow convergence, getting stuck in local minima, or issues with the chosen learning rate not being adaptive.

Adam Optimizer
Adam, short for "Adaptive Moment Estimation," builds on SGD by incorporating both the averages of the first moments (the mean) and the second moments (the uncentered variance) of the gradients. This allows Adam to adjust the learning rate for each weight individually by:
1. Calculating an exponentially weighted average of past gradients (momentum).
2. Calculating an exponentially weighted average of past squared gradients (this adjusts the learning rate based on the variance of the gradients).

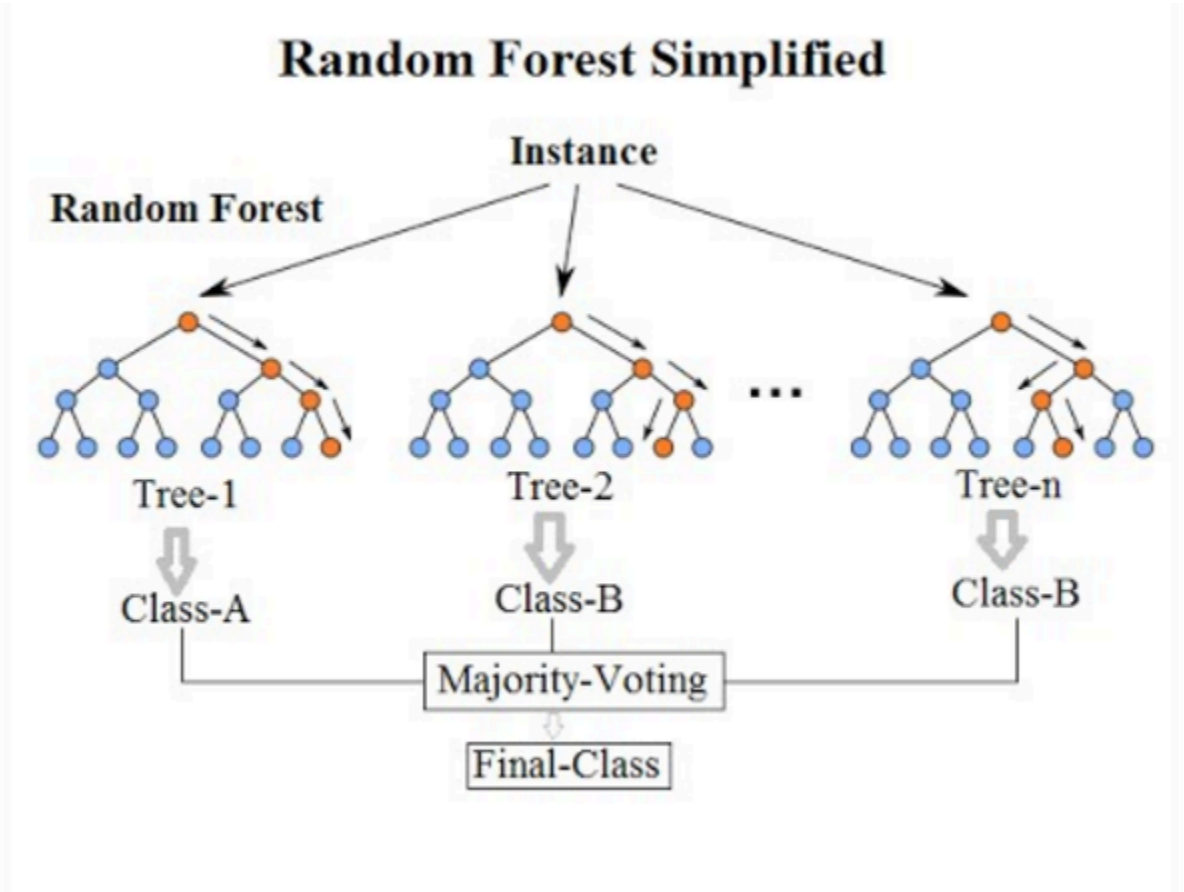
The update rules for Adam are given by:
$$ m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_w J(w) $$
$$ v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_w J(w))^2 $$
$$ \hat{m}_t = \frac{m_t}{1 - \beta_1^t} $$
$$ \hat{v}_t = \frac{v_t}{1 - \beta_2^t} $$
$$ w = w - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} $$
where β_1 and β_2 are decay rates for the moving averages, generally set to values like 0.9 and 0.999, respectively, and ϵ is a small constant (e.g., 10^{-8}) to prevent division by zero.

Why Adam is Often Used
Adam is favored in many neural network training scenarios because:
1. **Adaptive Learning Rates**: By adjusting the learning rate for each parameter, Adam often leads to faster convergence.
2. **Built-in Momentum**: By considering the first moment of the gradients, it incorporates a momentum mechanism that helps to accelerate the gradients vectors in the right directions, thus leading to faster converging.
3. **Handling Sparse Gradients**: Adam works well with sparse gradients (common in fields like natural language processing), as it inherently performs smaller updates for infrequent parameters.

Due to these advantages, Adam is often the go-to optimizer for training deep neural networks, especially when dealing with problems that involve complex data structures and large datasets. However, it's important to note that Adam can sometimes lead to poor generalization compared to SGD with carefully tuned learning rates and momentum, especially in settings where minimal overfitting is crucial.

https://chat.openai.com/share/d80bc04a-a35d-4ed6-9dfe-d2537752052c

Random Forests



A form of ensemble classifier. A random forest is a machine learning algorithm that combines the results of multiple decision trees to produce a single result. It's a popular ensemble learning method used for classification, regression, and other tasks.

Random Forest is a versatile machine learning algorithm capable of performing both regression and classification tasks. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model. Here's a breakdown of how Random Forest works, particularly focusing on classification:

Basic Concept
Random Forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. One big advantage of Random Forest over a single decision tree is that it reduces the risk of overfitting, which is a common problem with decision trees.

How It Works

1. **Bootstrap Aggregating (Bagging):** Random Forest starts by performing bootstrap aggregating, or bagging, where multiple subsets of the original dataset are created with replacement. This means each subset can have duplicate records, and some records may be left out of some subsets.

2. **Building Decision Trees:** A decision tree is built for each of these subsets. The tree construction is done by randomly selecting a subset of features at each node to determine the split. The number of features that can be selected is less than the total number of features and is a key parameter that can be adjusted. This randomness in feature selection contributes to making the trees diverse and robust against overfitting.

3. **Tree Growth:** Each tree in a Random Forest grows to its maximum length and is not pruned (though this can be adjusted depending on the implementation). This full growth usually involves creating very deep trees that are highly complex and can capture a lot of detailed data points.

4. **Prediction:** For a classification problem, each tree in the forest outputs a class prediction, and the final output prediction is the class that receives the majority of the votes from the trees. In essence, Random Forest applies a "majority rules" principle where each individual tree votes for a class, and the class with the most votes becomes the model’s prediction.

Key Parameters
- **Number of Trees:** The number of trees in the forest. More trees increase the prediction robustness but also the computational complexity.
- **Max Features:** The number of features to consider when looking for the best split at each node. Common values are the square root of the total features or about one-third of the total.
- **Max Depth:** The maximum depth of each tree. Deeper trees can learn more detailed data specifics but can lead to overfitting.
- **Min Samples Split:** The minimum number of samples a node must have before it can be split.

Advantages
- **Robustness:** Due to averaging multiple trees, it is less sensitive to outliers and noise.
- **Less Overfitting:** Randomness in feature selection and bootstrap aggregating help it to generalize better.
- **Feature Importance:** It can handle large data sets with higher dimensionality well and can provide estimates of what variables are important in the underlying data being modeled.

Disadvantages
- **Complexity:** More computationally intensive than other algorithms, especially with a large number of trees.
- **Interpretability:** Less interpretable than individual decision trees due to its complex ensemble structure.

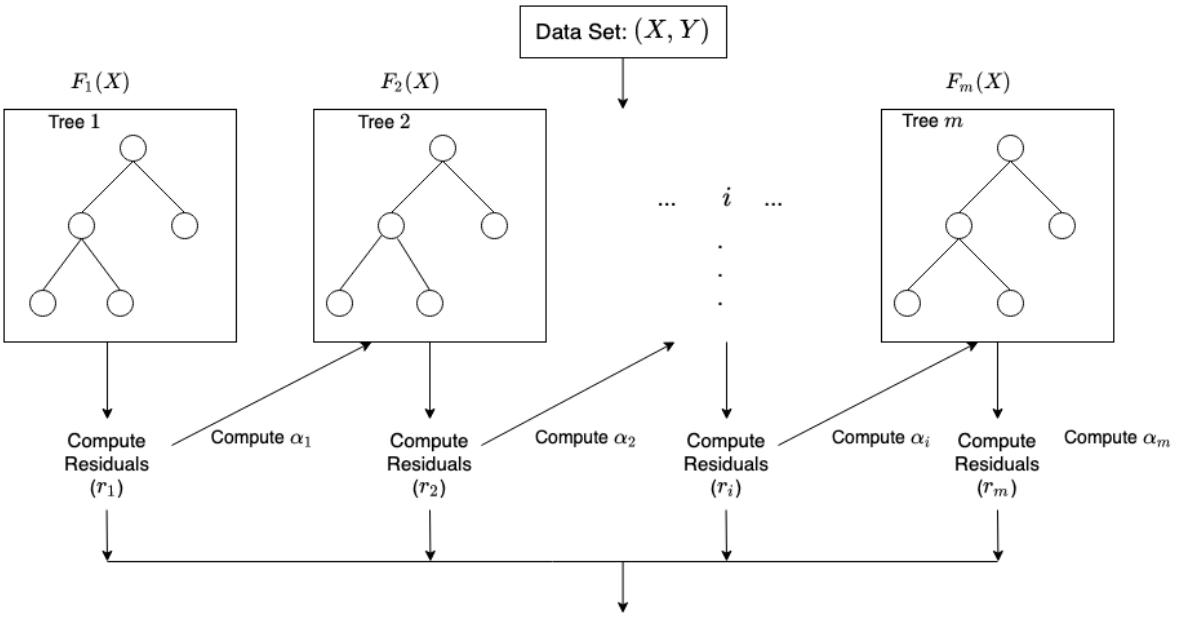
Random Forest is widely used in various fields, from banking for credit scoring to e-commerce for recommendation engines, due to its versatility, ease of use, and robust performance on many tasks.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://www.ibm.com/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly,both%20classification%20and%20regression%20problems.

XGBoost

XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.



https://hastie.su.domains/ElemStatLearn/printings/ESLII_print12.pdf#page=380

Gradient Boosting

Gradient boosting is a machine learning technique used for both regression and classification problems. It builds an ensemble of weak prediction models, typically decision trees, in a stage-wise fashion. Here's a detailed breakdown of how gradient boosting works:

1. **Base Model Setup:**
Gradient boosting starts with a simple model. Often, this initial model is just a single prediction that minimizes the loss function (like the mean of the target values for regression).

2. **Learning from Mistakes:**
The core idea is to sequentially add new models (usually decision trees) that effectively correct the errors made by previous models. The method focuses on improving the accuracy in areas where the previous models performed poorly.

3. **Gradient Descent on the Loss Function:**
Instead of updating the model parameters directly, gradient boosting updates the model by fitting new models to the residual errors made by previous predictions. This is analogous to performing a step of gradient descent on the loss function that measures the difference between the predicted and actual values. The "gradient" in gradient boosting refers to the gradient of the loss function, which guides how the model's predictions should be corrected.

4. **Additive Model Construction:**
Each new tree added is fitted on the residual errors of the ensemble thus far. The predictions from all the trees are then summed together to make the final prediction. This additive model construction continues until a specified number of trees are added, or if adding new trees ceases to improve the accuracy significantly.

5. **Regularization Techniques:**
To prevent overfitting, gradient boosting incorporates several regularization techniques such as limiting the number of trees, tree depth, learning rate (also known as shrinkage or step-size), and employing subsampling techniques like Stochastic Gradient Boosting.

6. **Output Decision:**
For classification, the ensemble of trees votes or averages to predict the class. For regression, the outputs of all trees are summed up to predict the continuous value.

Applications and Strengths:
- **Robustness to Overfitting:** Especially with small datasets and when using regularization techniques.

- **Handling of Heterogeneous Features:** Effective with datasets where features are of different types (e.g., binary, categorical, continuous).
- **Feature Importance:** Can be used to rank features in terms of their usefulness.

Gradient boosting is powerful but also computationally demanding compared to models like random forests, especially as the number of trees increases. Libraries like XGBoost, LightGBM, and CatBoost have optimized implementations of gradient boosting that are widely used in machine learning competitions and real-world applications due to their speed and performance.

Bias vs. Variance Trade-off

Cheat Sheet – Bias-Variance Tradeoff

What is Bias? $bias = \mathbb{E}[f'(x)] - f(x)$

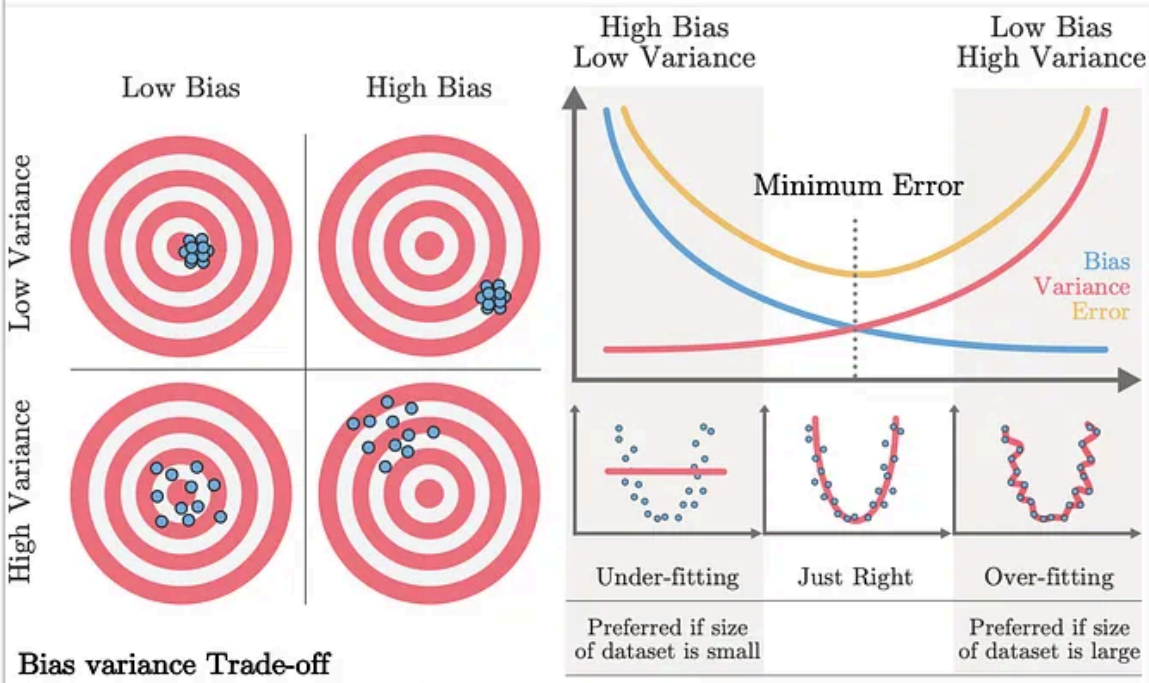
- Error between average model prediction and ground truth
- The bias of the estimated function tells us the capacity of the underlying model to predict the values

What is Variance? $variance = \mathbb{E}\left[\left(f'(x) - \mathbb{E}[f'(x)]\right)^2\right]$

- Average variability in the model prediction for the given dataset
- The variance of the estimated function tells you how much the function can adjust to the change in the dataset

High Bias
- → Overly-simplified Model
- → Under-fitting
- → High error on both test and train data

High Variance
- → Overly-complex Model
- → Over-fitting
- → Low error on train data and high on test
- → Starts modelling the noise in the input



Bias variance Trade-off

- Increasing bias reduces variance and vice-versa
- Error = bias^2 + variance +irreducible error
- The best model is where the error is reduced.
- Compromise between bias and variance

Source: https://www.cheatsheets.aqeel-anwar.com

The trade-off between bias and variance is a fundamental concept in machine learning and statistics, related to the problem of overfitting and underfitting. Here's a closer look at each component and how they interact:

1. **Bias**: This refers to the error introduced by approximating a real-world problem, which may be complex, by a much simpler model. Bias measures how far off on average these model predictions are from the actual value. A high-bias model is overly simplistic – it doesn't learn well from the training data, missing relevant relations between features and target outputs (underfitting). Models with high bias are usually simple (linear models) and do not capture the complex patterns in data.

2. **Variance**: Variance measures how much the predictions for a given point vary between different realizations of the model. High variance suggests that the model is extremely sensitive to small fluctuations in the training set. High-variance models often follow the training data very closely, including the noise in the data, leading to overfitting. These models perform well on training data but poorly on any unseen test data.

The trade-off:
- A model with high bias and low variance will generalize better but at the cost of failing to capture important nuances (underfits).
- A model with low bias and high variance captures the training data very well but does not generalize to new, unseen data (overfits).

In practice, you want to find a balance between bias and variance that minimizes the total error. This is often managed by:
- Choosing the right model complexity that is appropriate for the size and variety of the data.
- Employing techniques like cross-validation to validate model performance on unseen data.
- Using regularization techniques (like LASSO, Ridge) that can penalize overly complex models, thus reducing variance without increasing bias too much.
- Adjusting model hyperparameters to find the optimal balance.

Finding this balance helps in building models that are both accurate and robust enough to perform well on new, unseen data.

Class imbalance

Cheat Sheet – Imbalanced Data in Classification

Classifier that always predicts label blue yields prediction accuracy of 90%

Accuracy doesn't always give the correct insight about your trained model

Accuracy: %age correct prediction	Correct prediction over total predictions	One value for entire network
Precision: Exactness of model	From the detected cats, how many were actually cats	Each class/label has a value
Recall: Completeness of model	Correctly detected cats over total cats	Each class/label has a value
F1 Score: Combines Precision/Recall	Harmonic mean of Precision and Recall	Each class/label has a value

Performance metrics associated with Class 1

Predicted Labels \ Actual Labels	1	0
1	True Positive	False Positive
0	False Negative	True Negative

(Is your prediction correct?) (What did you predict)

True Negative

(Your prediction is correct) (You predicted 0)

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

$$\text{F1 score} = 2x \frac{(\text{Prec} \times \text{Rec})}{(\text{Prec} + \text{Rec})}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

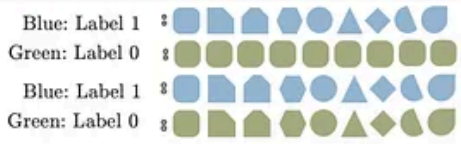
$$\text{False +ve rate} = \frac{FP}{TN + FP}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

$$\text{Recall, Sensitivity} = \text{True +ve rate} = \frac{TP}{TP + FN}$$

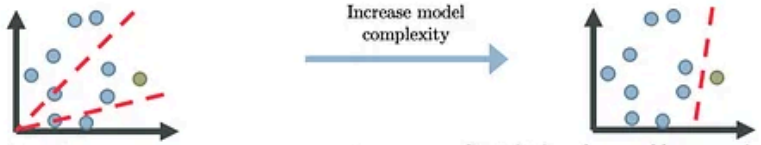
Possible solutions

1. **Data Replication:** Replicate the available data until the number of samples are comparable
2. **Synthetic Data:** Images: Rotate, dilate, crop, add noise to existing input images and create new data
3. **Modified Loss:** Modify the loss to reflect greater error when misclassifying smaller sample set



$$loss = a * loss_{green} + b * loss_{blue} \quad a > b$$

4. **Change the algorithm:** Increase the model/algorithm complexity so that the two classes are perfectly separable (Con: Overfitting)



No straight line (y=ax) passing through origin can perfectly separate data. **Best solution:** line y=0, predict all labels blue

Straight line (y=ax+b) can perfectly separate data. Green class will no longer be predicted as blue

Source: https://www.cheatsheets.aqeel-anwar.com

https://scikit-learn.org/stable/modules/model_evaluation.html#classification-report

Examples

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
              precision    recall  f1-score   support

     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3

    accuracy                           0.60         5
   macro avg       0.50      0.56      0.49         5
weighted avg       0.70      0.60      0.61         5

>>> y_pred = [1, 1, 0]
>>> y_true = [1, 1, 1]
>>> print(classification_report(y_true, y_pred, labels=[1, 2, 3]))
              precision    recall  f1-score   support

           1       1.00      0.67      0.80         3
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0

   micro avg       1.00      0.67      0.80         3
   macro avg       0.33      0.22      0.27         3
weighted avg       1.00      0.67      0.80         3
```

average*{'micro', 'macro', 'samples', 'weighted', 'binary'} or None, default='binary'*
This parameter is required for multiclass/multilabel targets. If `None`, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data:

`'binary'`:
Only report results for the class specified by `pos_label`. This is applicable only if targets (`y_{true,pred}`) are binary.

`'micro'`:
Calculate metrics globally by counting the total true positives, false negatives and false positives.

`'macro'`:
Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

`'weighted'`:

Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters ‘macro’ to account for label imbalance; it can result in an F-score that is not between precision and recall.

`'samples'`:
Calculate metrics for each instance, and find their average (only meaningful for multilabel classification where this differs from `accuracy_score`).

Bayes Thereom / Bayesian Inference

Cheat Sheet – Bayes Theorem and Classifier

What is Bayes' Theorem?

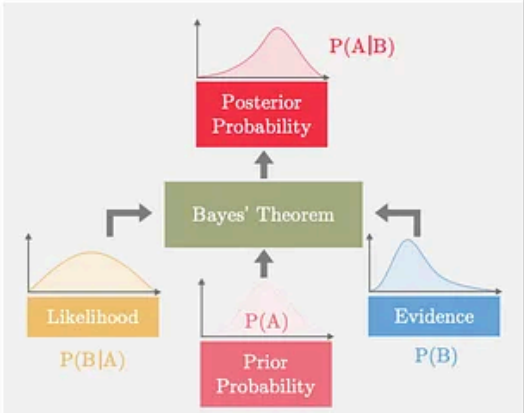
- Describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

$$P(A|B) = \frac{P(B|A)(likelihood) \times P(A)(prior)}{P(B)(prior)}$$

- How the probability of an event changes when we have knowledge of another event

P(A) ⟶ P(A|B)

Usually a better estimate than P(A)



Example

- Probability of fire P(F) = 1%
- Probability of smoke P(S) = 10%
- Prob of smoke given there is a fire P(S|F) = 90%
- What is the probability that there is a fire given we see a smoke P(F|S)?

$$P(F|S) = \frac{P(S|F) \times P(F)}{P(S)} = \frac{0.9 \times 0.01}{0.1} = 9\%$$

Maximum Apriori Probability (MAP) Estimation

The MAP estimate of the random variable y, given that we have observed iid (x_1, x_2, x_3, …), is given by. We try to accommodate our prior knowledge when estimating.

$$\hat{y}_{MAP} = argmax_y \; P(y) \prod_i P(x_i|y)$$

y that maximizes the product of prior and likelihood

Maximum Likelihood Estimation (MLE)

The MAP estimate of the random variable y, given that we have observed iid (x_1, x_2, x_3, …), is given by. We assume we don't have any prior knowledge of the quantity being estimated.

$$\hat{y}_{MLE} = argmax_y \prod_i P(x_i|y)$$

y that maximizes only the likelihood

MLE is a special case of MAP where our prior is uniform (all values are equally likely)

Naïve Bayes' Classifier (Instantiation of MAP as classifier)

Suppose we have two classes, $y=y_1$ and $y=y_2$. Say we have more than one evidence/features (x_1, x_2, x_3, …), using Bayes' theorem

$$P(y|x_1, x_2, x_3, \ldots) = \frac{P(x_1, x_2, x_3, \ldots|y) \times P(y)}{P(x_1, x_2, x_3, \ldots)}$$

Bayes' theorem assumes the features (x_1, x_2, x_3, …) are i.i.d. i.e $P(x_1, x_2, x_3, \ldots|y) = \prod_i P(x_i|y)$

$$P(y|x_1, x_2, x_3, \ldots) = \prod_i P(x_i|y) \frac{P(y)}{P(x_1, x_2, x_3, \ldots)}$$

$$\hat{y} = y_1 \; if \; \frac{P(y_1|x_1, x_2, x_3, \ldots)}{P(y_2|x_1, x_2, x_3, \ldots)} > 1 \quad else \; \hat{y} = y_2$$

Principal Component Analysis and Dimensionality Reduction

Cheat Sheet – PCA Dimensionality Reduction

What is PCA?

- Based on the dataset find a new set of orthogonal feature vectors in such a way that the data spread is maximum in the direction of the feature vector (or dimension)
- Rates the feature vector in the decreasing order of data spread (or variance)
- The datapoints have maximum variance in the first feature vector, and minimum variance in the last feature vector
- The variance of the datapoints in the direction of feature vector can be termed as a measure of information in that direction.

Steps

1. Standardize the datapoints — $X_{new} = \frac{X - mean(X)}{std(X)}$
2. Find the covariance matrix from the given datapoints — $C[i,j] = cov(x_i, x_j)$
3. Carry out eigen-value decomposition of the covariance matrix — $C = V\Sigma V^{-1}$
4. Sort the eigenvalues and eigenvectors — $\Sigma_{sort} = sort(\Sigma)$ $V_{sort} = sort(V, \Sigma_{sort})$

Dimensionality Reduction with PCA

- Keep the first m out of n feature vectors rated by PCA. These m vectors will be the best m vectors preserving the maximum information that could have been preserved with m vectors on the given dataset

Steps:

1. Carry out steps 1-4 from above
2. Keep first m feature vectors from the sorted eigenvector matrix — $V_{reduced} = V[:, 0:m]$
3. Transform the data for the new basis (feature vectors) — $X_{reduced} = X_{new} \times V_{reduced}$
4. The importance of the feature vector is proportional to the magnitude of the eigen value

Figure 1

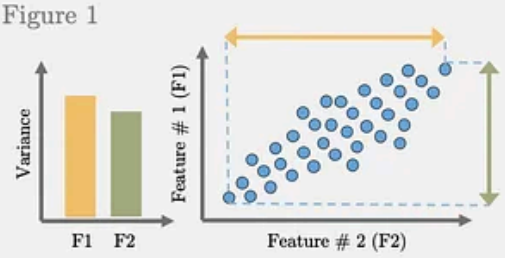


Figure 2

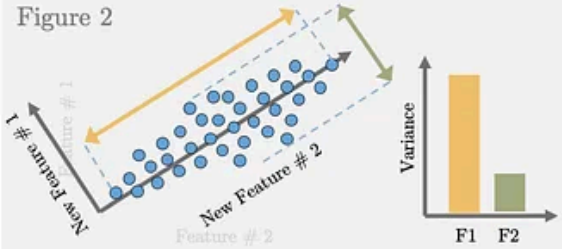


Figure 3

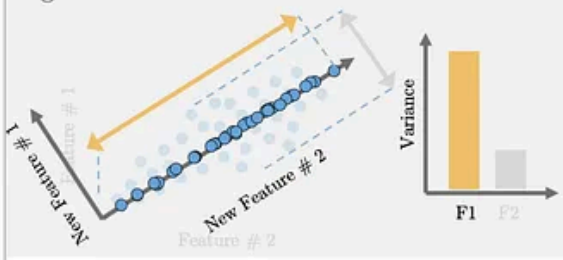


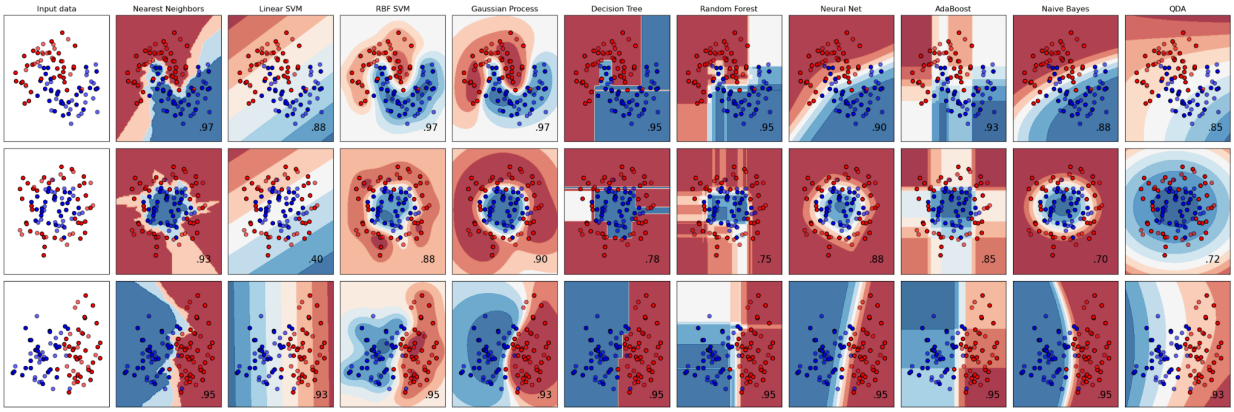
Figure 1: Datapoints with feature vectors as x and y-axis
Figure 2: The cartesian coordinate system is rotated to maximize the standard deviation along any one axis (new feature # 2)
Figure 3: Remove the feature vector with minimum standard deviation of datapoints (new feature # 1) and project the data on new feature # 2

Source: https://www.cheatsheets.aqeel-anwar.com

https://scikit-learn.org/stable/modules/decomposition.html#decompositions

Classifiers (Classification Problems)

https://scikit-learn.org/stable/supervised_learning.html#supervised-learning



Regression Problems

Cheat Sheet – Regression Analysis

What is Regression Analysis?
Fitting a function f(.) to datapoints y_i=f(x_i) under some error function. Based on the estimated function and error, we have the following types of regression

1. **Linear Regression:**
   Fits a line minimizing the sum of mean-squared error for each datapoint.

$$min_\beta \sum_i \|y_i - f_\beta^{linear}(x_i)\|^2$$
$$f_\beta^{linear}(x_i) = \beta_0 + \beta_1 x_i$$

2. **Polynomial Regression:**
   Fits a polynomial of order k (k+1 unknowns) minimizing the sum of mean-squared error for each datapoint.

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta^{poly}(x_i)\|^2$$
$$f_\beta^{poly}(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_k x_i^k$$

3. **Bayesian Regression:**
   For each datapoint, fits a gaussian distribution by minimizing the mean-squared error. As the number of data points x_i increases, it converges to point estimates i.e. $n \to \infty, \sigma^2 \to 0$

$$min_\beta \sum_i \|y_i - \mathcal{N}(f_\beta(x_i), \sigma^2)\|^2$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$
$$\mathcal{N}(\mu, \sigma^2) \to \text{Gaussian with mean } \mu \text{ and variance } \sigma^2$$

4. **Ridge Regression:**
   Can fit either a line, or polynomial minimizing the sum of mean-squared error for each datapoint and the weighted L2 norm of the function parameters beta.

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{k} \beta_j^2$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$

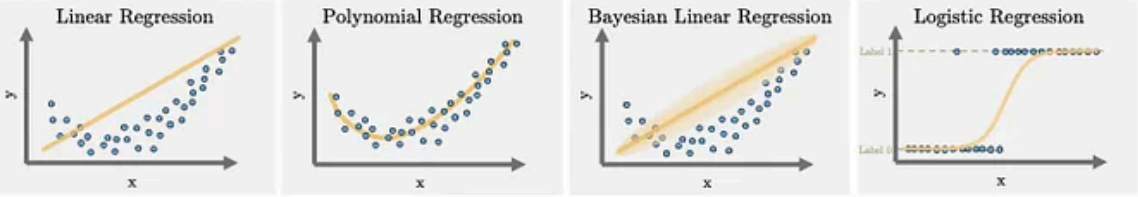
5. **LASSO Regression:**
   Can fit either a line, or polynomial minimizing the the sum of mean-squared error for each datapoint and the weighted L1 norm of the function parameters beta.

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{k} |\beta_j|$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$

6. **Logistic Regression:**
   Can fit either a line, or polynomial with sigmoid activation minimizing the binary cross-entropy loss for each datapoint. The labels y are binary class labels.

$$min_\beta \sum_i -y_i log(\sigma(f_\beta(x_i))) - (1 - y_i) log(1 - \sigma(f_\beta(x_i)))$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$
$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Visual Representation:



Summary:

	What does it fit?	Estimated function	Error Function
Linear	A line in n dimensions	$f_\beta^{linear}(x_i) = \beta_0 + \beta_1 x_i$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2$
Polynomial	A polynomial of order k	$f_\beta^{poly}(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2$
Bayesian Linear	Gaussian distribution for each point	$\mathcal{N}(f_\beta(x_i), \sigma^2)$	$\sum_i \|y_i - \mathcal{N}(f_\beta(x_i), \sigma^2)\|^2$
Ridge	Linear/polynomial	$f_\beta^{poly}(x_i)$ or $f_\beta^{linear}(x_i)$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{k} \beta_j^2$
LASSO	Linear/polynomial	$f_\beta^{poly}(x_i)$ or $f_\beta^{linear}(x_i)$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{k} \|\beta_j\|$
Logistic	Linear/polynomial with sigmoid	$\sigma(f_\beta(x_i))$	$min_\beta \sum_i -y_i log(\sigma(f_\beta(x_i))) - (1 - y_i) log(1 - \sigma(f_\beta(x_i)))$

Source: https://www.cheatsheets.aqeel-anwar.com

Starbucks project

https://github.com/jman4162/udacity-mle-starbucks-capstone-project

Algorithms and Techniques We perform customer segmentation using the k-means clustering algorithm and use principal component analysis (PCA) for dimensionality reduction and visualization. For the classification problems, we evaluate the logistic regression (LogReg), support vector machine (SVM), k-nearest neighbor (kNN), XGBoost, and deep neural network (DNN) algorithms. We evaluate the linear stochastic gradient descent (SGD) model, support vector regression (SVR), kNN, XGBoost, and DNN models for regression problems. XGBoost is an open-source and efficient implementation of the gradient boosted tree algorithm. Gradient boosting is a supervised learning algorithm used in one of the course lessons to predict housing price data. I used DNN models extensively using PyTorch in the Deep Learning Nanodegree program.

Project Overview

https://chat.openai.com/share/6132b813-148d-4ff1-97ab-806a6599aefb

In your capstone project for the Udacity Machine Learning Engineer Nanodegree, you focused on leveraging the Starbucks app data to tackle two main business challenges:

1. **Predicting Customer Engagement with Promotions**: You developed predictive models to ascertain whether a customer would view and complete a promotional offer based on their demographic information and the characteristics of the offer. This involved a classification approach, using metrics like precision, accuracy, recall, and the area under the ROC curve to evaluate model performance.

2. **Forecasting Revenue from Promotions**: You created a regression model to estimate the revenue generated from a completed offer after accounting for the cost of rewards. This model aimed to help Starbucks maximize ROI by predicting the financial outcome of promotional strategies. Key metrics for this model included RMSE, R2 score, and explained variance.

Key Techniques and Technologies:
- You integrated data from multiple sources (portfolio, profile, and transaction datasets) and applied exploratory data analysis, data cleaning, and feature engineering.
- You used various machine learning algorithms including logistic regression, support vector machines, k-nearest neighbors, XGBoost, and deep neural networks implemented in Python, utilizing libraries like Scikit-learn and PyTorch.
- You evaluated these models against benchmark models (logistic regression for classification and linear SGD for regression) and optimized them through hyperparameter tuning.

Outcomes and Business Impact:
- Your models successfully predicted customer engagement with promotional offers and the associated revenue impacts, thereby enabling personalized marketing strategies.
- The analysis provided insights into customer behavior, helping to tailor promotions effectively and improving the strategic deployment of marketing resources.

Conclusion:
- This project not only demonstrated the application of machine learning to real-world business problems but also emphasized the value of data-driven decision-making in marketing strategies. You showcased how predictive analytics could enhance customer engagement and optimize promotional expenditures.

You can present these points in your interview to highlight your proficiency in handling complex datasets, applying advanced machine learning techniques, and driving business value through analytics.

Results overview

- XGBoost performed best for the classification problem
- DNN performed best for regression problem (revenue prediction)

Project overview from the report

The object of this capstone project is to answer two business challenges using the Starbucks dataset:

- I. Can we use data to build a predictive model to determine whether a customer will view and complete a promotional offer based on the offer characteristics and customer demographic information?
- II. Can we use offer characteristics and customer demographic data to predict how much revenue a completed offer will generate after subtracting reward costs?

I analyze five different machine learning (ML) predictive model solutions and compare their performance for three case studies. These predictive models allow the business to personalize the characteristics of each promotional offer based on each customer's demographic profile to maximize the probability that the customer views and completes a promotional offer. Additionally, the regression model predicts how much revenue a completed promotional offer generates after subtracting the reward cost to help the business maximize return on investment (ROI) per customer.

Conclusion from report

Overall, this project was surprisingly challenging due to the structure of the transcript dataset and the preprocessing required to generate helpful ML model predictions. However, I learned a lot in this project and developed solutions to the challenges discussed in the project proposal:
● I developed predictive machine learning models to classify whether views and completes a promotional offer based on personalized customer and offer characteristics.
● Developed a predictive model to determine how much a customer spends by completing an offer after subtracting out the cost of the reward
● The ML predictive model outperforms the benchmark model in each of our three studies, and the benchmark models also perform pretty well.
● Determined the features that are most significant to whether or not a customer views and completes a promotional offer
● Defined key customer segment groups based on customer demographics

The models and insights developed in this project allow the company to understand the primary drivers of an effective and profitable promotional offer. The predictive models will enable the company to personalize the characteristics of each promotion to maximize its odds of success for each customer. This project is an excellent example of how machine learning models benefit marketing campaigns and create business value. Lastly, this project shows why a more efficient and straightforward machine learning model is often preferable to a larger and more complex deep learning model.

Data Exploration

The structure of the dataset provided Starbucks Capstone project notebook is structured as follows. Three files contain the data:

● portfolio.json - containing offer ids and metadata about each offer (duration, type, etc.) (See Fig. 1)
● profile.json - demographic data for each customer (See Fig. 2)
● transcript.json - records for transactions, offers received, offers viewed, and offers completed (See Fig. 3)

The three types of offers presented in the _type column of portfolio.json are:
● Buy-one-get-one (BOGO): a user needs to spend a certain amount to get a reward equal to that threshold amount.
● Discount: a user gains a reward equal to a fraction of the amount spent.
● Informational offer: there is no reward, but neither is there a required amount that the user is expected to spend.

Regularization

Cheat Sheet – Regularization in ML

What is Regularization in ML?

- Regularization is an approach to address over-fitting in ML.
- Overfitted model fails to generalize estimations on test data
- When the underlying model to be learned is low bias/high variance, or when we have small amount of data, the estimated model is prone to over-fitting.
- Regularization reduces the variance of the model

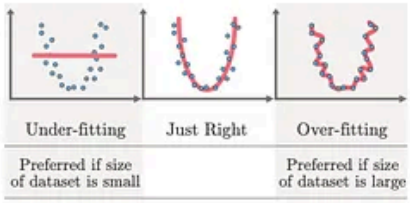


Figure 1. Overfitting

Types of Regularization:

1. Modify the loss function:

- **L2 Regularization:** Prevents the weights from getting too large (defined by L2 norm). Larger the weights, more complex the model is, more chances of overfitting.

$$loss = error(y, \hat{y}) \boxed{+ \lambda \sum_j \beta_j^2} \quad \lambda \geq 0,\ \lambda \propto model\ bias,\quad \lambda \propto \frac{1}{model\ variance}$$

- **L1 Regularization:** Prevents the weights from getting too large (defined by L1 norm). Larger the weights, more complex the model is, more chances of overfitting. L1 regularization introduces sparsity in the weights. It forces more weights to be zero, than reducing the the average magnitude of all weights

$$loss = error(y, \hat{y}) \boxed{+ \lambda \sum_j |\beta_j|} \quad \lambda \geq 0,\ \lambda \propto model\ bias,\quad \lambda \propto \frac{1}{model\ variance}$$

- **Entropy:** Used for the models that output probability. Forces the probability distribution towards uniform distribution.

$$loss = error(p, \hat{p}) \boxed{- \lambda \sum_i \hat{p}_i log(\hat{p}_i)} \quad \lambda \geq 0,\ \lambda \propto model\ bias,\quad \lambda \propto \frac{1}{model\ variance}$$

2. Modify data sampling:

- **Data augmentation:** Create more data from available data by randomly cropping, dilating, rotating, adding small amount of noise etc.
- **K-fold Cross-validation:** Divide the data into k groups. Train on (k-1) groups and test on 1 group. Try all k possible combinations.

3. Change training approach:

- **Injecting noise:** Add random noise to the weights when they are being learned. It pushes the model to be relatively insensitive to small variations in the weights, hence regularization
- **Dropout:** Generally used for neural networks. Connections between consecutive layers are randomly dropped based on a dropout-ratio and the remaining network is trained in the current iteration. In the next iteration, another set of random connections are dropped.

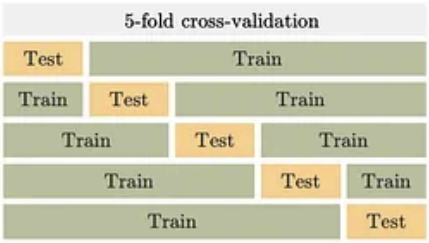


Figure 2. K-fold CV

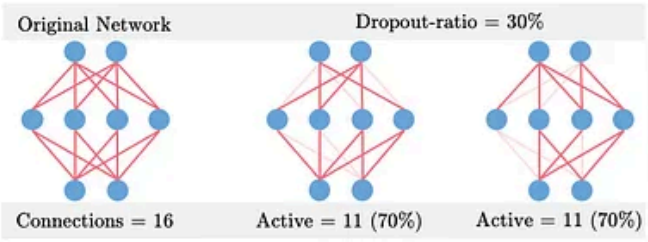


Figure 3. Drop-out

Source: https://www.cheatsheets.aqeel-anwar.com

Regularization is a technique in machine learning that helps to prevent models from overfitting the training data, which can lead to poor generalization to new, unseen data. Overfitting occurs when a model learns the noise or random fluctuations in the training data instead of the actual underlying patterns. Regularization addresses this by adding a penalty to the loss function that the model optimizes. Here are the key types of regularization:

1. **L1 Regularization (Lasso)**: Adds a penalty equal to the absolute value of the magnitude of coefficients. This can lead not only to smaller coefficients but can actually drive some coefficients to zero, thus performing variable selection. This is particularly useful when you suspect some features might be irrelevant to the output.

2. **L2 Regularization (Ridge)**: Adds a penalty equal to the square of the magnitude of coefficients. This tends to drive the coefficients to small values but not exactly zero. This is useful when you believe that all features have some influence on the output.

3. **Elastic Net**: Combines L1 and L2 regularization. It adds both penalties (absolute value and square of the coefficients) to the loss function. This method is particularly useful when there are correlations among the features, as it can maintain the group effect of variables.

Regularization parameters are hyperparameters that control the strength of the penalty. The values of these parameters are typically determined through hyperparameter tuning methods like cross-validation.

The choice of regularization can depend on the particularities of the dataset and the problem:

- **L1 regularization** is useful when trying to reduce the complexity of a model by eliminating some features entirely, which can be beneficial for models with high dimensionality.
- **L2 regularization** is generally good at dealing with multicollinearity and model overfitting but doesn't reduce the number of features, as it generally keeps all features in the model but reduces the magnitude of the coefficients.
- **Elastic Net** is best when there is a mix of features, some of which are important and others that are redundant or less important.

Regularization techniques can be applied across many types of models, including linear/logistic regression, neural networks, and even tree-based methods, although they manifest differently depending on the model architecture.

Data Preprocessing

Normalization

Data normalization is a crucial step in the data cleaning process for machine learning models for several key reasons:

1. **Feature Scaling**: Many machine learning algorithms, such as those involving gradient descent (like linear regression and neural networks), or those that use distance calculations (like k-means clustering and k-nearest neighbors), perform better when the feature data are scaled to a standard range, typically 0 to 1 or -1 to 1. Normalization ensures that each feature contributes equally to the distance computations, preventing features with larger scales from dominating the distance calculations.

2. **Speeds Up Learning**: Normalization can help in speeding up the learning process by helping the optimization algorithms converge more quickly. When features are on different scales, the gradients may oscillate back and forth and take longer paths towards convergence, slowing down the training process.

3. **Avoids Numerical Instability**: Algorithms can become numerically unstable due to very high or very low values. Normalization helps in avoiding these instabilities by ensuring the numbers stay within a range that is more manageable for the system.

4. **Improves Algorithm Performance**: Some algorithms, particularly those that assume data is normally distributed, such as logistic regression, can perform better if the features are normalized.

5. **Consistency**: When combining data from different sources, normalization is important to ensure that all data is on the same scale. This consistency is crucial when training a model to avoid biases toward one source or another due to scale differences.

Normalization, therefore, plays a pivotal role in preparing data for machine learning and ensuring that the predictive models are accurate, efficient, and robust.

Logistic regression

Is logistic regression a misnomer? (Yes, because it is not regression, but classification based on regression)

https://github.com/jman4162/machine-learning-review/blob/main/Review_Python_Tutorial_on_Logistic_Regression.ipynb

Logistic regression is a statistical model used for binary classification tasks, which predicts the probability that an observation falls into one of two categories. This makes it particularly useful in fields like medicine, finance, and social sciences for outcomes like disease/no disease, default/no default, etc. Unlike linear regression, logistic regression outputs probabilities by applying a logistic function to a linear combination of features.

A logit is the natural logarithm of odds, or the log of the odds that Y=1. It's a fundamental concept in machine learning, and is used in binary and multi-class classification problems.

$$\text{logit}(p) = \log(\frac{p}{1-p})$$

Support Vector Machine (SVM)

https://github.com/jman4162/machine-learning-review/blob/main/Python_Review_Tutorial_Support_Vector_Machines_(SVM).ipynb

Support Vector Machines (SVM) are a set of supervised learning methods used for classification, regression, and outliers detection. The advantages of support vector machines are:

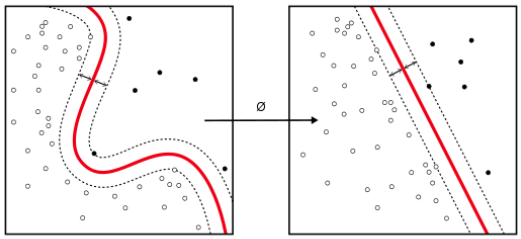
- Effective in high dimensional spaces.
- Still effective in cases where the number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Linear SVM and kernel SVM

Kernel tricks are used to map a non-linearly separable functions into a higher dimension linearly separable function. A support vector machine (SVM) training algorithm finds the classifier represented by the normal vector w and bias b of the hyperplane. This hyperplane (boundary) separates different classes by as wide a margin as possible. The problem can be converted into a constrained optimization problem:

$$\begin{aligned} &\underset{w}{\text{minimize}} && ||w|| \\ &\text{subject to} && y_i(w^T X_i - b) \geq 1, \ i = 1, \ldots, n. \end{aligned}$$

A support vector machine (SVM) training algorithm finds the classifier represented by the normal vector and bias of the hyperplane. This hyperplane (boundary) separates different classes by as wide a margin as possible. The problem can be converted into a constrained optimization problem:



Kernel tricks are used to map a non-linearly separable functions into a higher dimension linearly separable function.

When the classes are not linearly separable, a kernel trick can be used to map a non-linearly separable space into a higher dimension linearly separable space.

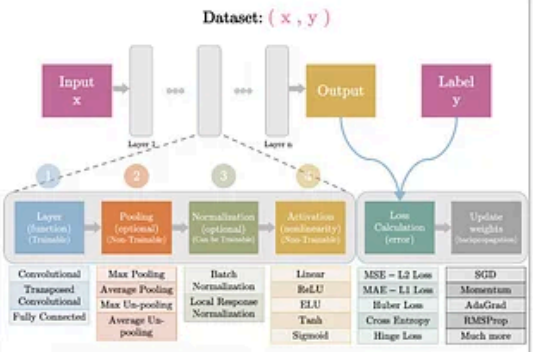
When most dependent variables are numeric, logistic regression and SVM should be the first try for classification. These models are easy to implement, their parameters easy to tune, and the performances are also pretty good. So these models are appropriate for beginners.

Convolutional Neural Networks (CNNs)

Cheat Sheet – Convolutional Neural Network

Convolutional Neural Network:

The data gets into the CNN through the input layer and passes through various hidden layers before getting to the output layer. The output of the network is compared to the actual labels in terms of loss or error. The partial derivatives of this loss w.r.t the trainable weights are calculated, and the weights are updated through one of the various methods using backpropagation.



CNN Template:

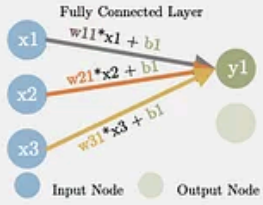
Most of the commonly used hidden layers (not all) follow a pattern

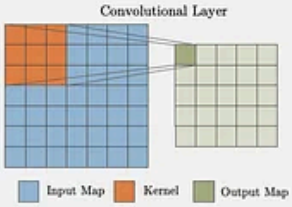
1. **Layer function:** Basic transforming function such as convolutional or fully connected layer.

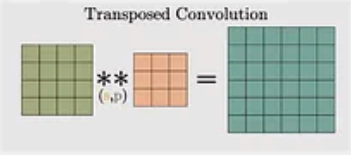
a. **Fully Connected:** Linear functions between the input and the output.

a. **Convolutional Layers:** These layers are applied to 2D (3D) input feature maps. The trainable weights are a 2D (3D) kernel/filter that moves across the input feature map, generating dot products with the overlapping region of the input feature map.

b. **Transposed Convolutional (DeConvolutional) Layer:** Usually used to increase the size of the output feature map (Upsampling) The idea behind the transposed convolutional layer is to undo (not exactly) the convolutional layer



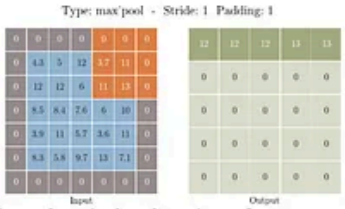




2. **Pooling:** Non-trainable layer to change the size of the feature map

a. **Max/Average Pooling:** Decrease the spatial size of the input layer based on selecting the maximum/average value in receptive field defined by the kernel

b. **UnPooling:** A non-trainable layer used to increase the spatial size of the input layer based on placing the input pixel at a certain index in the receptive field of the output defined by the kernel.



3. **Normalization:** Usually used just before the activation functions to limit the unbounded activation from increasing the output layer values too high

a. **Local Response Normalization LRN:** A **non-trainable layer** that square-normalizes the pixel values in a feature map within a local neighborhood.

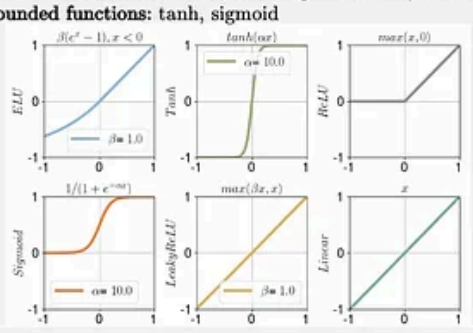
b. **Batch Normalization:** A trainable approach to normalizing the data by learning scale and shift variable during training.

3. **Activation:** Introduce non-linearity so CNN can efficiently map non-linear complex mapping.

a. **Non-parametric/Static functions:** Linear, ReLU

b. **Parametric functions:** ELU, tanh, sigmoid, Leaky ReLU

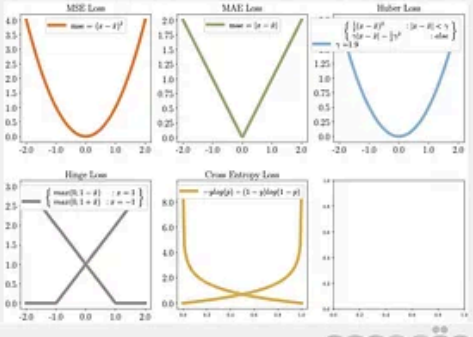
c. **Bounded functions:** tanh, sigmoid



5. **Loss function:** Quantifies how far off the CNN prediction is from the actual labels.

a. Regression Loss Functions: MAE, MSE, Huber loss

b. Classification Loss Functions: Cross entropy, Hinge loss

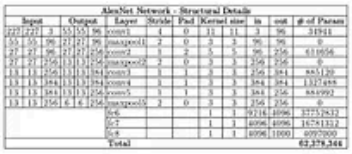


Source: https://www.cheatsheets.aqeel-anwar.com

Cheat Sheet – Famous CNNs

AlexNet – 2012

Why: AlexNet was born out of the need to improve the results of the ImageNet challenge.
What: The network consists of 5 Convolutional (CONV) layers and 3 Fully Connected (FC) layers. The activation used is the Rectified Linear Unit (ReLU).
How: Data augmentation is carried out to reduce over-fitting, Uses Local response localization.

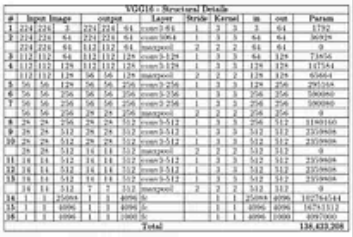


AlexNet Network - Structural Details

Input	Output	Layer	Stride	Pad	Kernel size	in	out	# of Params
227x227x 3	55x55x 96	conv-1	4	0	11	3	96	34944
55x 55x 96	27x27x 96	maxpool1	2	0	3	96	96	0
27x 27x 96	27x27x256	conv-2	1	2	5	96	256	614656
27x 27x256	13x13x256	maxpool2	2	0	3	256	256	0
13x13x256	13x13x384	conv-3	1	1	3	256	384	885120
13x13x384	13x13x384	conv-4	1	1	3	384	384	1327488
13x13x384	13x13x256	conv-5	1	1	3	384	256	884992
13x13x256	6 6 256	maxpool3	2	0	3	256	256	0
		fc-6			1	9216	4096	37753632
		fc-7			1	4096	4096	16781312
		fc-8			1	4096	1000	4097000
		Total						62,378,344

VGGNet – 2014

Why: VGGNet was born out of the need to reduce the # of parameters in the CONV layers and improve on training time
What: There are multiple variants of VGGNet (VGG16, VGG19, etc.)
How: The important point to note here is that all the conv kernels are of size 3x3 and maxpool kernels are of size 2x2 with a stride of two.

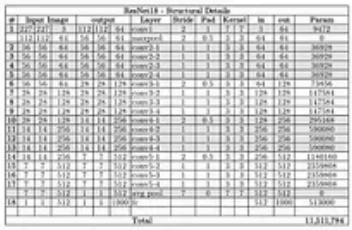


VGG16 - Structural Details

#	Input Image	output	Layer	Stride	Kernel	in	out	Param
1	224x224x 3	224x224x 64	conv3-64	1	3	3	64	1792
2	224x224x 64	224x224x 64	conv-3x64	1	3	64	64	36928
	224x224x 64	112x112x 64	maxpool	2	2	64	64	0
3	112x112x 64	112x112x 128	conv-3-128	1	3	64	128	73856
4	112x112x 128	112x112x 128	conv-3-128	1	3	128	128	147584
	112x112x 128	56 56 128	maxpool	2	2	128	128	65664
5	56 56 128	56 56 256	conv-3-256	1	3	128	256	295168
6	56 56 256	56 56 256	conv-3-256	1	3	256	256	590080
7	56 56 256	56 56 256	conv-3-256	1	3	256	256	590080
	56 56 256	28 28 256	maxpool	2	2	256	256	0
8	28 28 256	28 28 512	conv-3-512	1	3	256	512	1180160
9	28 28 512	28 28 512	conv-3-512	1	3	512	512	2359808
10	28 28 512	28 28 512	conv-3-512	1	3	512	512	2359808
	28 28 512	14 14 512	maxpool	2	2	512	512	0
11	14 14 512	14 14 512	conv-3-512	1	3	512	512	2359808
12	14 14 512	14 14 512	conv-3-512	1	3	512	512	2359808
13	14 14 512	14 14 512	conv-3-512	1	3	512	512	2359808
	14 14 512	7 7 512	maxpool	2	2	512	512	0
14	1 1 25088	1 1 4096	fc		1	25088	4096	102764544
15	1 1 4096	1 1 4096	fc		1	4096	4096	16781312
16	1 1 4096	1 1 1000	fc		1	4096	1000	4097000
		Total						138,422,208

ResNet – 2015

Why: Neural Networks are notorious for not being able to find a simpler mapping when it exists. ResNet solves that.
What: There are multiple versions of ResNetXX architectures where 'XX' denotes the number of layers. The most used ones are ResNet50 and ResNet101. Since the vanishing gradient problem was taken care of (more about it in the How part), CNN started to get deeper and deeper
How: ResNet architecture makes use of shortcut connections do solve the vanishing gradient problem. The basic building block of ResNet is a Residual block that is repeated throughout the network.



ResNet18 - Structural Details

#	Input Image	output	Layer	Stride	Pad	Kernel	in	out	Param
1	227x227x 3	112x112x 64	conv-1	2	3	7	3	64	9472
	112x112x 64	56 56 64	maxpool	2	0.5	3x3	64	64	0
2	56 56 64	56 56 64	conv-2-1	1	1	3x3	64	64	36928
3	56 56 64	56 56 64	conv-2-2	1	1	3x3	64	64	36928
4	56 56 64	56 56 64	conv-2-3	1	1	3x3	64	64	36928
5	56 56 64	56 56 64	conv-2-4	1	1	3x3	64	64	36928
6	56 56 64	28 28 128	conv-3-1	2	0.5	3x3	64	128	73856
7	28 28 128	28 28 128	conv-3-2	1	1	3x3	128	128	147584
8	28 28 128	28 28 128	conv-3-3	1	1	3x3	128	128	147584
9	28 28 128	28 28 128	conv-3-4	1	1	3x3	128	128	147584
10	28 28 128	14 14 256	conv-4-1	2	0.5	3x3	128	256	295168
11	14 14 256	14 14 256	conv-4-2	1	1	3x3	256	256	590080
12	14 14 256	14 14 256	conv-4-3	1	1	3x3	256	256	590080
13	14 14 256	14 14 256	conv-4-4	1	1	3x3	256	256	590080
14	14 14 256	7 7 512	conv-5-1	2	0.5	3x3	256	512	1180160
15	7 7 512	7 7 512	conv-5-2	1	1	3x3	512	512	2359808
16	7 7 512	7 7 512	conv-5-3	1	1	3x3	512	512	2359808
17	7 7 512	7 7 512	conv-5-4	1	1	3x3	512	512	2359808
	7 7 512	1 1 512	avg_pool	2	0	7	512	512	0
18	1 1 513	1 1 1000	fc				513	1000	513000
		Total							11,511,784

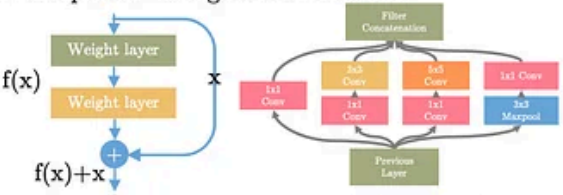
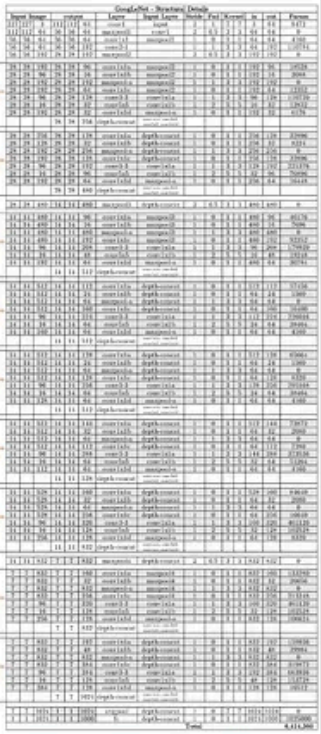


Figure 1 ResNet Block

Figure 2 Inception Block

Inception – 2014

Why: Lager kernels are preferred for more global features, on the other hand, smaller kernels provide good results in detecting area-specific features. For effective recognition of such a variable-sized feature, we need kernels of different sizes. That is what Inception does.
What: The Inception network architecture consists of several inception modules of the following structure. Each inception module consists of four operations in parallel, 1x1 conv layer, 3x3 conv layer, 5x5 conv layer, max pooling
How: Inception increases the network space from which the best network is to be chosen via training. Each inception module can capture salient features at different levels.



GoogLeNet - Structural Details

Comparison

Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

Source: https://www.cheatsheets.aqeel-anwar.com

Transposed Convolutional Layer

A convolutional layer and a transposed convolutional layer (also known as a deconvolutional layer or a fractionally strided convolutional layer) are both fundamental components in convolutional neural networks (CNNs), but they serve different purposes and operate differently.

https://chat.openai.com/share/ac406ba1-6dab-457b-8339-74a82ac919b9

Used in GAN

In summary, convolutional layers are primarily used for downsampling and feature extraction, whereas transposed convolutional layers are used for upsampling and reconstructing images to higher resolutions.

Ensemble Learning

Cheat Sheet – Ensemble Learning in ML

What is Ensemble Learning? Wisdom of the crowd

Combine multiple weak models/learners into one predictive model to reduce bias, variance and/or improve accuracy.

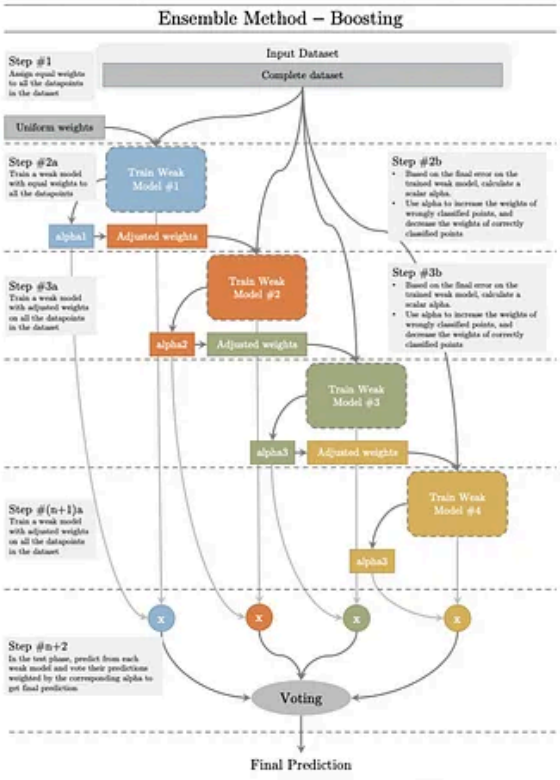
Types of Ensemble Learning: N number of weak learners

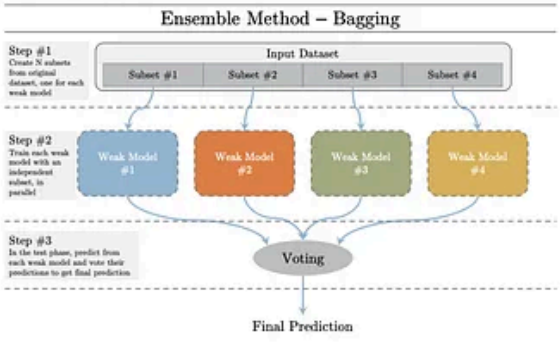
1.Bagging: Trains N different weak models (usually of same types – homogenous) with N non-overlapping subset of the input dataset in parallel. In the test phase, each model is evaluated. The label with the greatest number of predictions is selected as the prediction. Bagging methods reduces variance of the prediction

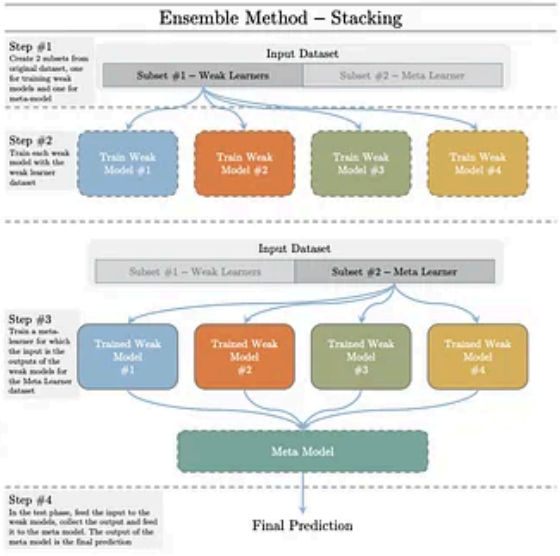
2.Boosting: Trains N different weak models (usually of same types – homogenous) with the complete dataset in a sequential order. The datapoints wrongly classified with previous weak model is provided more weights to that they can be classified by the next weak leaner properly. In the test phase, each model is evaluated and based on the test error of each weak model, the prediction is weighted for voting. Boosting methods decreases the bias of the prediction.

3.Stacking: Trains N different weak models (usually of different types – heterogenous) with one of the two subsets of the dataset in parallel. Once the weak learners are trained, they are used to trained a meta learner to combine their predictions and carry out final prediction using the other subset. In test phase, each model predicts its label, these set of labels are fed to the meta learner which generates the final prediction.

The block diagrams, and comparison table for each of these three methods can be seen below.







Parameter	Bagging	Boosting	Stacking
Focuses on	Reducing variance	Reducing bias	Improving accuracy
Nature of weak learners is	Homogenous	Homogenous	Heterogenous
Weak learners are aggregated by	Simple voting	Weighted voting	Learned voting (meta-learner)

Source: https://www.cheatsheets.aqeel-anwar.com

Decision trees, random forest and gradient boosting are all algorithms based on decision trees. There are many variants of decision trees, but they all do the same thing – subdivide the feature space into regions with mostly the same label. Decision trees are easy to understand and implement. However, they tend to over-fit data when we exhaust the branches and go very deep with the trees. Random Forrest and gradient boosting are two popular ways to use tree algorithms to achieve good accuracy as well as overcoming the over-fitting problem.

Boosting Methods

In machine learning, boosting methods are ensemble techniques that build strong models by combining multiple weak learners. Here are a few commonly used boosting methods:

1. **AdaBoost (Adaptive Boosting)**: One of the first and most popular boosting techniques, it adjusts the weights of incorrectly classified instances so that subsequent classifiers focus more on difficult cases.

2. **Gradient Boosting**: This method builds models sequentially, each new model correcting errors made by the previous ones. The model adjustments are made by fitting the new model to the gradient of the loss function used for the training.

3. **XGBoost (Extreme Gradient Boosting)**: An optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework.

4. **LightGBM (Light Gradient Boosting Machine)**: A gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient with faster training speed and lower memory usage.

5. **CatBoost (Category Boosting)**: An algorithm that handles categorical features automatically and is robust to different types of data and overfitting. It provides a state-of-the-art performance with categorical data.

These methods are widely used across various domains for tasks like classification, regression, and ranking due to their effectiveness in improving prediction accuracy.

Stacking Methods

Model stacking is a powerful ensemble technique used to improve the performance of machine learning models by combining multiple models to form a final prediction. The general approach to model stacking involves multiple layers of models, where each layer learns to correct the mistakes of the previous one. Here's a step-by-step breakdown of how model stacking is typically performed:

Step 1: Split the Data
The dataset is divided into at least two subsets: a training set and a holdout set. In more sophisticated setups, you might use K-fold cross-validation to create multiple training and validation folds.

Step 2: Train Base Models
Multiple base models (also known as level-0 models) are trained independently on the training set. These models can be of different types, including (but not limited to) decision trees, neural networks, SVMs, and more. The key is diversity among the models to capture various patterns in the data.

Step 3: Generate Meta-features
The base models are used to predict the outcomes for the validation set (or each fold in cross-validation). The predictions from these base models serve as meta-features for the next layer. This means that instead of the original features, the inputs for the next model layer are the predictions made by the base models.

Step 4: Train a Meta-model
A new model, known as the meta-model or level-1 model, is trained on the meta-features. The goal of this meta-model is to learn how best to combine the predictions of the base models to make a final prediction. This model essentially learns the strengths and weaknesses of the base models based on how well they perform on the validation set.

Step 5: Model Evaluation
Finally, the meta-model is evaluated using a separate test set (the holdout set), which was not used during the training of base models and the meta-model. This helps in assessing how well the stacking model generalizes on unseen data.

Optional: Multiple Layers
Although not always necessary, some stacking approaches may include more than one layer of meta-models, forming a deeper stack. Each additional layer aims to further refine the predictions made by the previous layer's models.

Final Prediction

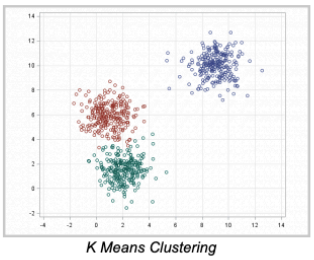
The final predictions are made by the meta-model, which uses the base models' predictions as inputs. This approach often leads to better predictive performance compared to any single model or simple averaging techniques, as it effectively leverages the strengths of multiple models while mitigating their individual weaknesses.

Stacking is particularly useful in competitions like those on Kaggle, where small improvements in predictive accuracy can be crucial. It's a flexible approach, allowing for various configurations and adaptations based on specific needs and constraints of the problem at hand.

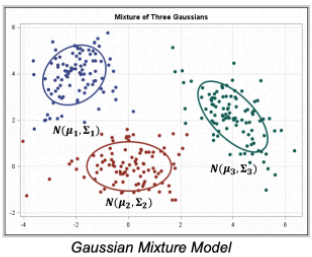
Clustering

K-means and GMM

k-means/k-modes, GMM (Gaussian mixture model) clustering



K Means Clustering



Gaussian Mixture Model

Kmeans/k-modes, GMM clustering aims to partition n observations into k clusters. K-means define hard assignment: the samples are to be and only to be associated to one cluster. GMM, however, defines a soft assignment for each sample. Each sample has a probability to be associated with each cluster. Both algorithms are simple and fast enough for clustering when the number of clusters k is given.

Gaussian Mixture Model (GMM)

https://chat.openai.com/share/4f02d619-68a9-460b-991d-4bfe34011143

Gaussian Mixture Models (GMMs) are probabilistic models used for representing normally distributed subpopulations within an overall population. They are commonly used in statistical modeling, pattern recognition, and machine learning for clustering and density estimation. Here's a detailed explanation of GMMs and their training process:

Gaussian Mixture Model

A GMM assumes that the data points are generated from a mixture of several Gaussian distributions, each with its own mean and covariance. The probability density function (PDF) of a GMM is a weighted sum of the PDFs of multiple Gaussian components:

Uses EM to train

Applications of GMMs

- Clustering: Grouping data into clusters, where each cluster is modeled by a Gaussian distribution.
- Density Estimation: Estimating the underlying probability distribution of a dataset.
- Anomaly Detection: Identifying data points that do not fit the learned model well.
- Image and Speech Processing: Modeling and classifying different segments or features.

Loss functions

The key takeaway is that the loss function is a measurable way to gauge the performance and accuracy of a machine learning model. In this case, the loss function acts as a guide for the learning process within a model or machine learning algorithm.
The role of the loss function is crucial in the training of machine learning models and includes the following:

- Performance measurement: Loss functions offer a clear metric to evaluate a model's performance by quantifying the difference between predictions and actual results.
- Direction for improvement: Loss functions guide model improvement by directing the algorithm to adjust parameters(weights) iteratively to reduce loss and improve predictions.
- Balancing bias and variance: Effective loss functions help balance model bias (oversimplification) and variance (overfitting), essential for the model's generalization to new data.
- Influencing model behavior: Certain loss functions can affect the model's behavior, such as being more robust against data outliers or prioritizing specific types of errors.

Let's explore the roles of particular loss functions in later sections and build a detailed intuition and understanding of the loss function.

Source: https://www.datacamp.com/tutorial/loss-function-in-machine-learning#

Loss Function	Applicability to Classification	Applicability to Regression
Mean Square Error (MSE) / L2 Loss	✗	✓
Mean Absolute Error (MAE) / L1 Loss	✗	✓
Binary Cross-Entropy Loss / Log Loss	✓	✗
Categorical Cross-Entropy Loss	✓	✗
Hinge Loss	✓	✗
Huber Loss / Smooth Mean Absolute Error	✗	✓
Log Loss	✓	✗

More:

- https://neptune.ai/blog/pytorch-loss-functions
- https://pytorch.org/docs/stable/nn.html#loss-functions

Log-loss function

Log-loss and cross-entropy loss:
https://chatgpt.com/share/fdc27e3c-c702-4e24-9e26-8890cdfdba69?oai-dm=1

Log loss function: https://scikit-learn.org/stable/modules/model_evaluation.html#log-loss

3.3.2.12. Log loss

Log loss, also called logistic regression loss or cross-entropy loss, is defined on probability estimates. It is commonly used in (multinomial) logistic regression and neural networks, as well as in some variants of expectation-maximization, and can be used to evaluate the probability outputs (`predict_proba`) of a classifier instead of its discrete predictions.

For binary classification with a true label $y \in \{0, 1\}$ and a probability estimate $p = \Pr(y = 1)$, the log loss per sample is the negative log-likelihood of the classifier given the true label:

$$L_{\log}(y, p) = -\log \Pr(y|p) = -(y \log(p) + (1 - y) \log(1 - p))$$

This extends to the multiclass case as follows. Let the true labels for a set of samples be encoded as a 1-of-K binary indicator matrix Y, i.e., $y_{i,k} = 1$ if sample i has label k taken from a set of K labels. Let P be a matrix of probability estimates, with $p_{i,k} = \Pr(y_{i,k} = 1)$. Then the log loss of the whole set is

$$L_{\log}(Y, P) = -\log \Pr(Y|P) = -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{k=0}^{K-1} y_{i,k} \log p_{i,k}$$

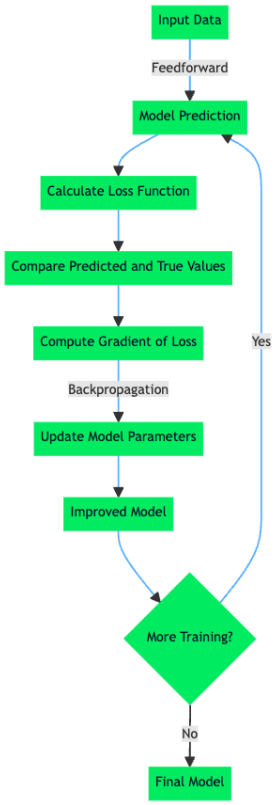
To see how this generalizes the binary log loss given above, note that in the binary case, $p_{i,0} = 1 - p_{i,1}$ and $y_{i,0} = 1 - y_{i,1}$, so expanding the inner sum over $y_{i,k} \in \{0, 1\}$ gives the binary log loss.

The `log_loss` function computes log loss given a list of ground-truth labels and a probability matrix, as returned by an estimator's `predict_proba` method.

```
>>> from sklearn.metrics import log_loss
>>> y_true = [0, 0, 1, 1]
>>> y_pred = [[.9, .1], [.8, .2], [.3, .7], [.01, .99]]
>>> log_loss(y_true, y_pred)
0.1738...
```

The first `[.9, .1]` in `y_pred` denotes 90% probability that the first sample has label 0. The log loss is non-negative.

ML Training Process



Autoencoder and Variational Autoencoder

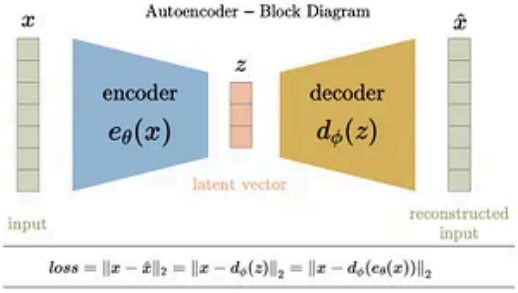
Cheat Sheet – Autoencoder & Variational Autoencoder

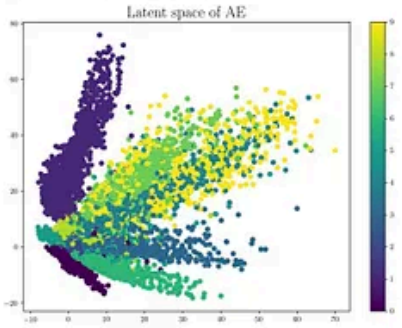
Context – Data Compression

- Data compression is an essential phase in training a network. The idea is to compress the data so that the same amount of information can be represented by fewer bits.

Auto Encoder (AE)

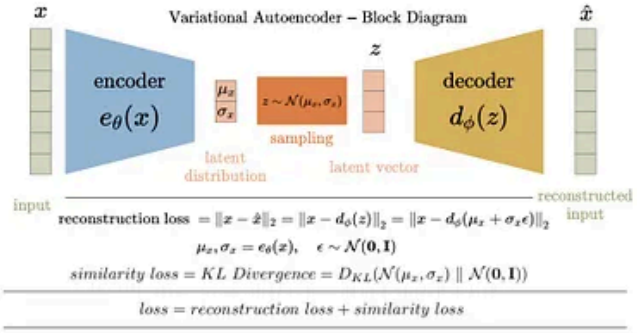
- Autoencoder is used to learn efficient embeddings of unlabeled data for a given network configuration. It consists of two parts, an **encoder**, and a **decoder**.
- The encoder compresses the data from a higher-dimensional space to a lower-dimensional space (also called the latent space),while the decoder converts the latent space back to higher-dimensional space.
- The entire encoder-decoder architecture is collectively trained on the loss function which encourages that the input is reconstructed at the output. Hence the loss function is the mean squared error between the encoder input and the decoder output.
- The latent variable is not regularized. Picking a random latent variable will generate garbage output.
- Latent variable is deterministic values and the space lacks the generative capability

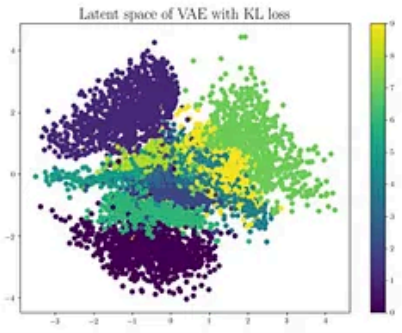




Variational Auto Encoder (VAE)

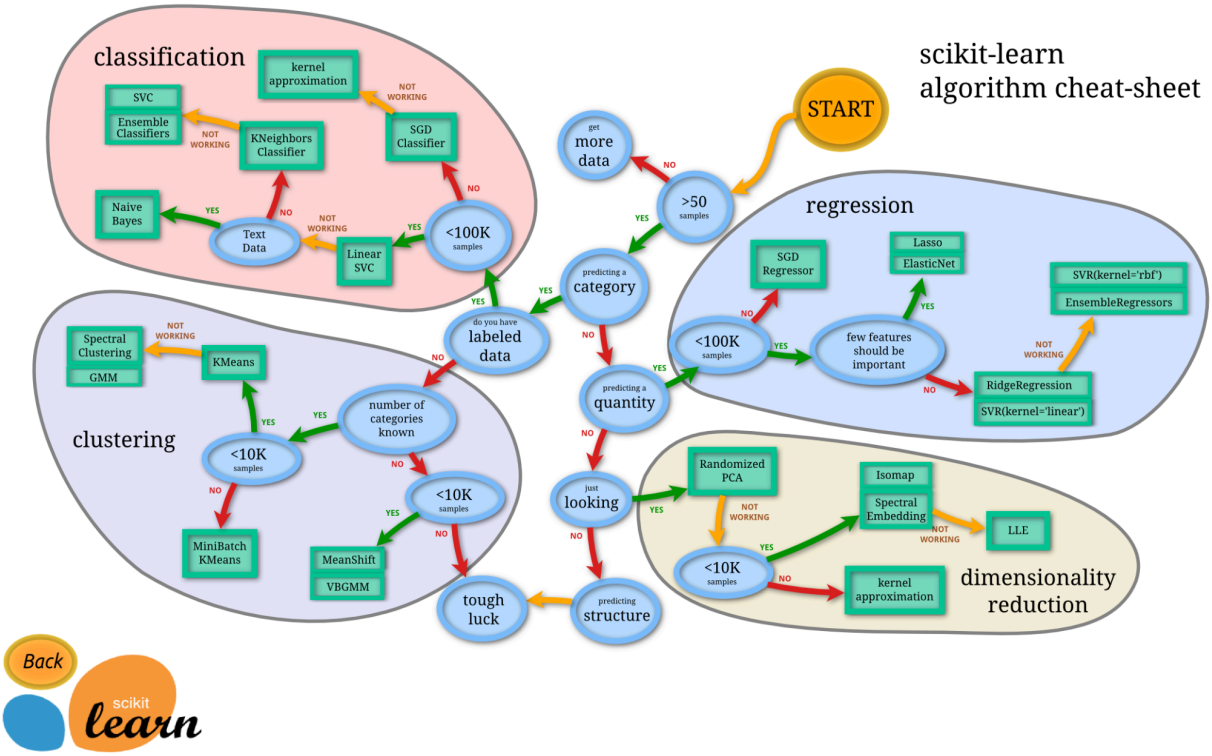
- Variational autoencoder addresses the issue of non-regularized latent space in autoencoder and provides the generative capability to the entire space.
- Instead of outputting the vectors in the latent space, the encoder of VAE outputs parameters of a pre-defined distribution in the latent space for every input.
- The VAE then imposes a constraint on this latent distribution forcing it to be a normal distribution.
- The latent variable in the compressed form is mean and variance
- The training loss of VAE is defined as the sum of the reconstruction loss and the similarity loss (the KL divergence between the unit gaussian and decoder output distribution.
- The latent variable is smooth and continuous i.e., random values of latent variable generates meaningful output at the decoder, hence the latent space has generative capabilities.
- The input of the decoder is sampled from a gaussian with mean/variance of the output of encoder.

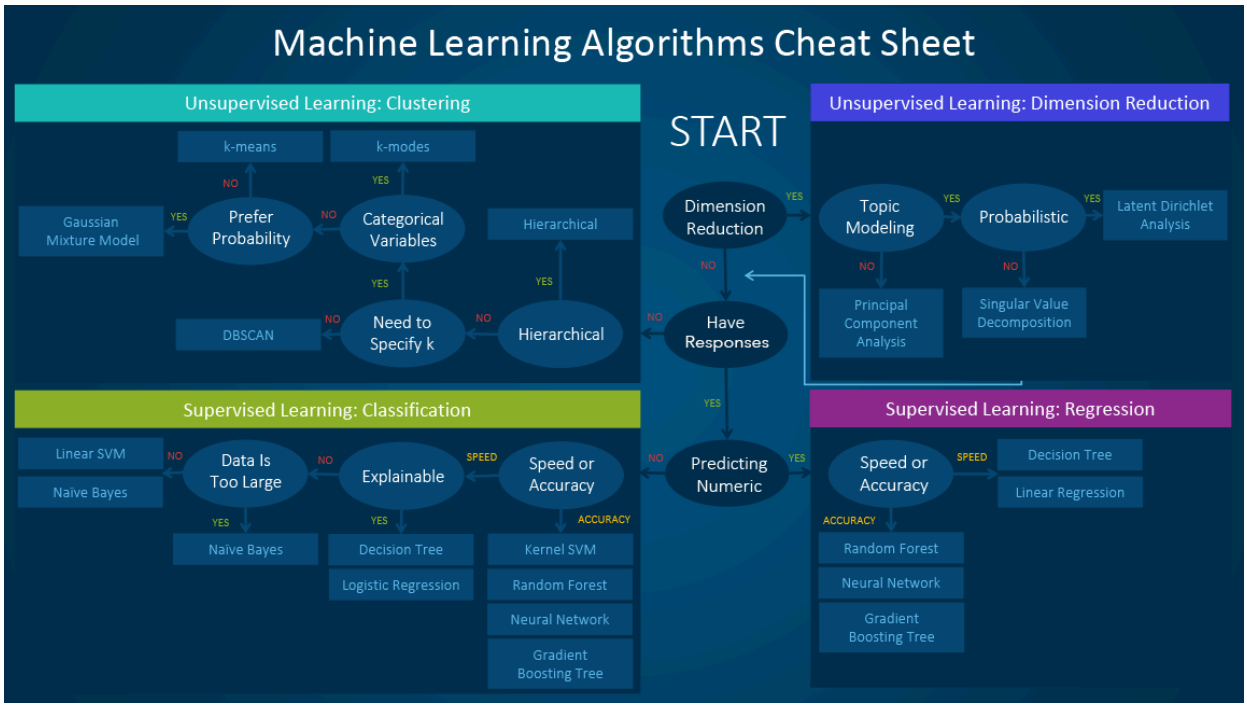




Source: https://www.cheatsheets.aqeel-anwar.com Tutorial: Click here

Model Search Diagrams





Types of Machine Learning (ML) Algorithms

This section provides an overview of the most popular types of machine learning. If you're familiar with these categories and want to move on to discussing specific algorithms, you can skip this section and go to "When to use specific algorithms" below.

Supervised learning

Supervised learning algorithms make predictions based on a set of examples. For example, historical sales can be used to estimate future prices. With supervised learning, you have an input variable that consists of labeled training data and a desired output variable. You use an algorithm to analyze the training data to learn the function that maps the input to the output. This inferred function maps new, unknown examples by generalizing from the training data to anticipate results in unseen situations.

- **Classification:** When the data are being used to predict a categorical variable, supervised learning is also called classification. This is the case when assigning a label or indicator, either dog or cat to an image. When there are only two labels, this is called binary classification. When there are more than two categories, the problems are called multi-class classification.
- **Regression:** When predicting continuous values, the problems become a regression problem.
- **Forecasting:** This is the process of making predictions about the future based on past and present data. It is most commonly used to analyze trends. A common example might be an estimation of the next year sales based on the sales of the current year and previous years.

Semi-supervised learning

The challenge with supervised learning is that labeling data can be expensive and time-consuming. If labels are limited, you can use unlabeled examples to enhance supervised learning. Because the machine is not fully supervised in this case, we say the machine is semi-supervised. With semi-supervised learning, you use unlabeled examples with a small amount of labeled data to improve the learning accuracy.

Unsupervised learning

When performing unsupervised learning, the machine is presented with totally unlabeled data. It is asked to discover the intrinsic patterns that underlie the data, such as a clustering structure, a low-dimensional manifold, or a sparse tree and graph.

- **Clustering:** Grouping a set of data examples so that examples in one group (or one cluster) are more similar (according to some criteria) than those in other groups. This is often used to segment the whole dataset into several groups. Analysis can be performed in each group to help users to find intrinsic patterns.
- **Dimension reduction:** Reducing the number of variables under consideration. In many applications, the raw data have very high dimensional features and some features are redundant or irrelevant to the task. Reducing the dimensionality helps to find the true, latent relationship.

Reinforcement learning

Reinforcement learning is another branch of machine learning which is mainly utilized for sequential decision-making problems. In this type of machine learning, unlike supervised and unsupervised learning, we do not need to have any data in advance; instead, the learning agent interacts with an environment and learns the optimal policy on the fly based on the feedback it receives from that environment. Specifically, in each time step, an agent observes the environment's state, chooses an action, and observes the feedback it receives from the environment. The feedback from an agent's action has many important components. One

component is the resulting state of the environment after the agent has acted on it. Another component is the reward (or punishment) that the agent receives from performing that particular action in that particular state. The reward is carefully chosen to align with the objective for which we are training the agent. Using the state and reward, the agent updates its decision-making policy to optimize its long-term reward. With the recent advancements of deep learning, reinforcement learning gained significant attention since it demonstrated striking performances in a wide range of applications such as games, robotics, and control. To see reinforcement learning models such as Deep-Q and Fitted-Q networks in action, check out this article.

Resources

- https://medium.com/swlh/cheat-sheets-for-machine-learning-interview-topics-51c2bc2bab4f
- https://github.com/jman4162/machine-learning-review
- https://blogs.sas.com/content/subconsciousmusings/2020/12/09/machine-learning-algorithm-use/
-