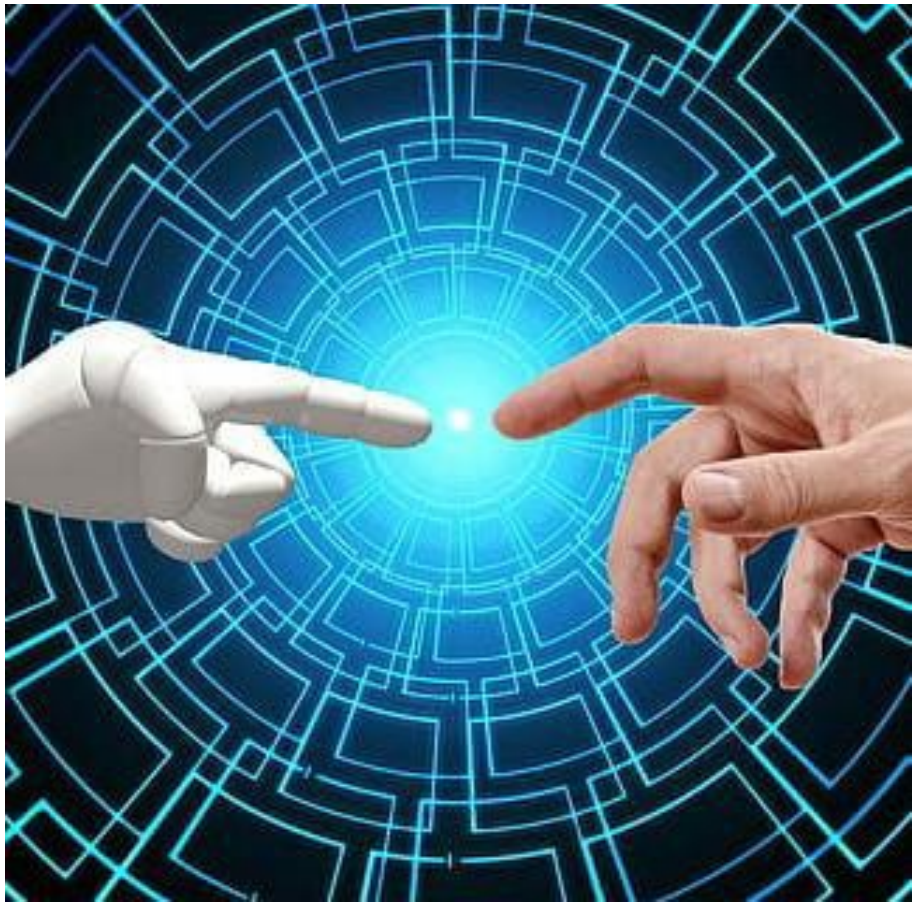
Artificial Intelligence (AI) for Engineering

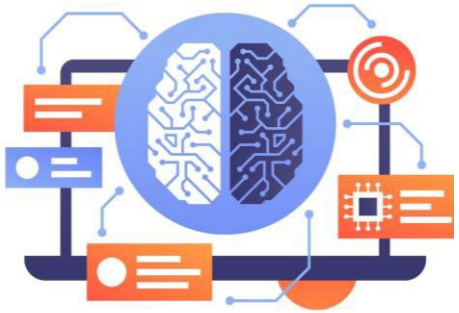


COS40007

Dr. Afzal Azeem Chowdhary

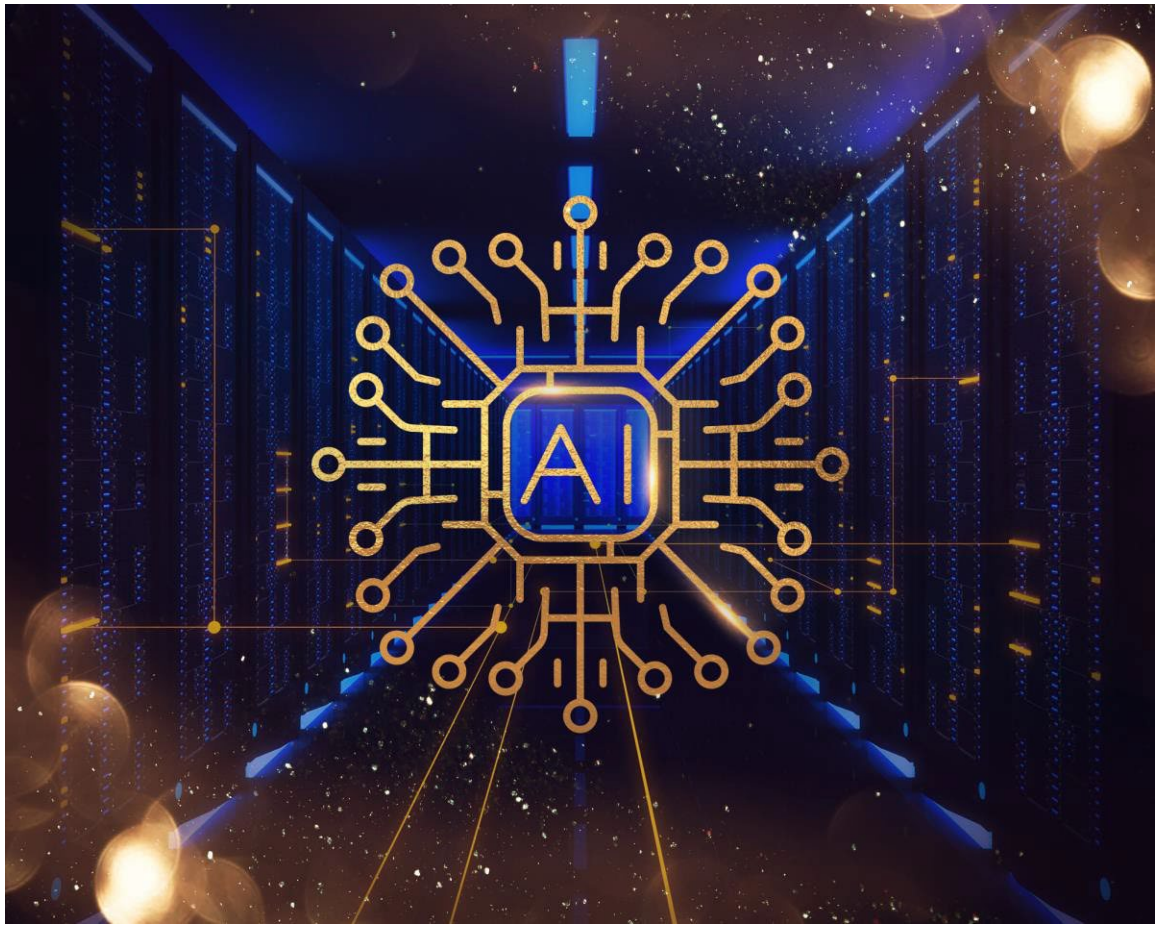
Lecturer, SoCET, Swinburne University of Technology

Seminar 10: 13th May 2025



Overview

- Ensemble Learning
- Working with limited data
- Working with unstructured
- Working with unlabelled data



At the end of this you should be able to

- Understand about Ensemble Learning
- Understand about how to develop AI with limited data
- Understand about developing AI with unlabelled data
- Understand about developing AI with unstructured data

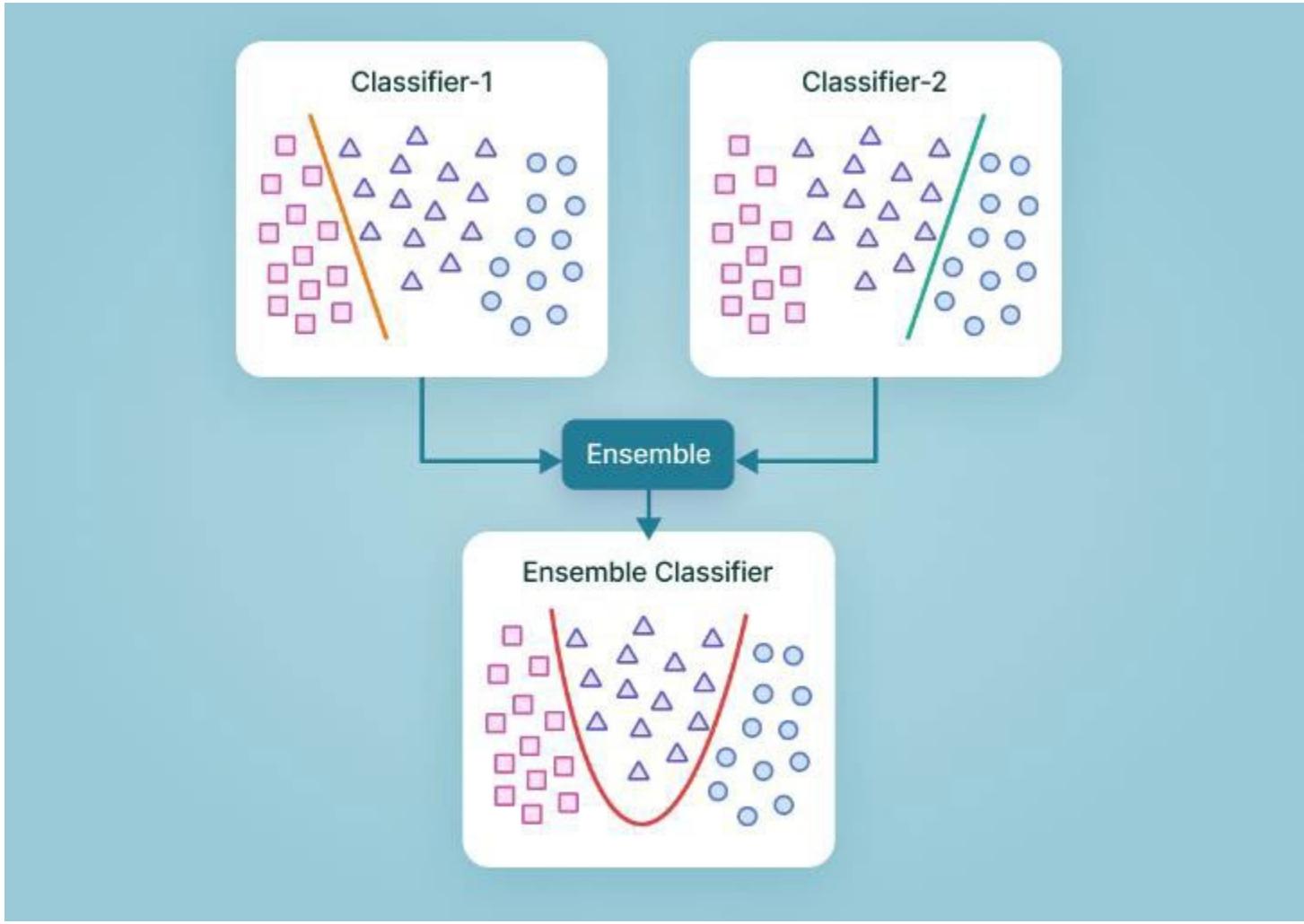
Ensemble Learning

Ensemble learning

- Ensemble learning combines multiple learners to improve predictive performance.
- It has been adopted in response to issues resulting from limited datasets.
- A machine learning technique that aggregates two or more learners (*e.g. regression models, neural networks*) to produce better predictions
- An ensemble model combines several individual models to produce more accurate predictions than a single model alone.
- **Example:** a model may be well adapted to differentiate between cats and dogs, but not so much when distinguishing between dogs and wolves. A second model can accurately differentiate between dogs and wolves while producing wrong predictions on the “cat” class.

Ensemble Learning

If we have a linear classifier and try to tackle a problem with a parabolic (polynomial) decision boundary, one linear classifier obviously cannot do the job well. However, an ensemble of multiple linear classifiers can generate any polynomial decision boundary.



Example of Ensemble learning

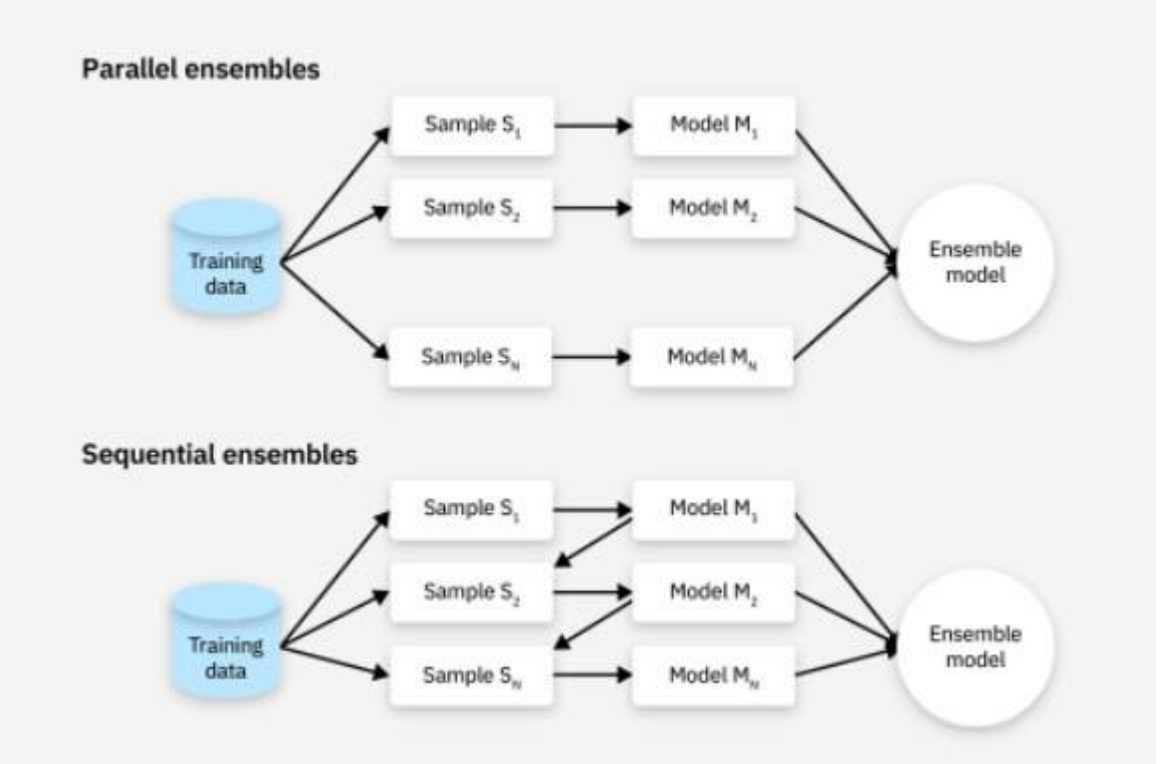
- Cardiovascular disease detection from X-ray and CT scans
- Landslide Detection and Scene Classification
- Detecting Credit Card Fraud and Impression Fraud
- Speech emotion recognition, especially in the case of multi-lingual environments

Some advantages of ensemble learning include:

- *Improved accuracy:* By averaging or combining the predictions from multiple models, ensembles often outperform individual models.
- *Reduced overfitting:* Ensemble methods help reduce overfitting by smoothing out noisy predictions.
- *Model diversity:* Ensembles make use of multiple algorithms or variations of the same algorithm, which can capture different aspects of the data.

Types of Ensemble learning

- Parallel methods train each base learner apart from the others. Per its name, parallel ensembles train base learners in parallel and independently of one another.
- Sequential methods train a new base learner to minimise errors made by the previous model trained in the preceding step. In other words, sequential methods construct base models sequentially in stages.



Combining multiple learners

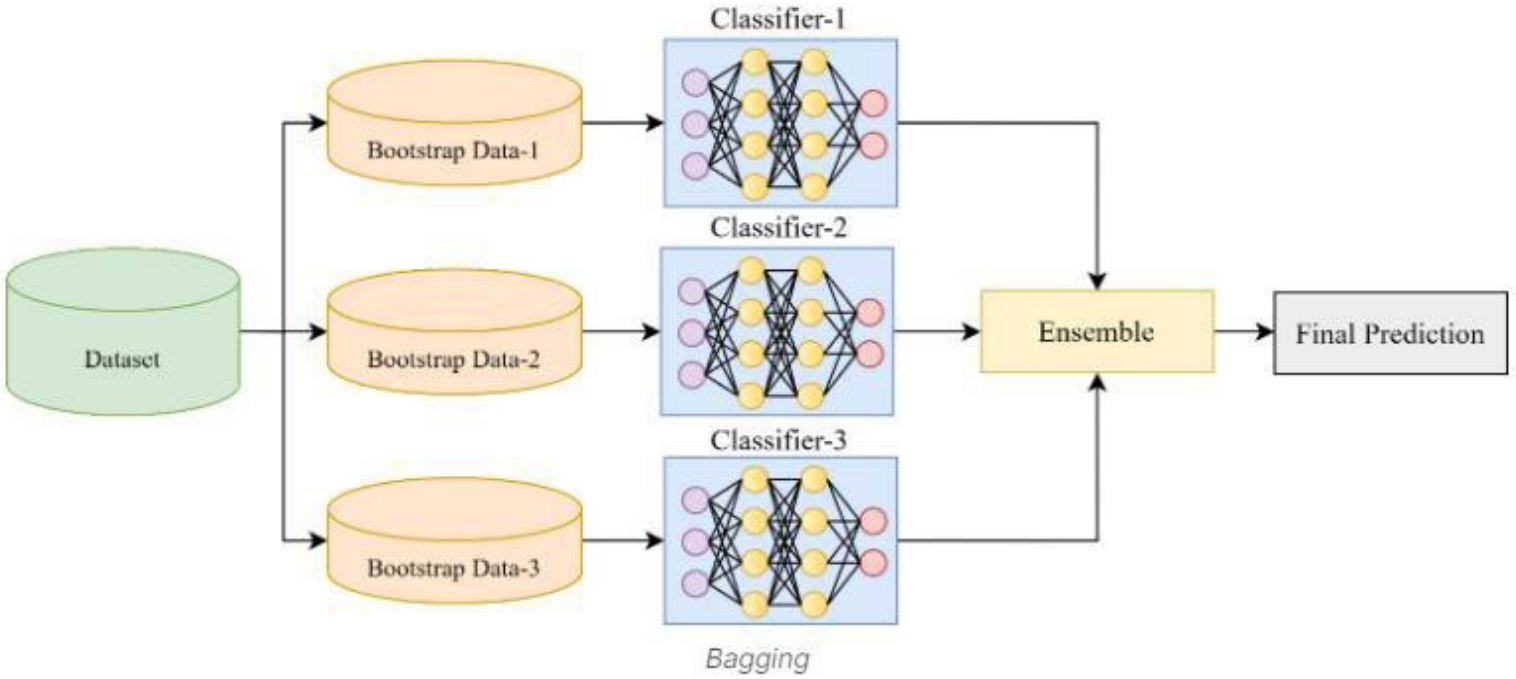
- *Stacking:* Use separate machine learning algorithms to train an ensemble learner from the base learners.
- *Majority Voting:* Considers each base learner's prediction for a given data instance and outputs a final prediction determined by whatever most learners predict.
- *Weighted Majority Voting:* an extension of voting technique that gives greater weight to certain learners' predictions over others

Bootstrapping

- Where the dataset available is small (for example, in the biomedical domain, where acquiring labelled medical data is costly), we can use a bootstrapping ensemble strategy.
- We train different classifiers using various “bootstrap samples” of data.
- We create several subsets of a single dataset using replacement. This means that the same data sample may be present in more than one subset.

Ensemble Learning techniques: Bagging

- Bagging: Bagging is a homogenous parallel method (also known as bootstrap aggregating). It uses modified replicates of a given training data set to train multiple base learners with the same training algorithm
- Scikit-learn's ensemble module in Python contains functions for implementing bagging, such as BaggingClassifier.
- Random forest is an extension of bagging that explicitly denotes the use of bagging to construct ensembles of randomised decision trees.



Ensemble Learning techniques: Bagging

How Bagging Works:

1. **Bootstrap Sampling:** Randomly sample data from the training set with replacement to create multiple subsets (bootstrap samples). Some data points may be repeated, while others might be omitted in each sample.
2. **Train Models:** A separate model (usually a weak learner like a decision tree) is trained on each bootstrap sample.
3. **Aggregate Predictions:** For classification tasks, the predictions from all the models are combined using majority voting. For regression tasks, the models' predictions are averaged.

Advantages of Bagging:

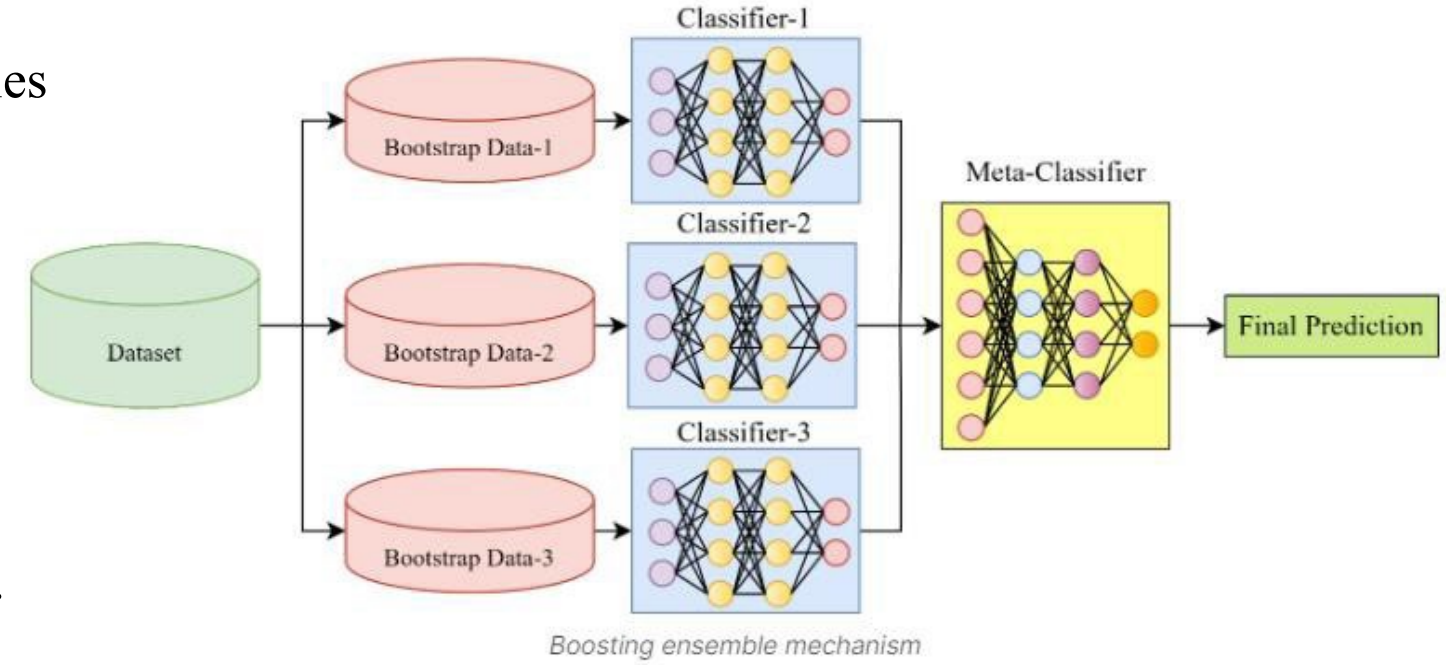
- Reduces variance: By averaging the predictions of multiple models, bagging reduces the model's variance, making it less likely to overfit.
- Parallelizable: Since each model is trained independently, bagging can be parallelized easily.

Disadvantages of Bagging:

- Less effective on bias: Bagging primarily reduces variance but does not significantly reduce bias if the individual models are weak.

Ensemble Learning techniques: Stacking

- Stacking, or stacked generalisation, is a heterogeneous parallel method that exemplifies what is known as meta-learning.
- Meta-learning consists of training a meta-learner from the output of multiple base learners.
- Stacking specifically trains several base learners from the same dataset using a different training algorithm for each learner.
- Each base learner makes predictions on an unseen dataset. These first model predictions are then compiled and used to train a final model, the **Meta-Model**



Boosting ensemble mechanism

sklearn ensemble module in Python provides various functions for implementing stacking techniques

Ensemble Learning techniques: Stacking

The idea behind stacking is to leverage the strengths of several models by training a meta-model (often called a second-level model) that learns to make predictions based on the outputs of the base models.

How Stacking Works:

1. Train multiple base models (e.g., decision trees, logistic regression, SVMs) on the training data.
2. The predictions from these base models are fed into a meta-model (typically a more complex model like a neural network or linear regression).
3. The meta-model learns to combine the predictions of the base models and outputs the final prediction.

Advantages of Stacking:

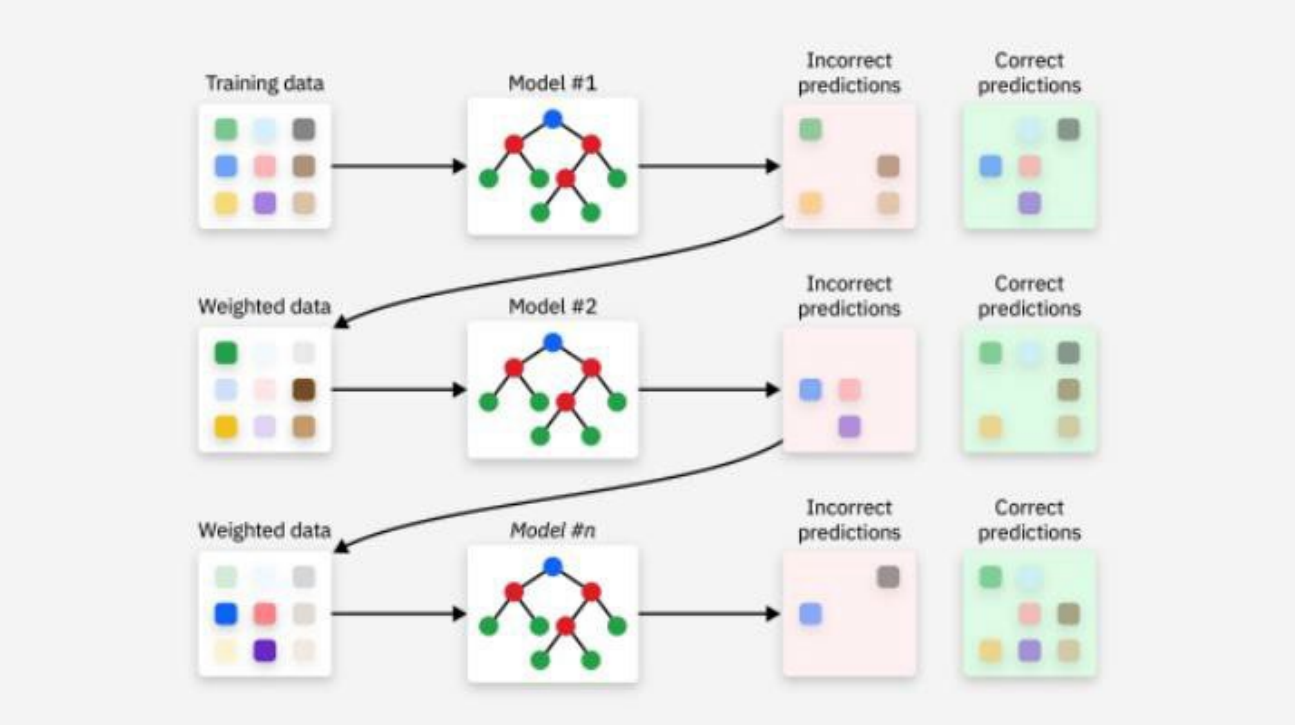
- Combines models with different strengths to improve overall performance.
- Often leads to better performance than using any single model.

Disadvantages of Stacking:

- Increased complexity, making the final model harder to explain and interpret, which might deter its adoption in some contexts.
- Increase computational time and resource requirements, especially when dealing with large datasets and complex base models.
- Effectiveness of stacking hinges on the diversity of the base models, and if they are highly correlated, the gains from stacking might be limited.

Ensemble Learning techniques: Boosting

- Boosting algorithms are a sequential ensemble method.
- Boosting trains a learner on an initial dataset, d. The resultant learner is typically weak, misclassifying many samples in the dataset.
- Boosting then samples instances from the initial dataset to create a new dataset (d_2)
- Boosting prioritises misclassified data instances from the first model or learner.
- A new learner is trained on this new dataset d_2. Then, a third dataset (d_3) is compiled from d_1 and d_2, prioritising the second learner's misclassified samples and instances in which d_1 and d_2 disagree.
- The process repeats n times to produce n learners. Boosting then combines and weights all the learners together to produce final predictions



Ensemble Learning techniques: Boosting

How Boosting Works:

1. **Train Weak Learners Sequentially**: Models are trained one after another, and each model pays more attention to the data points that were misclassified by previous models.
2. **Adjust Weights**: In each iteration, the weights of incorrectly predicted samples are increased, so subsequent models focus more on those difficult-to-predict samples.
3. **Aggregate Predictions**: The final prediction is made by taking a weighted average or sum of the individual model predictions.

Advantages of Boosting:

- Reduces bias and variance: Boosting reduces both the bias and variance of the model, resulting in highly accurate predictions.
- Works well with weak learners: Even weak models, like shallow decision trees, can be combined through boosting to create a powerful predictor.

Disadvantages of Boosting:

- Sensitive to outliers: Since boosting focuses on correcting errors, it can place too much emphasis on noisy data points or outliers.
- Computationally expensive: Boosting is a sequential process, making it slower than parallelizable methods like bagging.

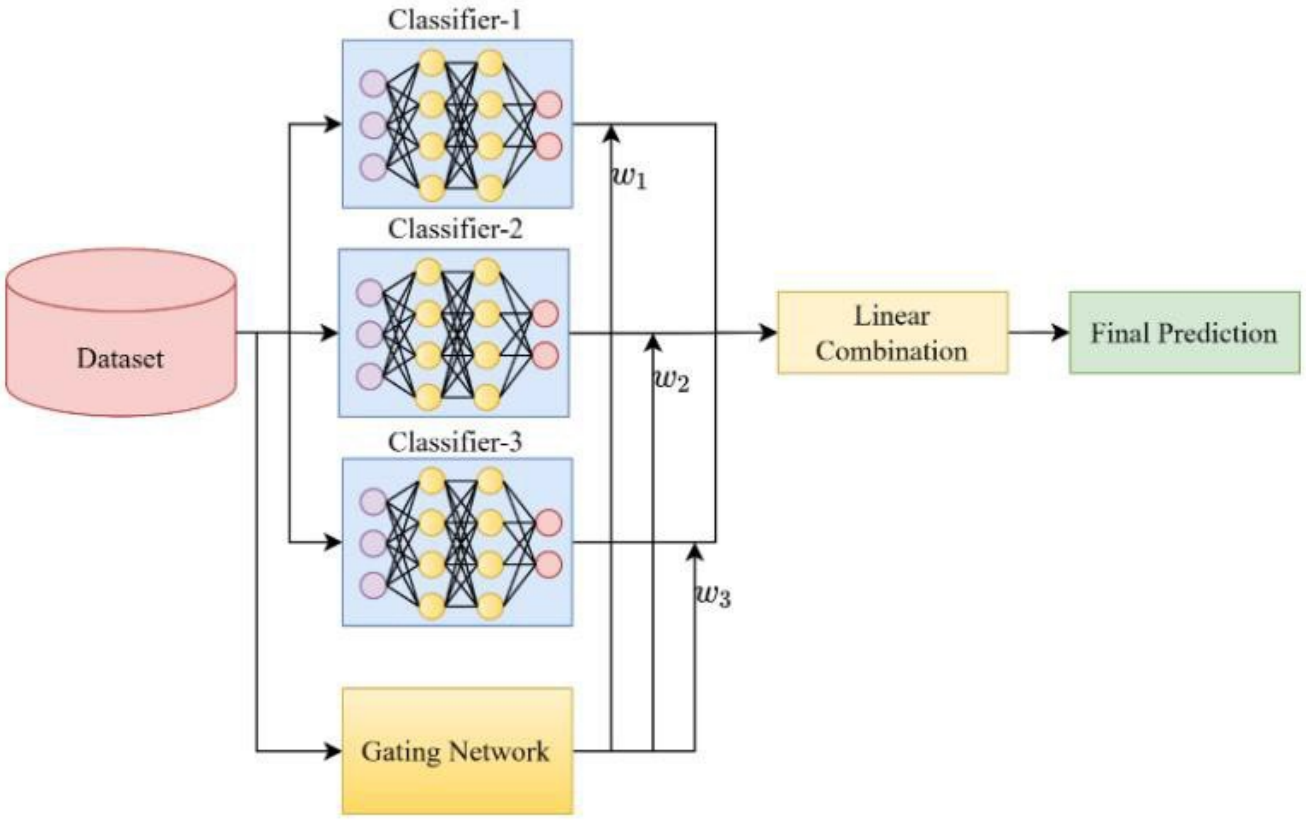
Boosting: Adaptive Boosting (AdaBoost)

- *Adaptive boosting (AdaBoost)* weights model errors. When creating a new iteration of a dataset for training the next learner, AdaBoost adds weights to the previous learner's misclassified samples, causing the next learner to prioritise those misclassified samples.
- *Gradient Boosting* uses residual errors when training new learners. Rather than weight misclassified samples, gradient boosting uses residual errors from a previous model to set target predictions for the next model. In this way, it attempts to close the gap of error left by one mode

Extreme Gradient Boosting (XGBoost), an open-source library, provides code for implementing gradient boosting in Python.

Mixture of Experts

- The “Mixture of Experts” genre of ensemble trains several classifiers, the outputs of which are ensembles using a generalised linear rule.
- The weights assigned to these combinations are further determined by a “Gating Network,” a trainable model and usually a neural network.
- Such an ensemble technique is usually used when different classifiers are trained on other parts of the feature space.



Voting

In voting, multiple models are trained independently on the same dataset, and their predictions are combined by voting in the case of classification tasks, or by averaging in the case of regression tasks. This is one of the simplest ensemble methods and can be classified into two types: hard voting and soft voting.

- **Hard Voting**: In classification tasks, the final ensemble prediction is determined by selecting the class that receives the most votes from the base models' predictions. This is often referred to as "hard voting."
- **Soft Voting**: In regression tasks, the final prediction is typically obtained by averaging the predictions of the base models. This is also known as "soft voting."

Example:

You can train three models (e.g., logistic regression, decision tree, and random forest) on a dataset and combine their predictions by hard voting. The final prediction is based on the majority vote.

Advantages of Voting:

- Simple to implement and interpret.
- Can improve accuracy by combining diverse models.
- Works well when the base models are fairly strong and complementary.

Majority Voting

- An odd number of contributing classifiers are chosen, and for each sample, the predictions from the classifiers are computed.
- The class that gets most of the class from the classifier pool is deemed the ensemble's predicted class.
- Such a method works well for binary classification problems, where there are only two candidates for which the classifiers can vote. However, it fails for a problem with many classes since many cases arise.

Max Rule

- Relies on the probability distributions generated by each classifier
- This method employs the concept of "confidence in prediction" of the classifiers
- For a predicted class by a classifier, the corresponding confidence score is checked.
- The class prediction of the classifier that predicts with the highest confidence score is deemed the prediction of the ensemble framework

Key Differences Between Bagging and Voting:

Aspect	Bagging	Voting
Dataset Subsets	Models are trained on **different subsets** of data (via bootstrapping)	Models are trained on the **same dataset**
Model Diversity	Typically uses the **same type of model** (e.g., decision trees)	Often uses **different types of models** (e.g., tree, SVM, logistic regression)
Training Process	Uses **bootstrapping** to generate diverse training data	No bootstrapping, models use the same full dataset
Aggregation Method	Uses **majority voting** (classification) or **averaging** (regression) for final prediction	Uses **majority voting** (classification) or **averaging** (regression)
Purpose	Primarily aims to **reduce variance** and overfitting by averaging models trained on different datasets	Aims to **leverage the strengths** of different models to improve generalization
Examples	Random Forest (multiple decision trees)	Combining SVM, decision tree, and logistic regression

Example: Heart Attack Prediction

```
import pandas as pd import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score,accuracy_score
from sklearn.preprocessing import StandardScaler

# machine learning
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier

#seed
seed = 40

#read input
df = pd.read_csv("heart.csv")

#target
target = df["output"]

# getting scaled train data
train = df.drop("output", axis=1)
scaled_train = StandardScaler().fit_transform(train)
```

https://www.kaggle.com/datasets/iamsouravbanerjee/heart-attack-prediction-dataset

Example: Heart Attack Prediction

```
# Splitting the data into training and validation
X_train, X_test, y_train, y_test = train_test_split(
train, target, test_size=0.20, random_state=seed
)

# building the all models
model_1 = RandomForestClassifier(random_state=seed)
model_2 = LogisticRegression(random_state=seed, max_iter=1000)
model_3 = SGDClassifier(random_state=seed)

# training
model_1.fit(X_train, y_train)
model_2.fit(X_train, y_train)
model_3.fit(X_train, y_train)

# predicting
pred_1 = model_1.predict(X_test)
pred_2 = model_2.predict(X_test)
pred_3 = model_3.predict(X_test)

# averaging
pred_final = np.round((pred_1 + pred_2 + pred_3) / 3)

# evaluation
accuracy = round(accuracy_score(y_test, pred_final) * 100, 3)
auc = round(roc_auc_score(y_test, pred_final), 3)

print(f" Accuracy: {accuracy}%")
print(f" AUC score: {auc}")
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

Example: Weighted average

	model_1	model_2	model_3
Weightage	30%	60%	10%

```
pred_final = np.round(0.3*pred_1 + 0.6*pred_2 + 0.1*pred_3)

# evaluation
accuracy = round(accuracy_score(y_test, pred_final) * 100, 3)
auc = round(roc_auc_score(y_test, pred_final), 3)

print(f" Accuracy: {accuracy}%")
print(f" AUC score: {auc}")
```

Example: Bagging

```
from sklearn.ensemble import BaggingClassifier

# bagging
clf = BaggingClassifier(LogisticRegression(random_state=seed,
        max_iter=2000),
        n_estimators=20,
        random_state=seed,
)

# training
clf.fit(X_train, y_train)

# prediction
prediction = clf.predict(X_test)

# evaluation
accuracy = round(accuracy_score(y_test, prediction) * 100, 3)
auc = round(roc_auc_score(y_test, prediction), 3)

print(f" Accuracy: {accuracy}%") print(f" AUC score: {auc}")
```

Example: Boosting

```
# boosting
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(random_state=seed)

# training
clf.fit(X_train, y_train)

# prediction
prediction = clf.predict(X_test)

# evaluation
accuracy = round(accuracy_score(y_test, prediction) * 100, 3)
auc = round(roc_auc_score(y_test, prediction), 3)

print(f" Accuracy: {accuracy}%")
print(f" AUC score: {auc}")
```

Example: Stacking

```
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier

# building the model
model_5 = AdaBoostClassifier(random_state=seed)

estimators = [("rf", model_1), ("lr", model_2), ("ada", model_5)]

# building the model
final_estimator = GradientBoostingClassifier(random_state=seed)

# Stacking Classifier
clf = StackingClassifier(estimators=estimators,
final_estimator=final_estimator)

# training
clf.fit(X_train, y_train)

# prediction
prediction = clf.predict(X_test)

# evaluation
accuracy = round(accuracy_score(y_test, prediction) * 100, 3)
auc = round(roc_auc_score(y_test, prediction), 3)

print(f" Accuracy: {accuracy}%")
print(f" AUC score: {auc}")
```

Example: Voting

```
from sklearn.ensemble import VotingClassifier
from sklearn.neighbors import KNeighborsClassifier

# building the model
model_4 = KNeighborsClassifier()
# voting classifier
final_model = VotingClassifier(
                estimators=[("rf", model_1), ("lr", model_2), ("knn", model_4)],
                voting="hard",
            )

# training
final_model.fit(X_train, y_train)


# prediction
prediction = final_model.predict(X_test)

# evaluation
accuracy = round(accuracy_score(y_test, prediction) * 100, 3)
auc = round(roc_auc_score(y_test, prediction), 3)

print(f" Accuracy: {accuracy}%")
print(f" AUC score: {auc}")
```

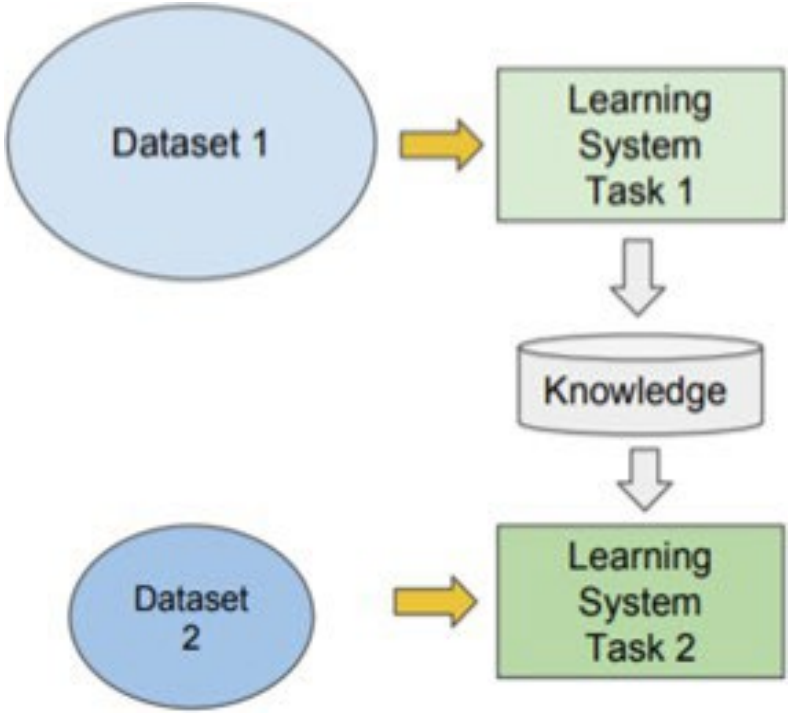
SWINBURNE UNIVERSITY OF TECHNOLOGY

Working with limited data

- More data, better model
- However, data annotation is an expensive job
- 14 million images. 20,000 categories. 25 Human Years to annotate!
- Reality: we have limited data to train a model
- Ways to deal with limited data
  - ✓ AWS Mechanical Turk (crowdsourcing labelling task)
  - ✓ CrowdFlower
  - ✓ Data Augmentation/Synthetic Generation

Inductive Transfer Learning

Learning new tasks using knowledge learned from other tasks



A **Domain** consists of two components: $D = \{\mathcal{X}, P(X)\}$

- Feature space: $\mathcal{X}$
- Marginal distribution: $P(X)$, $X = \{x_1, \cdots, x_n\}, x_i \in \mathcal{X}$

For a given domain **D**, a **Task** is defined by two components:

$$T = \{\mathcal{Y}, P(Y|X)\} = \{\mathcal{Y}, \eta\} \qquad Y = \{y_1, \cdots, y_n\}, y_i \in \mathcal{Y}$$

- A label space: $\mathcal{Y}$
- A predictive function η, learned from *feature vector/label* pairs, (x_i, y_i), $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$
- For each feature vector in the domain, η predicts its corresponding label: $\eta(x_i) = y_i$

Inductive Transfer Learning

How it Works

The process of inductive transfer learning involves two main steps:

1. Pre-training: A model is trained on a source task, where abundant data is available. This model learns a general representation of the data, capturing the underlying patterns and structures.
2. Fine-tuning: The pre-trained model is then fine-tuned on the target task, which may have less data available. The model's parameters are updated to better fit the target task, while still retaining the general knowledge learned from the source task.

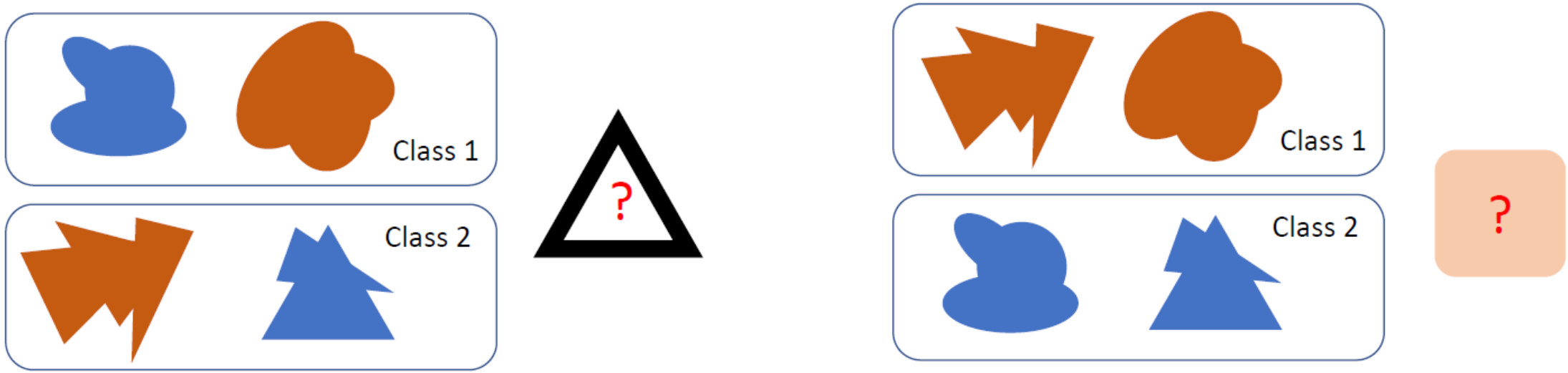
Use Cases

- Natural Language Processing (NLP): Models like BERT and GPT have been pre-trained on large text corpora and then fine-tuned for specific tasks such as sentiment analysis or question answering.
- Computer Vision: Models like ResNet and VGG have been pre-trained on large image datasets like ImageNet and then fine-tuned for tasks like object detection or image segmentation.
- Reinforcement Learning: Agents can be pre-trained in a simulated environment and then fine-tuned in the real world, reducing the amount of real-world interaction needed.

https://saturncloud.io/glossary/inductive-transfer-learning/

Few-shot Learning

- To classify new data after being given a few samples
- The conventional approach is to train the model using a dataset to perform classification
- Meta-learning is to 'train' the model to learn how to use datasets to perform classification (Learning to Learn)



https://www.geeksforgeeks.org/zero-shot-vs-one-shot-vs-few-shot-learning/#what-is-fewshot-learning

Working with un-labelled data: Semi-supervised Learning

Semi-supervised Learning

- Unsupervised learning doesn't work on the labelled data. So, the cost of labelling the data is saved. Works on large amounts of unlabeled data.
- Supervised learning by using a small amount of labelled dataset and classifying the dataset as accurately as possible.

Semi-supervised Learning

- Pick up the large unlabelled dataset.
- Label a small portion of the dataset.
- Put the unlabelled dataset into clusters using an Unsupervised Machine Learning algorithm.
- Build your model to use the labelled data to label and classify the rest of the unlabelled data.

  - **Internet Content Classification:** There are millions and millions of web pages. Labelling all the web pages is practically impossible. If you need to, semi-supervised machine learning helps classify web pages here.
  - **Audio/ Video analysis**: We have an overwhelming amount of audio and video files. Labelling them is a massive task, if not impossible. Semi-supervised learning comes in handy in audio/ video analysis.

Working with unstructured data

- Does not reside in traditional databases and data warehouses
- May have an internal structure but does not fit a relational data model
- Generated by both humans and machines
- Textual and multimedia content
- Machine-to-machine communication
- Examples include:
  - Personal messaging – email, instant messages, tweets, chat
  - Business documents – business reports, presentations, survey responses
  - Web content – web pages, blogs, wikis, audio files, photos, videos
  - Sensor output – satellite imagery, geolocation data, scanner transactions

Analysing unstructured data

For unstructured data to be useful it must be analysed to extract and expose the information it contains.

Different types of analysis are possible, such as:

- Entity analysis – people, organisations, objects and events, and the relationships between them
- Topic analysis – topics or themes and their relative importance
- Sentiment analysis – subjective view of a person to a particular topic
- Feature analysis – inherent characteristics that are significant for a particular analytical perspective (e.g. land coverage in satellite imagery)
- Many others

Learn, Practice and Enjoy the AI journey