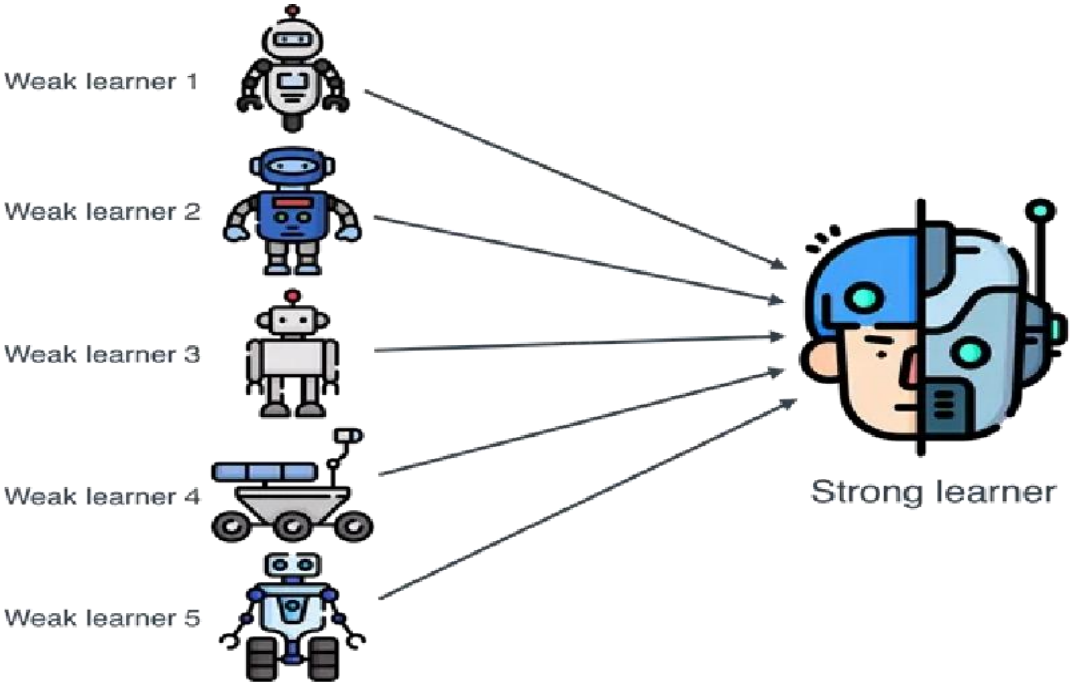
ENSEMBLE LEARNING

- Ensemble learning is a method where multiple models are combined instead of using just one. Even if individual models are weak, combining their results gives more accurate and reliable predictions.
- Uses multiple models together to improve overall accuracy.
- Reduces errors by balancing mistakes across models.
- Works on a simple idea similar to combining opinions from a group.



Why use ensemble learning?

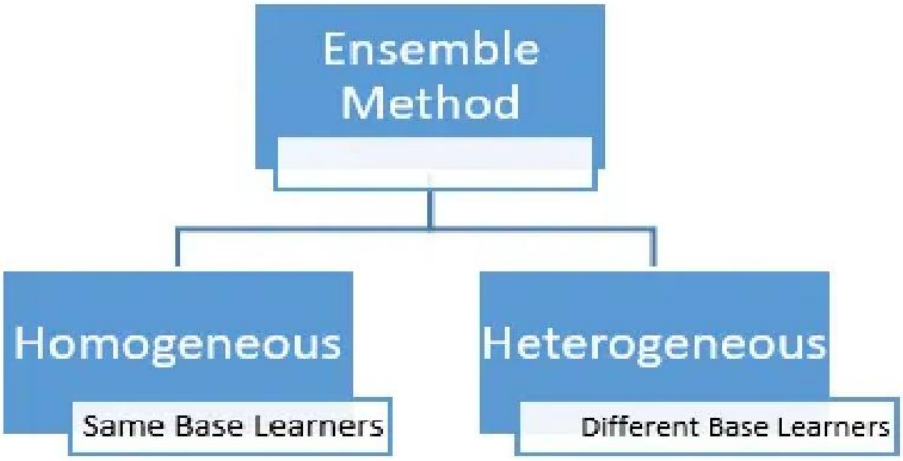
- The bias-variance problem
- One of the main reasons for using ensemble methods is to address the bias-variance trade-off, which is a fundamental challenge in machine learning:
- Bias refers to the error that comes from wrong assumptions in the model. High-bias models are too simple and miss important patterns in the data, leading to underfitting.
- Variance refers to the model's sensitivity to small changes in the training data. High variance models are too complex and learn not just the patterns but also the noise in the data, leading to overfitting.
- Ensemble learning helps manage this trade-off by combining multiple models.
- Better performance through diversity
- Another benefit of ensemble methods is that they can leverage the diversity of multiple models:
- Different perspectives: Various models can capture different aspects of the data.
- Error correction: When some models make mistakes, others can compensate.

- Noise reduction: By aggregating multiple predictions, random errors tend to cancel out.

Types of ensemble learning methods

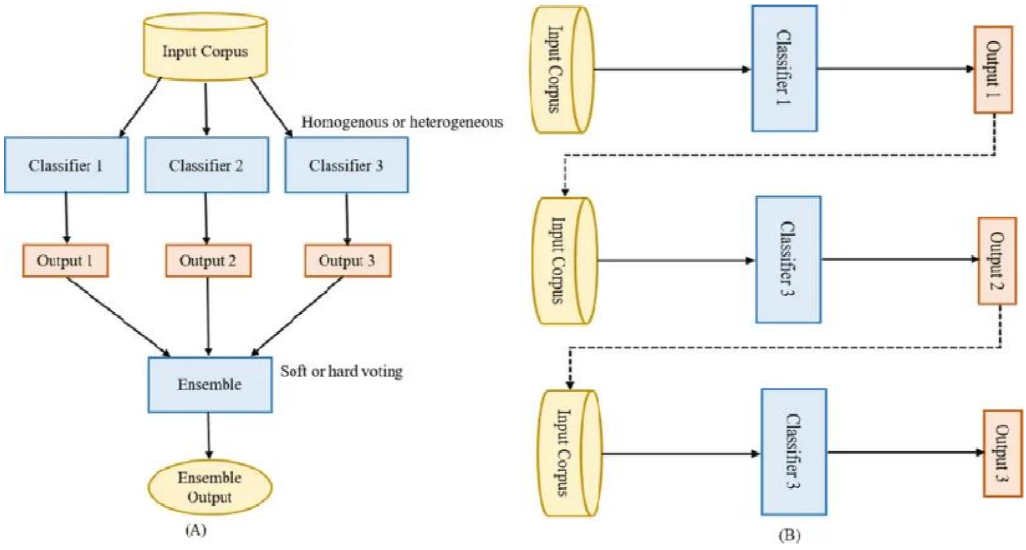
- Ensemble methods can be categorised in multiple ways. Here are the primary distinctions.

Based on model composition:



- **Homogeneous ensembles**: In homogeneous ensembles, all base models are of the same type but are trained on different subsets of data or with different parameters. For example, a Random Forest consists of multiple decision trees.
- **Heterogeneous ensembles**: In heterogeneous ensembles, the base models are of different types. For instance, you might combine a decision tree, a support vector machine, and a neural network.

Based on the learning approach:

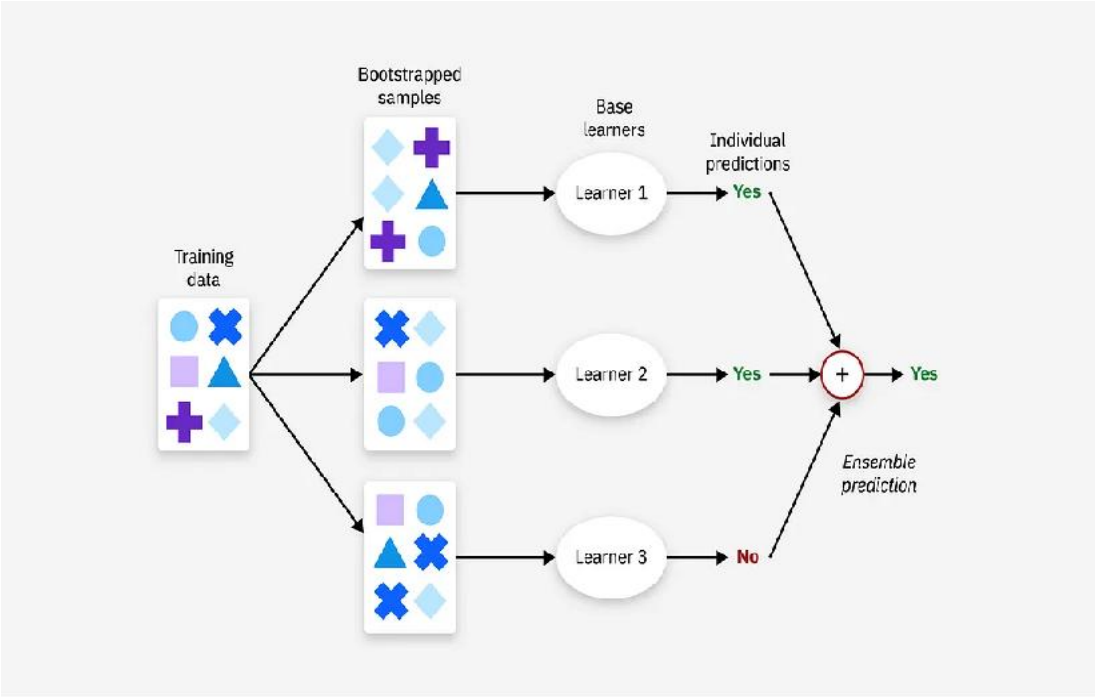


- **Parallel methods:** Parallel methods train all base models independently and combine their predictions at the end.
- **Sequential methods**: Sequential methods train models one after another, with each new model focusing on the mistakes made by previous models.

Common Ensemble Techniques

1. Bagging (Bootstrap Aggregating)

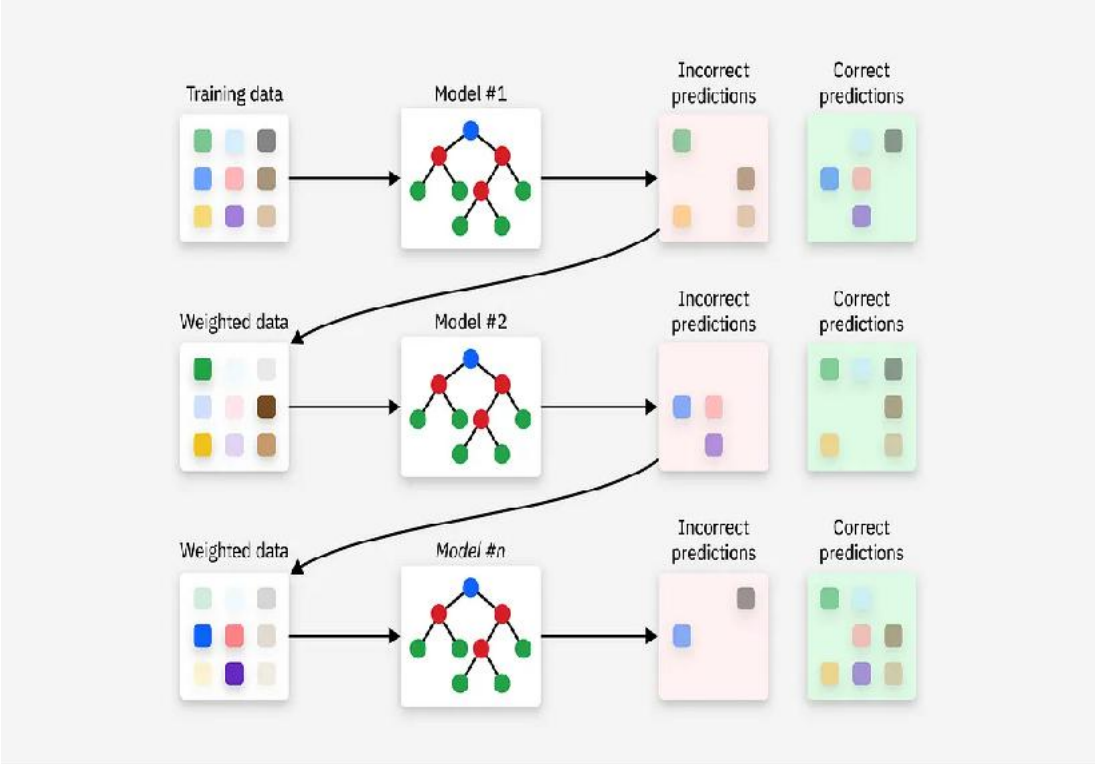
- Bagging involves training multiple versions of the same model on different random subsets of the training data, sampled with replacement (bootstrap sampling). The final prediction is typically the average (for regression) or majority vote (for classification) of all individual predictions.



- Example: Random Forest, which builds multiple decision trees on different data samples and aggregates their predictions.
- When to use bagging:
- Your base models have high variance (they overfit)
- Your dataset contains noise or outliers
- You need to improve stability and accuracy
- You have enough computational resources for parallel processing
- You're working with decision trees
- Your dataset is relatively small

2. Boosting

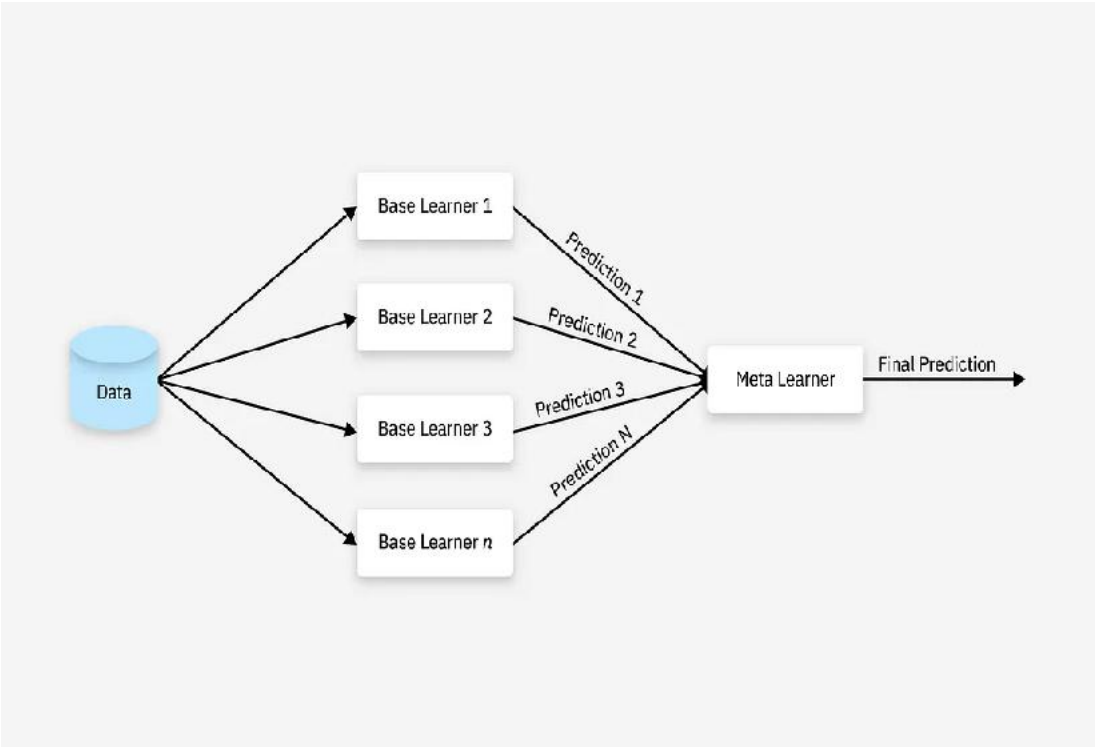
- Boosting is a sequential ensemble method that trains models one after another, with each new model focusing on the examples that previous models got wrong. It assigns higher weights to misclassified samples, forcing subsequent models to pay more attention to difficult cases.



- Examples: AdaBoost, Gradient Boosting, XGBoost, LightGBM
- When to use boosting:
- Your base models have high bias (they underfit)
- You want to improve model accuracy
- Your data is relatively clean with few outliers
- Training time is not a major constraint
- You need to capture subtle patterns in the data
- You're willing to risk some overfitting for better performance

3. Stacking (Stacked Generalization)

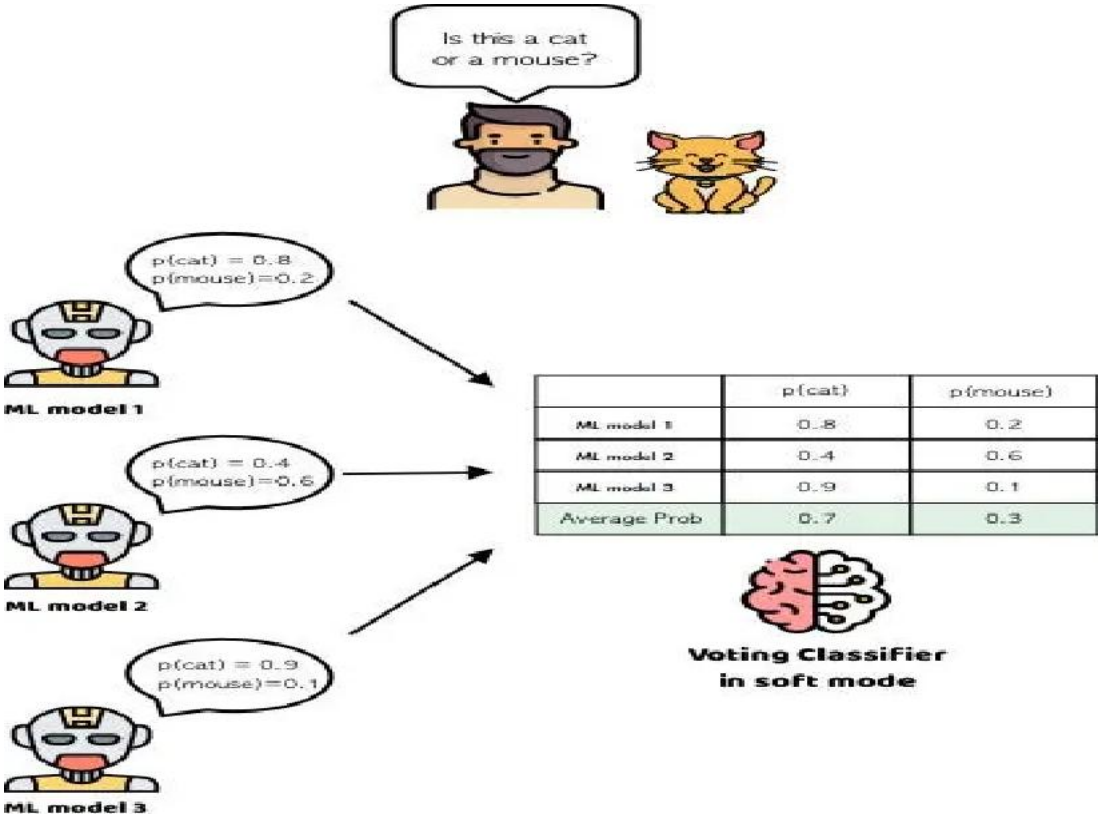
- Stacking combines multiple models by training a meta-model that learns how to best combine their predictions. The base models make predictions independently, and then a meta-learner uses these predictions as features to make the final prediction.



- When to use stacking:
- You have models with different strengths
- You're seeking the highest possible accuracy
- You have enough data to train both base models and meta-learner
- Interpretability is not your main concern
- You're working on a complex problem
- You're participating in machine learning competitions
- Real-world applications:
- Competitive machine learning (Kaggle, etc.)
- Complex natural language processing tasks
- Advanced recommendation systems
- Risk modelling in finance
- Medical outcome prediction

4. Voting

- Voting is one of the simplest ensemble techniques, where multiple models vote on the final prediction:
- Majority Voting (Hard Voting): Each model makes a prediction, and the class with the most votes becomes the final prediction.
- Weighted Voting (Soft Voting): Similar to majority voting, but some models' votes count more than others based on their performance or confidence.



- When to use voting:
- You have several well-performing but diverse models
- You need a simple and interpretable ensemble method
- Your models have complementary strengths
- You want quick improvements without complex implementation
- You have models from different sources or teams

Advantages of ensemble learning

- Improved accuracy: Ensembles typically outperform individual models.
- Reduced overfitting: By combining multiple models, ensembles can reduce the risk of overfitting.
- Robustness: Ensembles are less affected by outliers and noisy data.
- Stability: Results tend to be more stable across different datasets.
- Versatility: Ensemble methods can be applied to virtually any machine learning problem.

Limitations of ensemble learning

- Increased complexity: Ensemble models are more complex and harder to explain.
- Higher computational cost: Training multiple models requires more resources.
- Longer training time: Especially for sequential methods like boosting.
- Storage requirements: Storing multiple models needs more memory.
- Potential diminishing returns: Adding more models doesn't always improve performance proportionally.