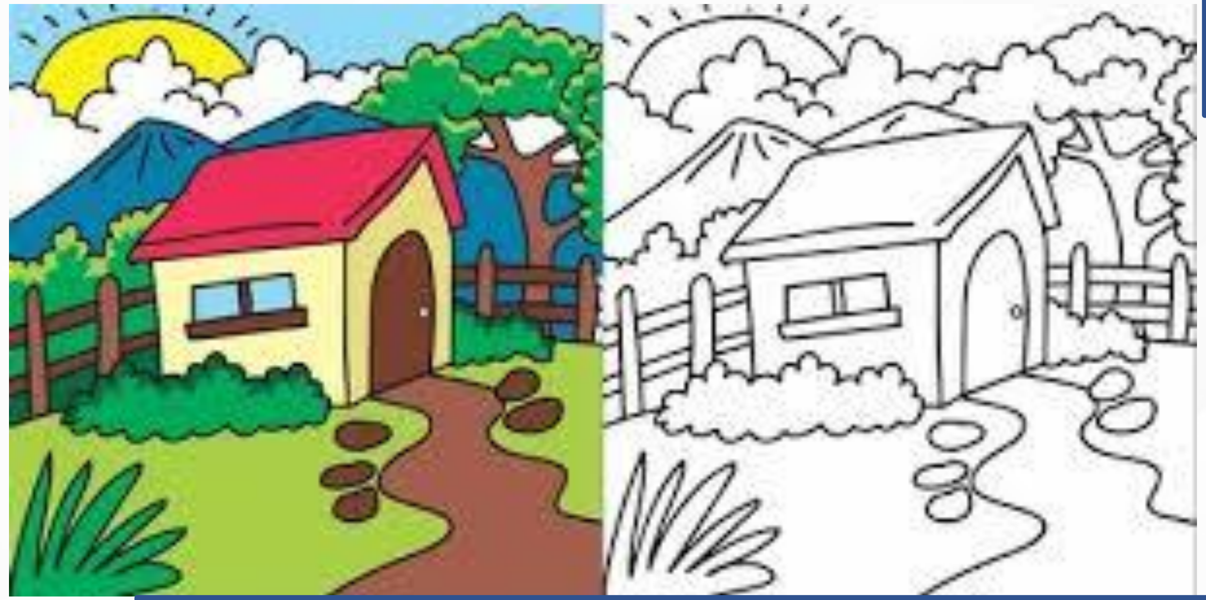
Lecture 7:
Random Forest and Ensemble Learning

Er.Roshan Shrestha
roshan.shrestha@islingtoncollege.edu.np

Agendas

- Random Forest
- Ensemble Learning
- Bagging, Boosting, Stacking

Need to buy this house



Rather than relying on a single person's opinion

- Ask multiple friends about various house experiences
- Look for online reviews and feedbacks
- Visit Real State agent for the consultation
- Consult Vastusastri for Vastu's viewpoint

Combining all these information helps to make a **WELL INFORMED DECISION.**

This is ***ENSEMBLE LEARNING***.

combining multiple sources of knowledge for a better outcome.

Ensemble: a collection of things

- a machine learning technique that combines multiple models to improve the accuracy of predictions.

- approach of combining multiple ML models

Ensemble methods in ML

- Bagging

- Boosting

- Stacking

Ensemble in Supervised, Unsupervised, Hybrid

In Supervised

Bagging (e.g., Random Forest): Combines predictions from multiple decision trees trained on random subsets of data.

Boosting (e.g., AdaBoost, Gradient Boosting): Sequentially trains models to correct errors of the previous ones.

Stacking: Combines predictions from several models using a meta-model.

In Unsupervised

ensemble techniques are less common but still applicable in unsupervised

Cluster Ensembles: Combine results from multiple clustering algorithms to achieve consensus clustering

Dimensionality Reduction Ensembles: Use different techniques (e.g., PCA, t-SNE, UMAP)

In Hybrid

Semi-Supervised Learning

Feature Engineering

Self-Training Ensembles

Bagging:
"Bring your own momo" party
Everyone brings their own plate (subset of data).
Each friend tastes and gives an opinion.
Final answer = majority vote.

Boosting → "Cooking class"
First student tries
Teacher corrects mistakes
Second student tries to fix those mistakes
Third student focuses on remaining errors

Stacking → "Team of experts"
One person checks taste
One checks smell
One checks texture
Then a super judge combines all opinions.

Bagging = Build many models on different random samples → Vote → Final result.

Boosting = sequential improvement
→ Each model tries to fix previous model's errors.

Stacking = combine predictions of different models using another model.

Random Forest:
- Many decision trees
- Each trained on random subset of data
- Each uses random features
- All vote

Prediction: C-Momo
(Because majority wins)

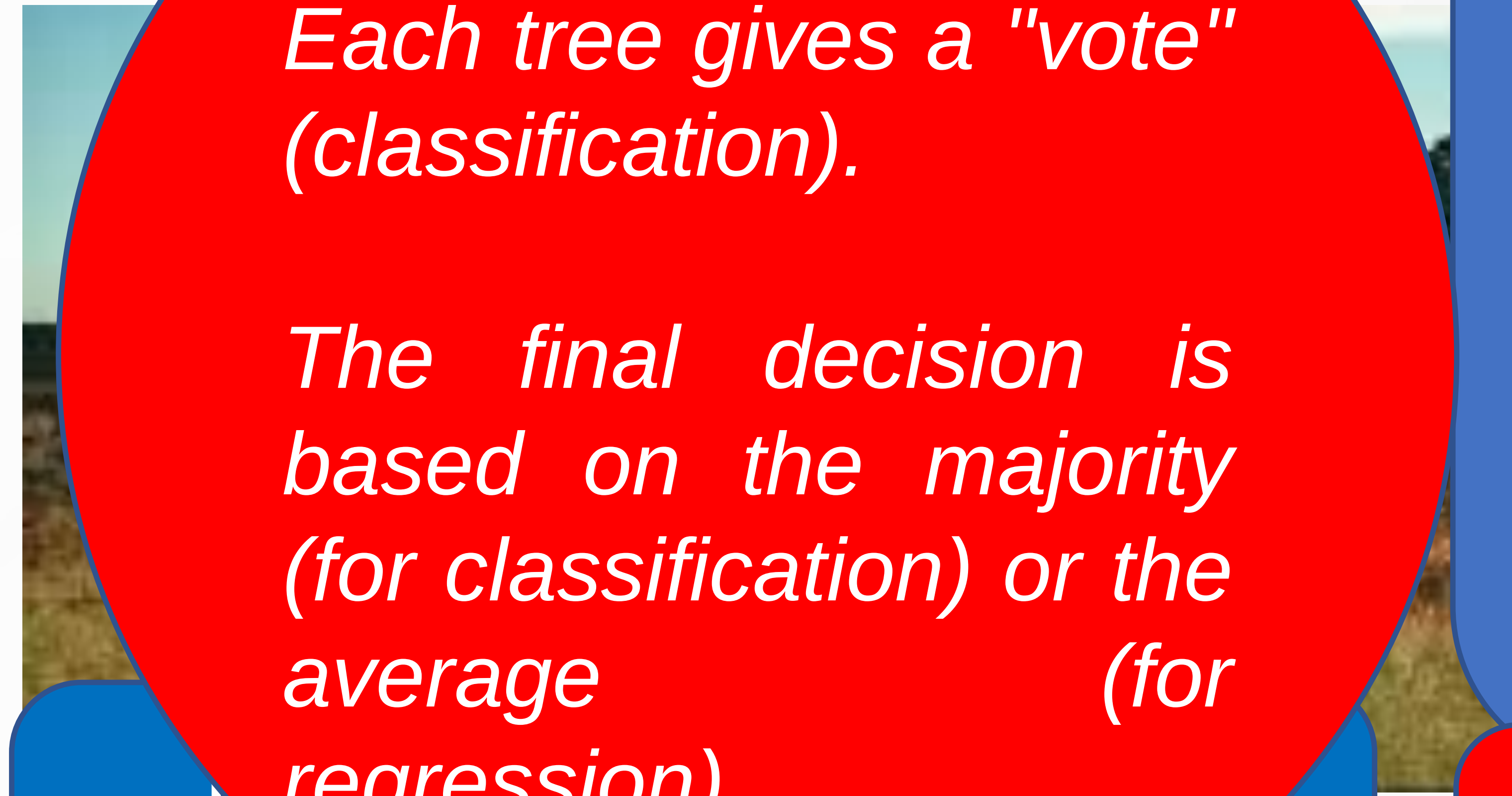
100 decision trees trained to tell:
Veg, Steam, Fried, C-Momo
Every tree sees a slightly different dataset and different features.

They vote:
60 trees → C-Momo
25 → Steam
15 → Fried

Let’s make a

In Random Forest:
Each tree gives a "vote" (classification).

The final decision is based on the majority (for classification) or the average (for regression).



opini **No,**
Not Now.

You ask 10 friends (Each friend is like a Decision Tree)
Each friend has their own criteria:
- One looks for the planned Duration
- Another checks the schedule
- Next one calls parents
- Next one checks the budget

You go with the **majority vote**.

DecisionTreeClassifier

`DecisionTreeClassifier(random_state=42)`

0.963

LogisticRegression

`LogisticRegression(max_iter=200, random_s`

0.97

KNeighborsClassifier

`KNeighborsClassifier()`

0.74

VotingClassifier

dt	lr	knn
DecisionTreeClassifier	LogisticRegression	KNeighborsClassifier

0.973

Guess tomorrow's weather

Strategy: Ask your team

Your mom
Your best friend
Your neighbor
Your dog
Google Weather
A monk on the hill

Each gives a prediction.
If one person makes the decision: Risk of being wrong
If many people vote: Much more accurate!

That's Ensemble Learning:
Instead of trusting 1 model, combine many models to get a strong, stable decision.

Decision Tree vs Random Forest

Loan Application

Input Features:
Age, income, credit score, employment history, etc.

Random Forest creates multiple decision trees. Each tree evaluates the customer based on a random subset of features:

Tree 1: Considers credit score and income.
Tree 2: Considers employment history and debt-to-income ratio.
Tree 3: Considers age and repayment history.

Output:
Each tree predicts either "Approve" or "Deny."
The Random Forest combines these predictions using a majority vote:
If most trees say "Approve," the loan is approved.
If most trees say "Deny," the loan is denied.

Result:
The decision is more accurate because it considers diverse perspectives (trees), reducing the risk of errors.

Reason Random Forest works well

- ***Reduces overfitting*** by averaging the predictions of multiple trees.
- ***Handles noisy data effectively*** since not all trees rely on the same noisy features.
- It works well with ***both classification*** (e.g., **loan approval**) and ***regression*** (e.g., **predicting house prices**).
- Have the ability to greatly increase the performance based on expanding ideas from Decision Tree
- Random Forests are ***known as ensemble learners*** since they rely on an ***ensemble of models*** (**multiple decision tree**)

```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```python
wine = load_wine()
X = wine.data  # Features
y = wine.target  #Target
```

```python
# Logistic Regression
lr = LogisticRegression()
```

```python
# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
dt_pred = dt.predict(X_test)
```

```
Accuracy Scores:
Logistic Regression: 0.98
Decision Tree: 0.94
Random Forest: 1.00
```

```python
# Random Forest
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
```

History and motivation:

- *W... ...st c*
- *f*
- s ... ha
- Create subsets of randomly picked features at each potential split i.e first proposed: **“Random decision forests” (1995)** was then extended b

prone to overfit

We can play with hyperparameters or try to pruning,still hard task to overcome the issue

Minor change in training data ----- > whole different tree

- Decision tree is restricted by the GINI IMPURITY
- No guarantee of using all features
- Root node will always be the same
- Root feature has huge influence over the tree
- Could try adjusting rules like max depth,min leaf etc

Working of random Forest

- Create multiple subsets of the data
- Train a decision tree on each subset.
- Make predictions using all trees.
- Combine predictions:
- **Classification**: Majority voting.
- **Regression**: Averaging.

- Bagging (Bootstrap Aggregating):
  - *Random sampling with replacement.*
- Feature Randomness:
  - *Each tree uses a random subset of features.*
- Ensemble Decision:
  - *Aggregated results from all trees.*

Advantages and Limitations:

Advantages:

- High accuracy.
- Works well with large datasets.
- Handles missing and categorical data.
- Resistant to overfitting (compared to individual trees).

Limitations:

- Computationally intensive for large datasets.
- Less interpretable compared to a single decision tree.
- Requires careful tuning (e.g., number of trees, max depth).

Applications:

Classification:

- Email spam detection.
- Disease diagnosis.

Regression:

- Stock price prediction.
- Weather forecasting.

islington college (इस्लिङ्टन कलेज) informatics college • pokhara ITAHARI INTERNATIONAL COLLEGE

Key Hyperparameters

- **Criterion**: gini
- *Majority of the hyperparameters are same between Decision tree classifiers and randomforest classifier*
- **n_estimators** = 100
- **max_features** = ‘auto’
- **bootstrap** = True
- **Oob_score** = False

- **Number of estimators**
  - *How many decision trees to use total in forest?*
- **Number of features**
  - *How many features to include in each random subsets in each splits?*
- **Bootstrap samples**
  - *Allow for bootstrap sampling of each training subset of features?*
- **Out-of-Bag Error**
  - *Calculate OOB error during training?*

Number of estimators and trees

- Normally more decision trees, more opportunities to learn from subsets
- Any limit to adding more trees?
- What about overfitting?
  - Random forest do not overfit

How to choose number of trees?

- Reasonable default value: 100
- Grid search for higher values
- Publications suggest 64-128 trees
- Error vs number of trees
  - Like elbow method of KNN

Number of features

Number of features

- Originally Suggests log2(N+1) random features in subset given a set of N total features
- Current suggested convention: sqrt(N)
- For regression: N/3
- ***NEED TO ADJUST BASED ON OUR DATASET***

BOOTSTRAP

- Random sampling with replacement

Bootstrap is a sampling technique where you create new datasets by **random sampling with replacement** from the original dataset.

- Original dataset size = *N*
- Bootstrap sample size = *N*
- Sampling is **with replacement**
- Some records repeat, some are left out

It is mainly used in **bagging** and **Random Forest**.

Create multiple different training datasets
Reduce variance
Improve model stability

- Bootstrap the selection of rows for each split
- Subset of features used
- Bootstrapped rows of data
- Bootstrapping is meant to reduce correlation between trees

- OUT OF **BAG** ERROR OOB
- Classification: Most voted
- Regression: Aggregated prediction of trees
- We use **b**ootstrapped data and then calculate prediction based on **ag**gregated prediction

In Random Forest, each tree is trained using a **bootstrap sample** (with replacement).

-about 63% of the data is used for training each tree, and the remaining **37%** is *NOT* used.

These unused samples are called **Out-of-Bag (OOB) samples**.

OOB Error = Error calculated by testing each tree on the data **not seen during its training**.

Works like built-in cross-validation
No need for a separate validation set
Efficient and fast
Gives a reliable estimate of model performance

A student learns from one notebook (bootstrap sample).
Questions from the other unused notebook (OOB samples) test the student.
Marking his score = OOB error.

Bagging?

- We used bootstrapping, for certain trees, certain rows of data were not used for training

OUT of Bag Samples:

- Not used for constructing some trees
- These could be used to get performance test metrics on trees that did not use these rows

Bagging (Bootstrap Aggregating)
Generate many bootstrap samples
Train a model (usually decision tree) on each sample
Combine outputs using:
Majority vote (classification)
Average (regression)

Statistical idea:
Averaging reduces variance.

Bootstrap Aggregating Bagging

- Reduce variance by averaging predictions from multiple models trained independently.
- Create multiple subsets of the training data using bootstrap sampling (random sampling with replacement).
- Train a separate model (weak learner) on each subset.
- Combine predictions by averaging (for regression) or majority voting (for classification).
- Reduces overfitting by averaging predictions.
- Works well with high-variance models like decision trees.
- Models are trained in parallel, making it computationally efficient.

Boosting

- Reduce bias by sequentially improving weak learners.

Working:

- Train a model on the full dataset.
- Identify instances where the model performs poorly (errors).
- Train the next model to focus more on these errors by assigning higher weights to misclassified instances.
- Repeat this process iteratively, creating a sequence of models.
- Combine the models’ predictions using a weighted sum or majority vote.

Advantages:

- Builds strong learners from weak learners by focusing on hard-to-predict samples.
- Excellent for handling complex datasets with high bias.

Disadvantages:

- More prone to overfitting if not regularized.
- Models are trained sequentially, which can be computationally expensive.

Boosting (AdaBoost, XGBoost, Gradient Boosting)
Models built sequentially
Each new model focuses on mistakes
Weighted errors
Combine all learners weighted by performance

Statistical idea:
Reduce bias by focusing on hard examples.

Examples

- **AdaBoost**: Assigns weights to misclassified samples and updates them iteratively.
- **Gradient Boosting**: Optimizes a loss function by adding models that minimize residual errors.
- **XGBoost/LightGBM**: Efficient implementations of Gradient Boosting.

Stacking

Stacking
Train different base models
Use their predictions as input to a meta-model
Meta-model learns how to combine answers

Statistical idea:
Diverse models reduce both bias and variance.

- Combine predictions from multiple models using a meta-model.
- Working:
- Train multiple base models (e.g., decision trees, logistic regression, etc.) on the full dataset.
- Use the predictions of these models as input features for a meta-model.
- The meta-model learns how to best combine the base models' predictions to improve overall performance.

When to use?

Bagging:

- when you have high-variance models and need to reduce overfitting (e.g., Random Forest).

Boosting:

- when you want to improve accuracy by reducing bias (e.g., Gradient Boosting).

Stacking:

- when you want to combine diverse models for maximum accuracy.

Conclusion

Ensemble = group of models working together
Bagging = reduce variance
Boosting = reduce bias
Stacking = combine experts
Random Forest = bagging + random features + many trees
RF = one of the most powerful out-of-the-box ML algorithms

End of Lecture 7

Thank you !
Any questions ?