Random Forest (VVIMP)

```
from sklearn.ensemble import RandomForestClassifier
```

- **Used in every ML project.**
- Can be used in all ML problems.
- Regression + Classification
- **No need to do much tuning.**

> Random Forest is an **ensemble learning method** that builds **multiple Decision Trees** and combines their results to **improve accuracy and reduce overfitting**.

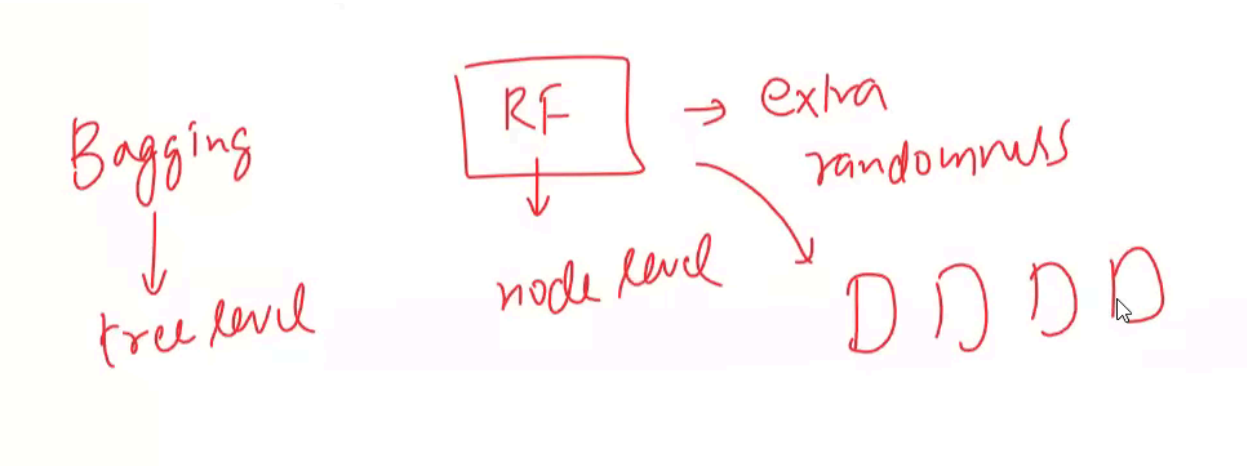
What is Random Forest?

- **Ensemble of Trees**:
  - A Random Forest is a collection of **Decision Trees**, where each tree is trained on a different subset of the data and features.
  - The final prediction is made by **averaging** the predictions of all trees (for regression) or taking a **majority vote** (for classification).
- **Randomness**:
  - Each tree is trained on a **random subset of the data** (bootstrap sampling) and a **random subset of features**.
  - This randomness ensures that the trees are diverse, reducing overfitting and improving generalization.

- You can sample rows &/or columns both

Bagging vs RF

- In RF, the base model is DT.
  - In Bagging, you can use any model.
- In case of column level sampling (`max_features`):
  - RF → Node level sampling
  - Bagging → Tree level sampling



✨**RF outperforms Bagging.**

Python Code:

```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import pandas as pd
```

```
# Load the Boston Housing dataset
df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv')
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]



rf_regressor = RandomForestRegressor(n_estimators=150, max_samples=0.8, n_jobs=-1)

# Train the model
rf_regressor.fit(X_train, y_train)

# Predictions
y_pred = rf_regressor.predict(X_test)

# Evaluate performance
mse = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
```

```
Mean Squared Error: 0.8805
```

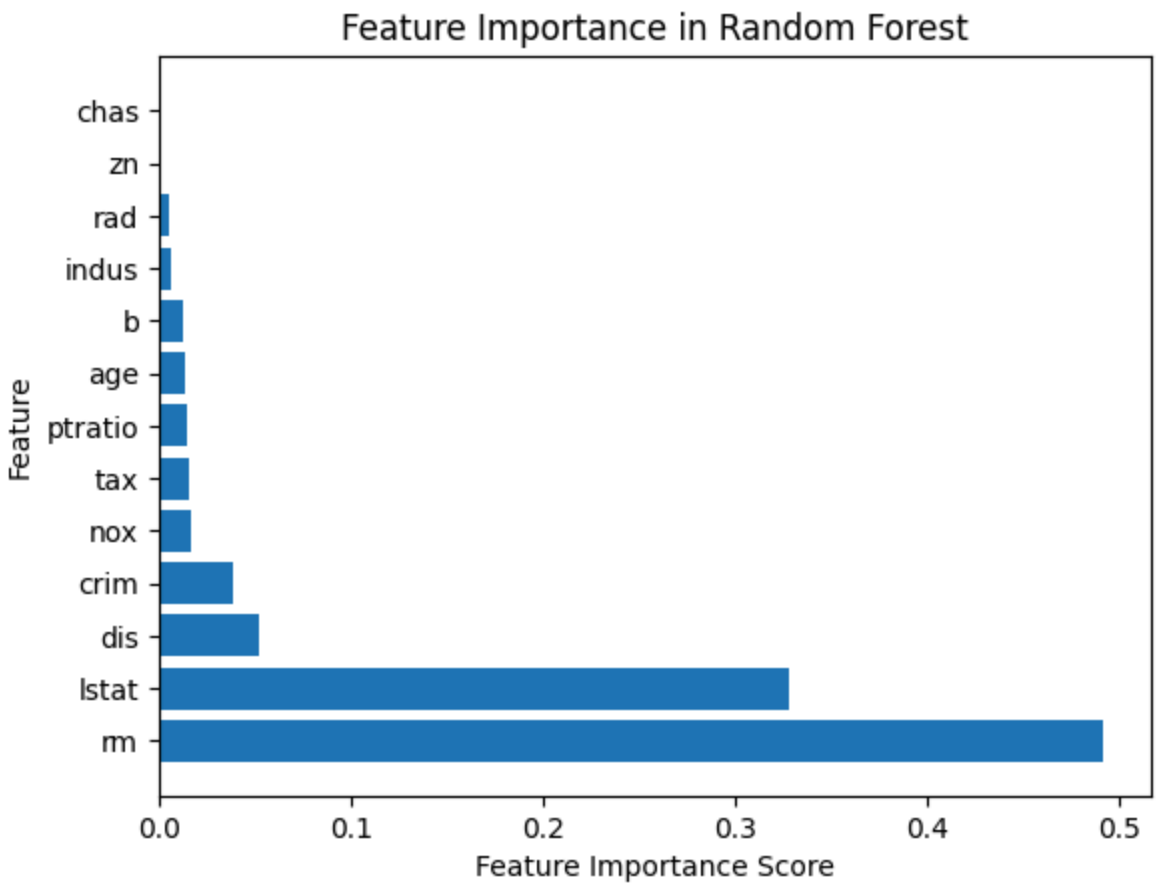
Feature Importance in Random Forest

```
importance_df = pd.DataFrame({'Feature': df.columns[:-1], 'Importance': rf_regressor.feature_importances_})
importance_df = importance_df.sort_values(by='Importance', ascending=False)
importance_df
```

	Feature	Importance
5	rm	0.492225
12	lstat	0.328478
7	dis	0.052715
0	crim	0.038702
4	nox	0.016622
9	tax	0.015871
10	ptratio	0.014803
6	age	0.014124
11	b	0.012371
2	indus	0.006612
8	rad	0.005198
1	zn	0.001303
3	chas	0.000975

```
# Plot feature importance
import matplotlib.pyplot as plt

plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.xlabel("Feature Importance Score")
plt.ylabel("Feature")
plt.title("Feature Importance in Random Forest")
plt.show()
```



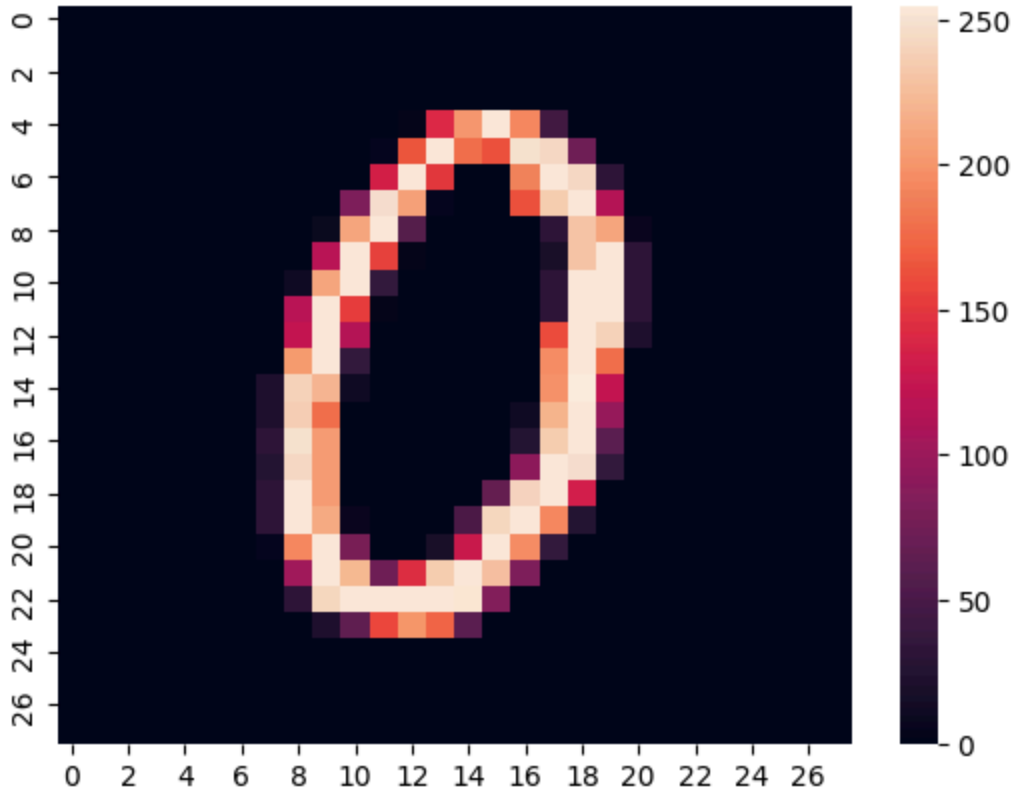
Feature Importance of MNist Dataset

```
import pandas as pd
import seaborn as sns

df = pd.read_csv(r'https://raw.githubusercontent.com/G1Codes/digit-recognizer/refs/heads/main/digit-recognizer_train.csv')

X = df.iloc[:,1:]
y = df.iloc[:,0]

sns.heatmap(X.iloc[5].values.reshape(28,28))
```

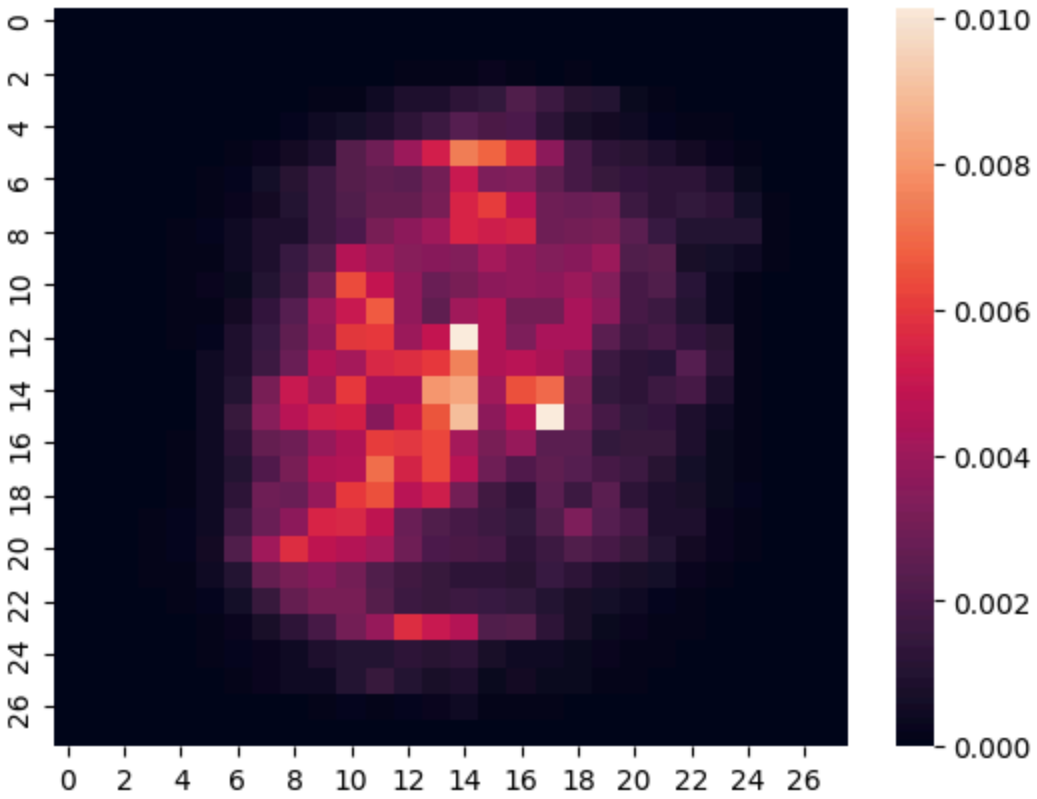



```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

rf.fit(X,y)
```

```
sns.heatmap(rf.feature_importances_.reshape(28,28))
```



Advantages of Random Forest

✔ **Handles Missing Values**: Can handle missing data better than decision trees.

✔ **Prevents Overfitting**: Reduces variance by averaging multiple trees.

✔ **Feature Selection**: Automatically ranks important features.

✔ **Works with Categorical & Numerical Data**: Flexible for different types of datasets.

Disadvantages of Random Forest

❌ **Computationally Expensive**: Slower than single decision trees for large datasets.

❌ **Hard to Interpret**: Individual trees are interpretable, but an ensemble of many trees is complex.

❌ **Can't Extrapolate** - This is because they do not predict beyond the range of the training data, and that they may not predict as accurately as other regression models.

Key Hyperparameters in Random Forest

1. `n_estimators` :
   - Number of trees in the forest.
   - **More trees generally improve performance but increase computation time.**
2. `max_depth` `=None` :
   - Maximum depth of each tree. A **smaller depth prevents overfitting**.
3. `max_samples` `=None` : (***0 to 1 (for %), INT (for number)***)
   - How many **rows** you want in each dataset
   - You can also provide integer (no. of rows)
4. `min_samples_split` `=2` :
   - Minimum number of samples required to split a node.
5. `min_samples_leaf` `=1` :
   - Minimum number of samples required to be at a leaf node.
6. `max_features` `s='sqrt'` :
   - Number of features(**columns**) to consider when looking for the best split.
   - Common values are `'sqrt'` (square root of total features) or `'log2'` (logarithm of total features).
7. `Warm_start` `=False` `:` means you can **add more trees to an existing forest without retraining from scratch**, saving time.
   - Useful when you want to experiment with different numbers of trees or when training on **large datasets**.

`set_params` is a method in scikit-learn that allows you to **update the hyperparameters** of a model after it has been initialized. It is particularly useful

when you want to change some parameters without creating a new model instance.

```
from sklearn.ensemble import RandomForestClassifier

# Initialize a Random Forest model
rf = RandomForestClassifier(n_estimators=50, warm_start=True, random_state
=42)

# Train the model with 50 trees
rf.fit(X_train, y_train)

# Use set_params to increase the number of trees to 100
rf.set_params(n_estimators=100)

# Train the model again (only trains 50 additional trees due to warm_start)
rf.fit(X_train, y_train)
```

Forest Level HP	Tree Level HP	Miscellaneous HP
N_estimators =100	**Criterion** = *'gini'*	Oob_score
Max_features ='sqrt'	Max_depth	N_jobs
Bootstrap = True	Min_Samples_split	Random_state
Max_samples	Min_samples_leaf	verbose
	Min_weight_fraction_leaf	Warm_start
	Max_leaf_nodes	Class_weight
	Min_impurity_decrease	
	Ccp_apha	

- Refer the link👆

Extremely Randomized Trees

`from sklearn.ensemble import ExtraTreesClassifier`

> 💡 **It’s altogether a different model.**

- Introduces an extra randomness.
- It's a modification of the Random Forest algorithm that **changes the way the splitting points for decision tree branches are chosen.**

Random Splits:

- In Random Forests, the best split is chosen from a random subset of features at each node.
- In ExtraTrees, the split is chosen **completely at random** from the range of values in the selected feature. This adds an extra layer of randomness.

1. **Reduced Variance**:
    - By using random splits, ExtraTrees reduce the variance of the model, which can lead to better generalization on unseen data.
2. **Faster Training**:
    - Since ExtraTrees do not search for the optimal split (unlike Random Forests), they are generally faster to train.
3. **Ensemble of Trees**:
    - Like Random Forests, ExtraTrees combine the predictions of multiple decision trees to make a final prediction (e.g., through voting for classification or averaging for regression).

How ExtraTrees Work:

1. **Tree Construction**:
    - For each tree in the ensemble:
        - At each node, a random subset of features is selected.
        - A split is chosen **randomly** from the range of values in the selected feature.
        - The tree is grown until a stopping criterion is met (e.g., maximum depth or minimum samples per leaf).
2. **Ensemble Prediction**:
    - For classification, the final prediction is the majority vote of all trees.
    - For regression, the final prediction is the average of all tree predictions.

Advantages of ExtraTrees:

1. **Reduced Overfitting**:
    - The additional randomness helps prevent overfitting, making the model more robust to noise in the data.
2. **Faster Training**:
    - Since ExtraTrees do not search for the optimal split, they are computationally faster than Random Forests.
3. **Good for High-Dimensional Data**:
    - ExtraTrees perform well on datasets with a large number of features.

Disadvantages of ExtraTrees:

1. **Less Interpretability**:
    - The added randomness makes it harder to interpret the model compared to standard decision trees.
2. **May Require More Trees**:

- Due to the increased randomness, ExtraTrees may require a larger number of trees to achieve optimal performance.

Python code for **ExtraTrees:**

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat
e=42)

# Initialize ExtraTreesClassifier
model = ExtraTreesClassifier(random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

- **Same hyperparameters as that of RF**