Assignment 04

Bhargavi Manyala

October 8, 2025

> explain Random Forest Regressor in detail

What is Random Forest Regressor? Random Forest Regressor is an ensemble learning method used for regression tasks (predicting continuous values). It builds multiple decision trees during training and outputs the average prediction of all individual trees.

- Core Concepts

1. Ensemble Learning Instead of relying on a single decision tree (which can overfit), Random Forest combines many trees to create a more robust and accurate model. This is called "wisdom of the crowds."
2. Bootstrap Aggregating (Bagging) Each tree is trained on a different random subset of the training data, created by:

- Sampling with replacement (bootstrap sampling) Some data points may appear multiple times, others not at all Typically, each tree sees about 63% of unique training samples

3. Feature Randomness At each split in every tree, Random Forest:

Only considers a random subset of features (not all features) Default: √(total features) for classification, or (total features)/3 for regression This decorrelates the trees, making them more independent

- How It Works Training Phase:

Create bootstrap samples: Generate N different random samples from training data (with replacement) Build N decision trees: For each sample, grow a decision tree where:

At each node, randomly select a subset of features Choose the best split among those features only Grow tree to maximum depth (or until stopping criteria)

Store all trees: Keep all N trees in the "forest"

- Prediction Phase:

Pass input through all trees: Each tree makes its own prediction Aggregate predictions: Average all tree outputs for the final prediction

Final Prediction = (Tree1 + Tree2 + … + TreeN) / N

- Key Hyperparameters n_estimators (numTrees in Spark)

Number of trees in the forest More trees = better performance but slower training Typical range: 100-500 Trade-off: Accuracy vs computation time

max_depth (maxDepth in Spark)

Maximum depth of each tree Controls model complexity Too deep: Overfitting Too shallow: Underfitting Typical range: 5-30

min_samples_split (minInstancesPerNode in Spark)

Minimum samples required to split a node Higher values prevent overfitting Typical range: 2-20

max_features (featureSubsetStrategy in Spark)

Number of features to consider at each split "auto" or "sqrt": √(n_features) "log2": log (n_features) Lower values = more randomness, less correlation between trees

min_samples_leaf

Minimum samples required at leaf nodes Smooths predictions, prevents overfitting

- Feature Importance Random Forest provides feature importance scores based on:

Mean Decrease in Impurity (MDI): How much each feature decreases impurity (variance for regression) across all trees Features used in early splits or that reduce variance significantly get higher scores

```
# In sklearn
importances = model.feature_importances_

# In Spark
model.featureImportances
```

```python
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator

# Initialize model
rf = RandomForestRegressor(
    featuresCol="features",
    labelCol="label",
    numTrees=100,            # Number of trees
    maxDepth=10,             # Max depth of each tree
    minInstancesPerNode=1,  # Min samples per node
    featureSubsetStrategy="auto",  # Features per split
    seed=42
)

# Train model
model = rf.fit(train_data)

# Make predictions
predictions = model.transform(test_data)

# Evaluate
evaluator = RegressionEvaluator(labelCol="label", metricName="rmse")
rmse = evaluator.evaluate(predictions)

# Feature importance
print(model.featureImportances)
```

> Explain how to extract the feature importance from the Random Forest model

- Overview Feature importance measures how much each feature contributes to the model's predictions. Random Forest calculates this based on how much each feature reduces impurity (variance for regression, Gini/entropy for classification) across all trees.
- How Feature Importance is Calculated Mean Decrease in Impurity (MDI) For each feature: Look at all nodes across all trees where that feature is used for splitting Calculate how much that split reduced impurity (variance in regression) Average these reductions across all trees Normalize so all importances sum to 1.0
- Extracting Feature Importance in Spark MLlib
- Method 1: From Trained Model

```python
from pyspark.ml.regression import RandomForestRegressor

# Train the model
rf = RandomForestRegressor(featuresCol="features", labelCol="label", numTrees=100)
model = rf.fit(train_data)

# Extract feature importances
importances = model.featureImportances

print(importances)
# Output: SparseVector or DenseVector like (5,[0,1,2,3,4],[0.25,0.35,0.15,0.20,0.05])
```

- Method 2: Convert to Readable Format

```python
# Get feature importances as a list
importance_scores = model.featureImportances.toArray()

# Assuming you have feature names
feature_names = ["MIN_YEARS_EXPERIENCE", "MAX_YEARS_EXPERIENCE",
                 "DURATION", "SALARY_FROM", "COMPANY_SIZE"]

# Create a list of (feature, importance) tuples
feature_importance_list = list(zip(feature_names, importance_scores))

# Sort by importance (descending)
feature_importance_list.sort(key=lambda x: x[1], reverse=True)

# Print
for feature, importance in feature_importance_list:
    print(f"{feature}: {importance:.4f}")
```

- Method 3: Create a Pandas DataFrame for Analysis

```python
import pandas as pd

# Convert to pandas DataFrame
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importance_scores
}).sort_values('Importance', ascending=False)

print(importance_df)
```

- Visualizing Feature Importance

```python
import matplotlib.pyplot as plt

# Sort by importance
importance_df_sorted = importance_df.sort_values('Importance', ascending=True)

# Create horizontal bar chart
plt.figure(figsize=(10, 6))
plt.barh(importance_df_sorted['Feature'], importance_df_sorted['Importance'])
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.tight_layout()
plt.show()
```