Algorithm: Random Forest

Random Forest is an ensemble-based supervised learning algorithm that combines multiple decision trees to produce more accurate and robust predictions. In this project, it is used for multi-class text classification, predicting review ratings (1–5 stars) based on review content vectorized using TF-IDF. Random Forest works well with both structured and unstructured data, and is particularly effective when feature interactions are important.

How the Algorithm Works

Ensemble Learning (Bagging)

Random Forest builds multiple decision trees during training and merges their outputs (via majority voting for classification). Each tree is trained on a **bootstrap sample** (random sample with replacement) of the training data, and at each node, a random subset of features is considered for splitting. This introduces **decorrelation** among the trees, which reduces variance and avoids overfitting.

Steps:

1. Draw n_estimators bootstrap samples from the training set.
2. For each sample, grow an unpruned decision tree.
3. At each node, select the best split from a random subset of features.
4. Final prediction: **majority vote** across all trees.

Multi-Class Classification

Random Forest handles multi-class classification **natively**, using **majority voting** across all trees. Each tree gives a class prediction, and the most frequently predicted class becomes the model’s output:

$$\hat{y} = \mathrm{mode}(h_1(x), h_2(x), ..., h_n(x))$$

Where hi(x) is the prediction of the *i*-th decision tree.

Key Hyperparameters

Hyperparameter	Description	Default	Tuning Tips
n_estimators	Number of decision trees in the forest	100	Higher = better performance but slower training
max_depth	Maximum depth of each tree	None	Limit to prevent overfitting; tune based on validation
min_samples_split	Minimum samples to split a node	2	Increase to make trees more conservative
min_samples_leaf	Minimum samples per leaf node	1	Helps reduce model complexity
max_features	Number of features to consider when splitting	'auto'	'sqrt' is common for classification tasks
criterion	Function to measure split quality (e.g., 'gini', 'entropy')	'gini'	Try both to compare

Strengths

- Handles both **numerical and categorical features**
- Naturally supports **multi-class classification**
- **Robust to outliers** and **nonlinear** relationships
- Performs well even with **imbalanced features**
- **Reduces overfitting** compared to individual decision trees
- **Feature importance** can be extracted directly

Limitations

- Can be **computationally expensive** (especially with many trees)
- Less interpretable than single decision trees or logistic regression
- May **overfit** on noisy datasets if not properly tuned
- Requires more memory and training time than simpler models

Use When

- You want a **strong baseline** with minimal preprocessing
- Data includes **nonlinear relationships** or interactions
- You want **feature importance** insights
- TF-IDF features have **high dimensionality**

Avoid When

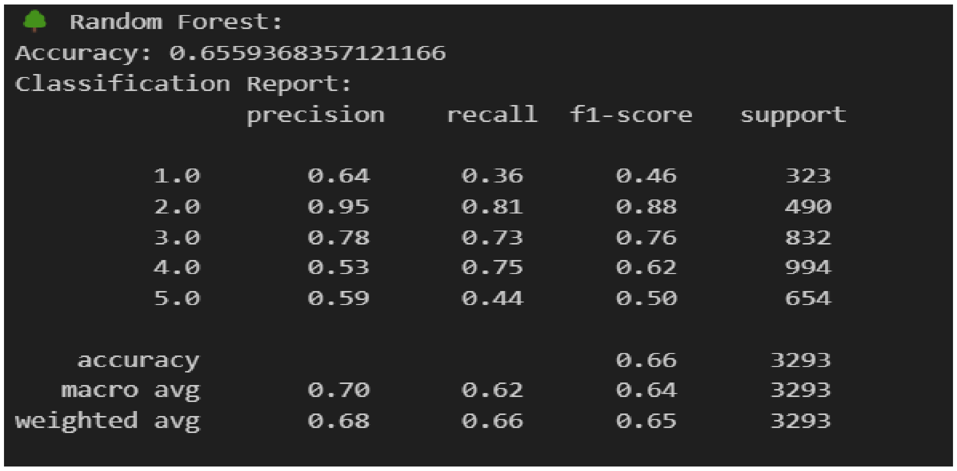
- Interpretability is critical for deployment
- You need **real-time predictions** with low latency
- Dataset is **very large** and computational efficiency is a concern

Model Evaluation: Random Forest Results:

Code:

```
# Train Random Forest

rf = RandomForestClassifier(n_estimators=100, random_state=42)

rf.fit(Xtrain_vec, y_train)


# Predict

y_pred_rf = rf.predict(Xtest_vec)


# Evaluation

print("🌳 Random Forest:")

print("Accuracy:", accuracy_score(y_test, y_pred_rf))

print("Classification Report:\n", classification_report(y_test, y_pred_rf))
```

Result:



```
🌳 Random Forest:
Accuracy: 0.6559368357121166
Classification Report:
               precision    recall  f1-score   support

         1.0       0.64      0.36      0.46       323
         2.0       0.95      0.81      0.88       490
         3.0       0.78      0.73      0.76       832
         4.0       0.53      0.75      0.62       994
         5.0       0.59      0.44      0.50       654

    accuracy                           0.66      3293
   macro avg       0.70      0.62      0.64      3293
weighted avg       0.68      0.66      0.65      3293
```

The **Random Forest classifier** was applied to a multi-class text classification task aimed at predicting review ratings (1 to 5 stars). The evaluation was done on a test set of **3,293 samples**, and the performance is summarized using standard classification metrics: **precision**, **recall**, **F1-score**, and **support** for each class.

Overall Accuracy

- **Accuracy**: 0.66
  This means that **66%** of the predictions made by the Random Forest model match the actual labels. This is a strong result for a multi-class problem with potential class imbalance.

Hyperparameter tuning

To improve the predictive performance and generalization ability of the **Random Forest classifier**, we applied **hyperparameter tuning** using `GridSearchCV`. Random Forest has several hyperparameters that can significantly affect model behavior. Tuning them allows us to balance bias-variance trade-offs and prevent overfitting or underfitting.

Objective

The goal was to identify the optimal set of hyperparameters that yield the highest accuracy on the validation set. We used **GridSearchCV**, which performs an exhaustive search over specified parameter values using **cross-validation (cv=3)** and **accuracy** as the scoring metric.

Random Forest models can overfit or underfit if not properly tuned. Hyperparameter tuning ensures:

- **Deeper trees** don’t overfit small noise.
- **Sufficient trees** are used to stabilize predictions.
- **Balanced splits and leaf sizes** help avoid complex or trivial trees.
- **Bootstrap** variations prevent over-reliance on certain samples.

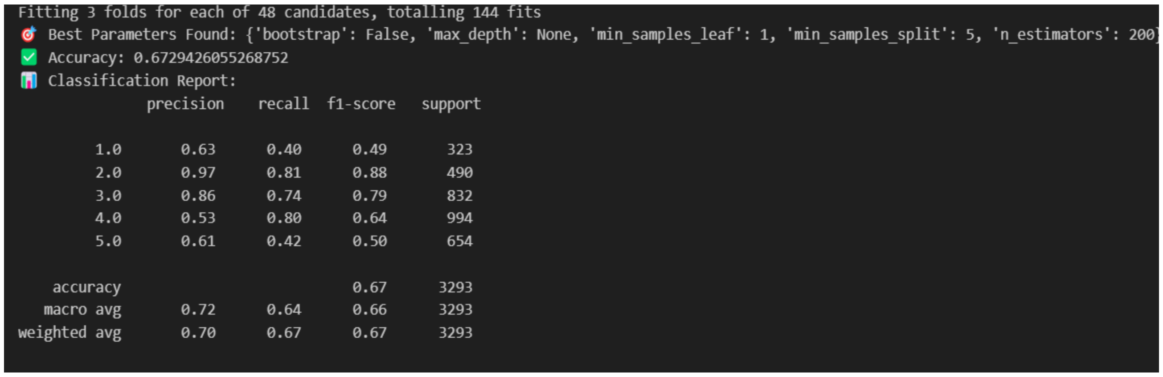
This tuning process ensures that the Random Forest model generalizes well across unseen data, boosting performance while reducing the risk of model variance or bias.

Code

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'bootstrap': [True, False]
}

# Initialize RandomForestClassifier
rf = RandomForestClassifier(random_state=42)

# Setup GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=3, n_jobs=-1, verbose=1, scoring='accuracy')

# Fit on training data
grid_search.fit(Xtrain_vec, y_train)

# Best model
best_rf = grid_search.best_estimator_
```

```
# Predict

y_pred_rf = best_rf.predict(Xtest_vec)

# Evaluation

print(" Best Parameters Found:", grid_search.best_params_)

print(" Accuracy:", accuracy_score(y_test, y_pred_rf))

print(" Classification Report:\n", classification_report(y_test, y_pred_rf))
```

OUTPUT



```
Fitting 3 folds for each of 48 candidates, totalling 144 fits
🎯 Best Parameters Found: {'bootstrap': False, 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200
✅ Accuracy: 0.6729426055268752
📊 Classification Report:
              precision    recall  f1-score   support

         1.0       0.63      0.40      0.49       323
         2.0       0.97      0.81      0.88       490
         3.0       0.86      0.74      0.79       832
         4.0       0.53      0.80      0.64       994
         5.0       0.61      0.42      0.50       654

    accuracy                           0.67      3293
   macro avg       0.72      0.64      0.66      3293
weighted avg       0.70      0.67      0.67      3293
```