Random Forest Classifier: Model Summary

Why Logistic Regression?

Logistic Regression is a **simple, interpretable, and effective baseline** for multiclass classification problems — especially in **text classification**, where TF-IDF features often lead to high-dimensional, sparse data.

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import GridSearchCV

```
# Define hyperparameter grid
param_grid2 = {
    'C': [0.01, 0.1, 1, 10],            # Inverse of regularization strength
    'penalty': ['l2'],                  # L1 and L2 are regularization methods
    'solver': ['liblinear', 'lbfgs', 'saga'],
    'max_iter': [100, 200, 500]         # Number of iterations to converge
}

# Initialize the base model
log_reg = LogisticRegression(multi_class='auto', random_state=42)

# Grid Search with cross-validation
grid_search2 = GridSearchCV(log_reg, param_grid2, scoring='f1_macro', cv=3, verbose=2,
n_jobs=-1)
grid_search2.fit(X_train_tfidf, y_train)

# Best model
best_model2 = grid_search2.best_estimator_

# Evaluate
y_pred2 = best_model2.predict(X_test_tfidf)
print(classification_report(y_test, y_pred2))
```

Algorithm Type & Objective

Type**: Supervised Learning → Classification algorithm**

Objective**: Predict the** class label **(in our case, review rating 1–5) by modeling the** probability **that a given input belongs to a particular class.**

How the Algorithm Works

Mathematical Intuition

- Unlike Linear Regression, which outputs continuous values, **Logistic Regression uses a sigmoid (logistic) function** to output a value between 0 and 1 → interpreted as a **probability**.

For Binary Classification:

P(y=1|X)=11+e−(β0+β1x1+β2x2+⋯+βnxn)P(y=1|X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)}}P(y=1|X)=1+e−(β0+β1x1+β2x2+⋯+βnxn)1

- The model predicts `y=1` if this probability is above 0.5 (default threshold).

For Multiclass Classification (5-star rating problem):

Uses **Softmax Regression** (also called Multinomial Logistic Regression), where the model estimates probabilities for each class:

P(y=k|x)=ex⊤βk∑j=1Kex⊤βjP(y = k|x) = \frac{e^{x^\top \beta_k}}{\sum_{j=1}^{K} e^{x^\top \beta_j}}P(y=k|x)=∑j=1Kex⊤βjex⊤βk

Workflow

1. **Input**: TF-IDF vectorized text (e.g., reviews).
2. **Weight Learning**: Learns optimal coefficients β to minimize the **log-loss (cross-entropy loss)**.
3. **Prediction**: Computes class probabilities and selects the class with the highest one.

Key Hyperparameters

One of the most important hyperparameters in Logistic Regression is `C`, which controls the inverse of regularization strength. Smaller values of `C` imply stronger regularization, helping to prevent overfitting in high-dimensional data like TF-IDF vectors. Tuning `C` across a range such as 0.01, 0.1, 1, and 10 can significantly impact performance.

The `penalty` parameter determines the type of regularization applied—commonly either L2 (Ridge) or L1 (Lasso). L2 is typically used by default and is suitable for most text classification problems, while L1 can be helpful if feature selection is desired, though it's supported only with specific solvers like `liblinear` or `saga`.

The `solver` parameter chooses the optimization algorithm. For smaller datasets, `liblinear` works well, whereas `saga` is preferred for large, sparse datasets (common in NLP). The `max_iter` parameter defines the maximum number of iterations taken for the optimization to converge. It may need to be increased (e.g., 200–300) if the model fails to converge with a warning. Lastly, the `multi_class` parameter decides how multiclass classification is handled. Using `auto` lets scikit-learn decide based on the solver, but explicitly setting it to `'multinomial'` is better for softmax-based multiclass logistic regression, especially with the `lbfgs` or `saga` solvers.

Strengths (Especially for Text Classification)

Interpretable (you can examine feature importance via weights)
Works well with **sparse**, **high-dimensional** data (like TF-IDF)
Fast training & prediction
Robust with regularization
Easily tunable

Limitations

Assumes **linear decision boundaries** between classes
Not ideal for **nonlinear patterns** in data
Struggles with **severe class imbalance**
May underperform on complex interactions (which tree-based models capture better)

When to Use Logistic Regression

- As a **baseline model** in NLP problems
- When interpretability matters

- When the dataset is **linearly separable** or near-linear
- For **sparse high-dimensional** input features (like Bag of Words / TF-IDF)

When NOT to Use

- When relationships between features and output are **nonlinear** and complex
- If performance plateaus and more expressive models (like XGBoost, SVM, or neural networks) are required
- For very large datasets with many classes and severe imbalance

Classification Report

```
precision    recall  f1-score   support

           1       0.51      0.63      0.56       400
           2       0.32      0.26      0.29       400
           3       0.35      0.30      0.32       400
           4       0.42      0.42      0.42       400
           5       0.54      0.59      0.56       400

    accuracy                           0.44      2000
   macro avg       0.43      0.44      0.43      2000
weighted avg       0.43      0.44      0.43      2000
```