Algorithm: Logistic Regression

Logistic Regression is a supervised learning algorithm used primarily for classification tasks. In this project, it is used for multi-class text classification to predict review ratings (1–5 stars) based on review content. Despite its name, Logistic Regression is a classification, not regression, algorithm.

How the Algorithm Works

Binary Classification

For binary classification, logistic regression models the probability that an input x belongs to class 1:

$$z = \mathbf{w}^T\mathbf{x} + b$$

$$P(y = 1 \mid \mathbf{x}) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Where:

- w: weight vector
- x: feature vector (e.g., TF-IDF)
- b: bias term
- σ(z): sigmoid function that maps z to (0, 1)

Prediction is made by:

$$\hat{y} = \begin{cases} 1 & \text{if } \sigma(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Multi-Class Classification (Softmax Regression)

When there are more than two classes (e.g., 5 review rating levels), logistic regression uses the **softmax function** to compute the probability of each class k ∈ {1,2,...,K}:

$$z_k = \mathbf{w}_k^T \mathbf{x} + b_k$$

$$P(y = k \mid \mathbf{x}) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } k = 1, 2, ..., K$$

The model predicts the class with the **highest probability**:

$$\hat{y} = \arg\max_k \; P(y = k \mid \mathbf{x})$$

Key Hyperparameter:

Hyperparameter	Description	Default	Tuning Tips
C	Inverse of regularization strength	1.0	Lower = stronger regularization
penalty	Type of regularization (`l1`, `l2`, etc.)	l2	Use l2 for text data
solver	Optimization algorithm (liblinear, saga, etc.)	lbfgs	Use saga for l1 or large datasets
multi_class	Strategy: ovr (one-vs-rest) or multinomial (softmax)	auto	Use multinomial with softmax logic
max_iter	Max optimization iterations	100	Increase if not converging

Strengths:

- Fast training and prediction
- Works well with high-dimensional sparse data (like TF-IDF)
- Easy to interpret (coefficients show feature importance)
- Performs well on linearly separable classes

Limitations:

- Assumes linear decision boundary
- Not ideal for complex, nonlinear relationships
- Sensitive to irrelevant features and multicollinearity
- Performance drops with highly imbalanced classes

Use When:

- Dataset is large and well-preprocessed
- You need a fast, interpretable baseline
- Input features are TF-IDF or BoW vectors

Avoid When:

- Data is highly nonlinear or hierarchical
- High accuracy is required for long-term deployment
- You want contextual embeddings (BERT, LSTM, etc.)

Model Evaluation: Logistic Regression Results:

Code:

```
# Train Logistic Regression
lr = LogisticRegression(max_iter=1000)
lr.fit(Xtrain_vec, y_train)

# Predict
y_pred_lr = lr.predict(Xtest_vec)

# Evaluation
print(" Logistic Regression:")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Classification Report:\n", classification_report(y_test, y_pred_lr))
```

Result:

```
 Logistic Regression:
Accuracy: 0.44718864950078824
Classification Report:
              precision    recall  f1-score   support

         1.0       0.49      0.56      0.52       371
         2.0       0.38      0.33      0.35       382
         3.0       0.36      0.32      0.34       383
         4.0       0.40      0.43      0.42       387
         5.0       0.58      0.60      0.59       380

    accuracy                           0.45      1903
   macro avg       0.44      0.45      0.44      1903
weighted avg       0.44      0.45      0.44      1903
```

After training the Logistic Regression model on the text data using TF-IDF features, we evaluated its performance on the test set using standard classification metrics.

- **Accuracy:** 0.447 (44.7%)
  This indicates that the model correctly predicted the review class in about 45% of the test samples, which is a reasonable baseline for a 5-class classification task.
- **Macro Average F1-Score:** 0.44
  A simple average over all classes, treating each class equally.
- **Weighted Average F1-Score:** 0.44
  Accounts for the support (number of instances) of each class, giving a more balanced overall view.
- The model performs **best on class 5.0**, with an F1-score of **0.59**, indicating it is best at identifying highly positive reviews.
- Classes **2.0 and 3.0** have the **lowest performance**, which suggests overlap in feature space or ambiguity in the reviews that correspond to neutral sentiments.

Hyperparameter tuning

To improve the performance of the Logistic Regression model, we performed **hyperparameter tuning** using **GridSearchCV**, a powerful technique from `scikit-learn` that automates the process of searching for the best combination of hyperparameters through cross-validation.

We focused on tuning the following key hyperparameters of the `LogisticRegression` model:

- **`C`**: This is the inverse of regularization strength. A lower value implies stronger regularization. We tested values `[0.01, 0.1, 1, 10]` to find the optimal trade-off between underfitting and overfitting.
- **`penalty`**: This determines the type of regularization applied. We tested both `'l1'` (Lasso) and `'l2'` (Ridge) penalties.
- **`solver`**: Different solvers handle optimization differently and support different penalties. We chose `'liblinear'` and `'saga'`, both of which support `l1` and `l2` penalties.

A **5-fold cross-validation** (`cv=5`) was used to ensure robust and unbiased performance estimation for each hyperparameter combination. The grid search was parallelized using `n_jobs=-1` to speed up the computation.

Once the grid search was completed, the model with the highest cross-validation accuracy was selected as the **best estimator**. This tuned model was then used to make predictions on the test set.

Finally, we evaluated the model using **accuracy score** and **classification report**, which includes precision, recall, and F1-score for each class. The improvement in test accuracy after tuning confirmed that proper hyperparameter optimization can significantly enhance model performance.

Code

```
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import LogisticRegression

# Define parameter grid

param_grid = {

    'C': [0.01, 0.1, 1, 10],              # Regularization strength

    'penalty': ['l1', 'l2'],              # L1 = Lasso, L2 = Ridge

    'solver': ['liblinear', 'saga']       # Compatible solvers

}

# Create base model

lr = LogisticRegression(max_iter=1000)

# Grid search

grid_search = GridSearchCV(lr, param_grid, cv=5, scoring='accuracy', n_jobs=-1)

grid_search.fit(Xtrain_vec, y_train)

# Best parameters

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

# Best model

best_lr = grid_search.best_estimator_

# Predict using best model

y_pred = best_lr.predict(Xtest_vec)

# Evaluate

from sklearn.metrics import accuracy_score, classification_report

print("☑ Accuracy after tuning:", accuracy_score(y_test, y_pred))

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

output:

```
Best Parameters: {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
✅ Accuracy after tuning: 0.44984802431610943

📋 Classification Report:
              precision    recall  f1-score   support

         1.0       0.50      0.61      0.55       389
         2.0       0.37      0.29      0.33       397
         3.0       0.35      0.29      0.32       395
         4.0       0.40      0.42      0.41       397
         5.0       0.57      0.64      0.61       396

    accuracy                           0.45      1974
   macro avg       0.44      0.45      0.44      1974
weighted avg       0.44      0.45      0.44      1974
```