**Before hyperparameter tuning & model evaluation results**

**Key Observations:**

1. **Accuracy**: The overall accuracy is **0.6168**, which means about 62% of the test samples were correctly classified.
2. **Class Performance**:
   * **Precision** and **recall** are high for some classes (e.g., class 3), indicating good performance.
   * **Classes 1, 5, 7, 9, and 13** have very low or zero recall and precision, suggesting that the model struggles to classify these categories effectively.
3. **Macro vs. Weighted Average**:
   * **Macro average** (which averages metrics for each class without considering support/number of instances) is lower, showing that some classes are underperforming significantly.
   * **Weighted average** accounts for the number of instances per class, so it reflects overall model performance more fairly.
4. **High Precision but Low Recall**: Some classes (e.g., class 6 and 10) have high precision but low recall, indicating that while the model is correct when it predicts these classes, it often fails to identify all relevant instances.

**After hyperparameter tuning & model evaluation results**

The accuracy after hyperparameter tuning is approximately **59.35%**, with a detailed breakdown shown in the classification report. Here are some insights and recommendations based on these results:

**Key Insights:**

1. **Overall Accuracy**:
   * The test accuracy of **59.35%** indicates moderate performance, suggesting that while the model has been optimized through tuning, there may still be room for improvement.
2. **Class Performance**:
   * **High Performance (e.g., Class 3)**: Class 3 has strong performance with a high recall and f1-score, indicating that the model is good at identifying this category.
   * **Low Performance (e.g., Classes 5, 7, 9, and 13)**: These classes have poor or zero recall and precision, meaning the model struggles to correctly identify instances of these categories.
3. **Macro vs. Weighted Averages**:
   * The **macro average** shows a lower f1-score compared to the **weighted average**, which suggests that while some classes perform well, there are significant disparities in the model’s performance across different categories.
   * The **weighted average** gives more weight to classes with more instances, providing a better overview of overall performance.

Confusion Matrix

**Key Observations**

1. **Diagonal Values**:
   * The values on the diagonal (from the top left to the bottom right) represent the correctly classified instances for each class. Higher values along the diagonal indicate better performance for that class.
   * For instance, Class 6 has a high value of **49** in the diagonal, suggesting that the model performs well in correctly classifying instances of this class.
2. **Off-Diagonal Values**:
   * Non-zero values off the diagonal indicate misclassifications, where instances were assigned to an incorrect class.
   * For example, for True Label 3, we see values in columns 1, 2, 4, and others, showing that instances from Class 3 were sometimes misclassified as these other classes.
3. **Class-Specific Insights**:
   * **Class 3**: Shows **26** correctly classified instances but also some misclassifications spread across multiple classes (notably columns 2, 4, and 5). This indicates that while the model performs moderately well for Class 3, there is some confusion with similar classes.
   * **Class 6**: With **49** correctly classified instances and very few misclassifications, this class has the best performance in terms of classification accuracy.
   * **Class 7**: Shows a fair amount of confusion with neighboring classes, such as Class 6, with **12** correct predictions but several misclassifications, indicating that the model struggles more with this class.
   * **Classes 5, 9, and 13**: These classes have very few or no correct predictions, reflecting the challenges the model faces with these specific categories, possibly due to low support or overlapping features.
4. **Confusion Across Similar Categories**:
   * Some classes, such as 2 and 3 or 6 and 7, have noticeable misclassifications with each other. This could imply that these classes have similar features in the dataset, making them harder for the model to distinguish.

**Overall Model Performance**

* **High Accuracy for Certain Classes**: Classes like 3 and 6 show relatively high diagonal values, suggesting better performance in these categories.
* **Poor Performance for Low-Support Classes**: Classes with lower support, such as 5 and 13, have more misclassifications, which could affect the model's overall accuracy.
* **Class Imbalance**: The confusion matrix hints at class imbalance, with some classes having fewer correct predictions due to likely having fewer training instances.

Why Accuracy decreased after fine-tuning

**1. Overfitting on the Training Data**

* **Explanation**: The tuning process optimizes hyperparameters using cross-validation, which splits the training data into subsets. If the optimal parameters fit the cross-validation folds too well but fail to generalize, it can lead to overfitting.
* **Evidence**: The selected hyperparameters (C=10, kernel=linear, gamma=scale) might have been overly tuned to the training data patterns, leading to a model that performs worse on unseen test data.

**2. Poor Generalization of Selected Parameters**

* **Explanation**: The chosen hyperparameters may not generalize well to the test dataset due to differences in data distributions or imbalances.
* **Solution**: You can explore different ranges or combinations of hyperparameters. For instance:

python

Copy code

param\_grid = {

'C': [0.01, 0.1, 1, 10, 100],

'kernel': ['linear', 'rbf', 'poly'],

'gamma': ['scale', 'auto', 0.01, 0.1]

}

**3. Limited Cross-Validation Folds**

* **Explanation**: Using cv=5 splits the training data into five folds for validation. If the training dataset is small or imbalanced, this may not sufficiently represent the data variability, leading to suboptimal parameter selection.
* **Solution**: Use more folds (cv=10) or a stratified approach (e.g., StratifiedKFold) to ensure better representation of all classes in each fold.

**4. Class Imbalance in Data**

* **Explanation**: If certain classes dominate the dataset, GridSearchCV may choose hyperparameters that favor those classes at the expense of others. This can lead to a decline in accuracy for the minority classes.
* **Solution**: Adjust the scoring parameter in GridSearchCV to focus on a metric that accounts for class imbalance, such as f1\_weighted or balanced\_accuracy.

**5. Trade-off from Higher Regularization (C=10)**

* **Explanation**: In SVM, the C parameter controls the trade-off between a low-margin hyperplane and allowing misclassifications. A higher value of C (like 10) forces the model to fit the training data more closely, potentially at the cost of generalization.
* **Solution**: Test smaller values of C (e.g., 0.01 or 0.1) to see if a less complex model generalizes better.

**6. Linear Kernel Limitation**

* **Explanation**: The linear kernel assumes the data is linearly separable in the input space, which may not be true for your dataset. A non-linear kernel (e.g., rbf) might capture complex relationships better.
* **Solution**: Test non-linear kernels like rbf and poly to check if they improve accuracy.

**7. Potential Noise in the Dataset**

* **Explanation**: If there is noise in the data (e.g., mislabeled instances or inconsistent features), the hyperparameter tuning process can amplify the impact of this noise, leading to worse performance.
* **Solution**: Conduct a data quality check, clean noisy entries, or try feature selection to remove irrelevant features.