Model Optimization and Tuning Phase Report

Date	: **22-06-2025**
Team ID	: SWTID1749705847
Project Title	: **" Rising Waters: A Machine Learning Approach to Flood Prediction"**
Maximum Marks	: 10 Marks

Hyper-parameter Tuning Documentation:

MODEL	Tuned Hyperparameters

Decision Tree	`#  Decision Tree with GridSearchCV`
	`from sklearn.tree import DecisionTreeClassifier`
	
	`dt_classifier = DecisionTreeClassifier()`
	`param_grid_dt = {`
	`    'criterion': ['gini', 'entropy'],`
	`    'splitter': ['best', 'random'],`
	`    'max_depth': [None, 10, 20, 30, 40, 50],`
	`    'min_samples_split': [2, 5, 10],`
	`    'min_samples_leaf': [1, 2, 4]`
	`}`

Random Forest	`#  Random Forest with GridSearchCV`
	`from sklearn.ensemble import RandomForestClassifier`
	
	`rf_classifier = RandomForestClassifier()`
	`param_grid_rf = {`
	`    'n_estimators': [50, 100, 200],`
	`    'criterion': ['gini', 'entropy'],`
	`    'max_depth': [None, 10, 20, 30],`
	`    'min_samples_split': [2, 5, 10],`
	`    'min_samples_leaf': [1, 2, 4]`
	`}`

KNN

```python
#  KNN with GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier()
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}
```

Gradient Boosting

```python
#  Gradient Boosting with GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier

gb_classifier = GradientBoostingClassifier()
param_grid_gb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.8, 1.0]
}
```

Optimal Values

```python
grid_dt = GridSearchCV(estimator=dt_classifier, param_grid=param_grid_dt, cv=5)
grid_dt.fit(x_train, y_train)
y_pred_dt = grid_dt.predict(x_test)
print("Optimal Hyperparameters:", grid_dt.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
```

```
Optimal Hyperparameters: {'criterion': 'entropy', 'max_depth': 30, 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'random'}
Accuracy: 1.0
```

```python
grid_rf = GridSearchCV(estimator=rf_classifier, param_grid=param_grid_rf, cv=5)
grid_rf.fit(x_train, np.ravel(y_train))
y_pred_rf = grid_rf.predict(x_test)
print("Optimal Hyperparameters:", grid_rf.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Optimal Hyperparameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Accuracy: 0.9615384615384616
```

```python
grid_rf = GridSearchCV(estimator=rf_classifier, param_grid=param_grid_rf, cv=5)
grid_rf.fit(x_train, np.ravel(y_train))
y_pred_rf = grid_rf.predict(x_test)
print("Optimal Hyperparameters:", grid_rf.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Optimal Hyperparameters: {'n_neighbors': 5, 'p': 1, 'weights': 'uniform'}
Accuracy: 0.9615384615384616
```

```python
grid_knn = GridSearchCV(estimator=knn_classifier, param_grid=param_grid_knn, cv=5)
grid_knn.fit(x_train, np.ravel(y_train))
y_pred_knn = grid_knn.predict(x_test)
print("Optimal Hyperparameters:", grid_knn.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
```

```
Optimal Hyperparameters: {'learning_rate': 0.2, 'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 50, 'subsample'
Accuracy: 1.0
```

Performance Metrics Comparision Report:

<table>
<tr><th>Model</th><th>Optimized Metric</th></tr>
<tr><td>Decision Tree</td><td><pre>
print(classification_report(y_test, y_pred_dt))
print(confusion_matrix(y_test, y_pred_dt))

              precision   recall f1-score   support

           0   1.00   1.00   1.00   24
           1   1.00   1.00   1.00   2

    accuracy   1.00   26
   macro avg   1.00   1.00   1.00   26
weighted avg   1.00   1.00   1.00   26

[[24 0]
 [0 2]]
</pre></td></tr>
</table>

<table>
<tr><td>Random Forest</td><td>

```
print(classification_report(y_test, y_pred_rf))
print(confusion_matrix(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        24
           1       1.00      0.50      0.67         2

    accuracy                           0.96        26
   macro avg       0.98      0.75      0.82        26
weighted avg       0.96      0.96      0.96        26

[[24  0]
 [ 1  1]]
```

</td></tr>
<tr><td>KNN</td><td>

```
print(classification_report(y_test, y_pred_knn))
print(confusion_matrix(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        24
           1       0.67      1.00      0.80         2

    accuracy                           0.96        26
   macro avg       0.83      0.98      0.89        26
weighted avg       0.97      0.96      0.96        26

[[23  1]
 [ 0  2]]
```

</td></tr>
<tr><td>Gradient Boosting</td><td>

```
print(classification_report(y_test, y_pred_gb))
print(confusion_matrix(y_test, y_pred_gb))
```

</td></tr>
</table>

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        24
           1       1.00      1.00      1.00         2

    accuracy                           1.00        26
   macro avg       1.00      1.00      1.00        26
weighted avg       1.00      1.00      1.00        26

[[24  0]
 [ 0  2]]
```

Final Model Selection Justification:

Our Dataset is imbalanced.

Model	Accuracy	Recall (1)	F1 (1)
Decision Tree	1.00	1.00	1.00
Random Forest	0.96	0.50	0.67
KNN	0.96	1.00	0.80
Gradient Boosting	1.00	1.00	1.00

After tuning and evaluating all models, **Gradient Boosting Classifier** was selected as the final model.

- It achieved **100% accuracy** and **perfect scores (1.00)** in all metrics (precision, recall, F1) for both flood and non-flood classes.
- It **handled class imbalance** well and ensured **zero false negatives**, which is crucial in flood detection.
- Compared to Random Forest and KNN, it performed better on the rare flood cases.
- Unlike Decision Tree, it is **less prone to overfitting** and combines multiple learners for stronger performance.
- Gradient Boosting provides **better control over learning rate and depth**, making it ideal for **future improvements**.

In real-world flood prediction, **missing a flood is riskier than a false alarm**, and this model gives the **most reliable and safe predictions**.