**Title for the Report:**

**"Forest Cover Type Prediction Using Random Forest Classifier: A Machine Learning Approach to Environmental Data Classification"**

**Introduction**

This project aims to develop a machine learning model capable of predicting the forest cover type based on various environmental features such as elevation, distance to hydrology, slope, and others. The project is based on supervised learning using classification techniques, particularly focusing on a Random Forest Classifier. The dataset used for this project contains several features related to geographical and environmental data, and the goal is to predict which type of forest cover is present at a particular location.

**1. Problem Definition**

The main problem is to predict the forest cover type, given a set of environmental features. There are seven different forest cover types to classify, as defined by the dataset. The cover types are:

1. **Spruce/Fir**
2. **Lodgepole Pine**
3. **Ponderosa Pine**
4. **Cottonwood/Willow**
5. **Aspen**
6. **Douglas-fir**
7. **Krummholz**

Each record in the dataset represents a specific geographical point with several numerical attributes like elevation, slope, and distances to hydrological or road features. Our task is to build a predictive model that can learn from these features and classify the data points into one of the seven cover types.

**2. Dataset Overview**

The dataset is structured with several features that are critical for the prediction task:

**Key Features:**

* **Elevation**: Elevation of the point in meters.
* **Aspect**: Compass direction the slope faces in degrees.
* **Slope**: Slope of the point in degrees.
* **Horizontal\_Distance\_To\_Hydrology**: Horizontal distance to the nearest surface water feature.
* **Vertical\_Distance\_To\_Hydrology**: Vertical distance to the nearest surface water feature.
* **Horizontal\_Distance\_To\_Roadways**: Horizontal distance to the nearest roadway.
* **Hillshade\_9am, Hillshade\_Noon, Hillshade\_3pm**: Hillshade indices at different times of day.
* **Horizontal\_Distance\_To\_Fire\_Points**: Horizontal distance to the nearest wildfire ignition point.

The target variable is **Cover\_Type**, which represents the type of forest cover.

**3. Workflow Breakdown**

**a) Data Preprocessing (preprocess.py)**

Preprocessing is one of the most critical stages in machine learning projects. The raw dataset cannot be directly fed into a machine learning model, so several steps are applied:

* **Data Loading**: The dataset is loaded from a CSV file using pandas.

python

Copy code

def load\_data(filepath):

return pd.read\_csv(filepath)

* **Scaling Continuous Features**: Continuous features like Elevation, Slope, Horizontal\_Distance\_To\_Hydrology, etc., need to be standardized to ensure the model can effectively learn from them. StandardScaler is used to scale these features so that each has a mean of 0 and a standard deviation of 1.

python

Copy code

def scale\_features(X):

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

return X\_scaled

* **Preprocessing Pipeline**:
  + Features (X) and the target (y) are separated.
  + The continuous features are scaled.
  + The transformed data is returned for further model training.

This step ensures that the data is well-prepared for the machine learning model by handling feature scaling and separating features from the target variable.

python

Copy code

def preprocess\_data(data):

X = data.drop('Cover\_Type', axis=1)

y = data['Cover\_Type']

continuous\_features = [

'Elevation',

'Slope',

'Horizontal\_Distance\_To\_Hydrology',

'Vertical\_Distance\_To\_Hydrology',

'Horizontal\_Distance\_To\_Roadways'

]

# Scaling the continuous features

X\_scaled = scale\_features(X[continuous\_features])

X[continuous\_features] = X\_scaled

return X, y

**b) Model Training (trainmodel.py)**

After preprocessing the data, the next step is to train the machine learning model. This step involves splitting the data into training and testing sets, training the model on the training set, and evaluating it on the test set.

* **Train-Test Split**: The dataset is split into 70% training data and 30% testing data using train\_test\_split from Scikit-learn.

python

Copy code

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

* **Model Choice**: A Random Forest Classifier is chosen for this task due to its effectiveness with structured data and ability to handle large datasets with many features.

python

Copy code

model = RandomForestClassifier()

model.fit(X\_train, y\_train)

* **Model Evaluation**: After training, predictions are made on the test data, and performance metrics are calculated. The classification\_report function generates precision, recall, and F1-score for each class (forest cover type).

python

Copy code

predictions = model.predict(X\_test)

print(classification\_report(y\_test, predictions))

* **Confusion Matrix**: A confusion matrix is plotted to visualize the model’s classification performance.

python

Copy code

conf\_matrix = confusion\_matrix(y\_test, predictions)

sns.heatmap(conf\_matrix, annot=True, fmt="d")

plt.show()

**c) Model Tuning (Hyperparameter Optimization)**

To further improve the model's performance, hyperparameter tuning is performed using RandomizedSearchCV. This technique searches for the best combination of parameters by randomly sampling from a parameter grid.

* **Parameter Grid**: A dictionary specifying possible values for key hyperparameters like the number of trees (n\_estimators), maximum tree depth (max\_depth), and minimum samples required to split a node (min\_samples\_split).

python

Copy code

param\_grid = {

'n\_estimators': [100, 200, 300],

'max\_depth': [10, 20, 30],

'min\_samples\_split': [2, 5, 10],

}

* **Random Search**: The best parameters are found through cross-validation, and the model is refit with the optimal parameters.

python

Copy code

rf\_random = RandomizedSearchCV(estimator=model, param\_distributions=param\_grid, cv=3)

rf\_random.fit(X\_train, y\_train)

**d) Model Testing on New Data (testmodel.py)**

Once the model is tuned and finalized, it is tested on a new, unseen test dataset. This step mimics the real-world deployment of the model.

* **Test Data Preprocessing**: Similar to training data, the test data is preprocessed to ensure it is in the correct format for prediction.

python

Copy code

X\_test\_new, \_ = preprocess\_data(test\_data)

* **Predictions**: The saved model (final\_model.pkl) is loaded, and predictions are made on the new test data.

python

Copy code

model = load('models/final\_model.pkl')

predictions = model.predict(X\_test\_new)

* **Mapping Predictions**: The numeric predictions (1-7) are mapped to their respective forest cover type names (e.g., Spruce/Fir, Lodgepole Pine, etc.).

python

Copy code

cover\_type\_mapping = {1: "Spruce/Fir", 2: "Lodgepole Pine", ...}

* **Saving Predictions**: The final predictions are saved to a CSV file for further analysis or submission.

python

Copy code

output = pd.DataFrame({'Id': test\_data.index, 'Cover\_Type': predicted\_cover\_types})

output.to\_csv('predictions.csv', index=False)

**4. Results and Evaluation**

The Random Forest Classifier performed well on the classification task. The following metrics were used to evaluate the model:

* **Accuracy**: The percentage of correctly predicted forest cover types.
* **Precision, Recall, F1-score**: Detailed metrics for each class to evaluate the performance for each cover type.
* **Confusion Matrix**: A matrix to visualize the distribution of true positives and false positives across the seven cover types.

After hyperparameter tuning, the model's performance improved, and the best set of hyperparameters was identified. The final model was evaluated on the test dataset, and predictions were saved for further use.

**5. Conclusion**

This project successfully developed a machine learning pipeline for predicting forest cover types based on geographical and environmental features. The project followed a structured workflow, starting from data preprocessing, model training, evaluation, and hyperparameter tuning, to finally testing the model on new data. Random Forest was chosen as the classification model due to its robustness and accuracy in handling structured data.

This pipeline can now be deployed for real-world applications in forestry management or environmental studies where predicting forest cover types is crucial.