Part 1

Q1 solution: To calculate precision, recall, F-measure, and accuracy, we need to define,

True Positive (TP): occurrence of the model correctly predicts the positive class, Instances where Prediction=True and Gold=True.

True Negative (TN): Instances where the model correctly predicts the negative class, Instances where Prediction=True and Gold=False

False Positive (FP): Instances where the model incorrectly predicts the positive class, Instances where Prediction=False and Gold=False

False Negative (FN): Instances where the model incorrectly predicts the negative class, Instances where Prediction=False and Gold=True

Precision: Proportion given by true positive predictions out of all positive predictions.
Recall: Out of all actual positive instances we check proportion of true positive. predictions.
F-measure: The harmonic mean of precision and recall.
Accuracy: Proportion of correct predictions out of all predictions.

Let's calculate each of these metrics:

Precision = TP / (TP + FP)
Recall = TP / (TP + FN)
F-measure = 2 * (Precision * Recall) / (Precision + Recall)
Accuracy = (TP + TN) / (TP + TN + FP + FN)

Now, let's calculate step by step:

True Positives (TP): 6
False Positives (FP): 3
True Negatives (TN): 7
False Negatives (FN): 4

Precision = 6 / (6 + 3) = 6 / 9 = 0.67

Recall = 6 / (6 + 4) = 6 / 10 = 0.6
F-measure = 2 * (0.67 * 0.6) / (0.67 + 0.6) = 2 * (0.402) / 1.27= 0.633
Accuracy = (6 + 7) / (6 + 7 + 3 + 4) = 13 / 20 = 0.65

So, the manually calculated metrics are:

Precision: 0.67
Recall: 0.6
F-measure: 0.633
Accuracy: 0.65

Q2 Solution:

Regression python code

```
# Regression

import pandas as pd

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

from sklearn.model_selection import GridSearchCV


# Loading the training and testing datasets


train_data = pd.read_csv("../Part 1/real-state/train_full_Real-estate.csv")

test_data = pd.read_csv("../Part 1/real-state/test_full_Real-estate.csv")


# Preparing data for regression task

X_train_reg = train_data.drop(columns=['Y house price of unit area'])

y_train_reg = train_data['Y house price of unit area']

X_test_reg = test_data.drop(columns=['Y house price of unit area'])

y_test_reg = test_data['Y house price of unit area']
```

```
# Defining the hyperparameter grid
param_grid = {
'fit_intercept': [True, False],
'copy_X': [True, False],
'positive': [True, False]
}

# Initializing GridSearchCV
grid_search = GridSearchCV(LinearRegression(), param_grid, cv=5, scoring='neg_mean_squared_error')

# Performing hyperparameter tuning
grid_search.fit(X_train_reg, y_train_reg)

# Geting the best hyperparameters
best_params = grid_search.best_params_

# Training Linear Regression model with the best hyperparameters
best_linear_reg_model = LinearRegression(**best_params)
best_linear_reg_model.fit(X_train_reg, y_train_reg)

# Predicting with the best model
y_pred_reg_best = best_linear_reg_model.predict(X_test_reg)

# Calculating RMSE with the best model
rmse_reg_best = mean_squared_error(y_test_reg, y_pred_reg_best, squared=False)

# Printing the best hyperparameters and the corresponding RMSE
print("Best hyperparameters:", best_params)
print("Regression - Root Mean Squared Error (RMSE) with best hyperparameters:", rmse_reg_best)
```

Output:

Best hyperparameters: {'copy_X': True, 'fit_intercept': False, 'positive': False} Regression - Root Mean Squared Error (RMSE) with best hyperparameters: 8.602718711928972

Python code description

Using pandas, LinearRegression, mean_squared_error, and GridSearchCV, and reading the training and testing data from csv files, performed grid search and provided best hyperparameters found to the linear regression model.

Classification python code

```
# Classification

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import GridSearchCV

from sklearn.metrics import accuracy_score


# Loading the training and testing datasets

train_data = pd.read_csv("../Part 1/real-state/train_full_Real-estate.csv")

test_data = pd.read_csv("../Part 1/real-state/test_full_Real-estate.csv")


# Preparing data for classification task

X_train_cls = train_data.drop(columns=['Y house price of unit area'])

y_train_cls = train_data['Y house price of unit area'] >= 30 # Convert to binary labels

X_test_cls = test_data.drop(columns=['Y house price of unit area'])

y_test_cls = test_data['Y house price of unit area'] >= 30 # Convert to binary labels


# Defining the hyperparameter grid

param_grid = {
```

```python
'n_estimators': [100, 200, 300],

'max_depth': [None, 10, 20],

'min_samples_split': [2, 5, 10],

'min_samples_leaf': [1, 2, 4],

'max_features': ['auto', 'sqrt', 'log2']

}


# Creating the grid search model

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')


# Performing hyperparameter tuning

grid_search.fit(X_train_cls, y_train_cls)


# Getting the best hyperparameters

best_params = grid_search.best_params_

print("Best hyperparameters:", best_params)


# Predicting with the best model

best_model = grid_search.best_estimator_

y_pred_cls_rf = best_model.predict(X_test_cls)


# Calculating the accuracy

accuracy_cls_rf = accuracy_score(y_test_cls, y_pred_cls_rf)

print("Random Forest Classification - Accuracy with best hyperparameters:", accuracy_cls_rf)
```

Output

Best hyperparameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 100}

Random Forest Classification - Accuracy with best hyperparameters: 0.8938053097345132

Python code description

Using pandas, RandomForestClassifier, GridSearchCV, and accuracy_score, reading data from csv files and performing the grid search, provided best hyperparameters to the random forest classifier model.