DATA MINING - HW1

G# : G01445209
Miner Username: Inataray

STUDENT PERFORMANCE PREDICTION USING KNN

AIM:

The aim of this assignment is to implement a k-Nearest Neighbor (KNN) Classifier to predict students' performance based on a set of 36 variables. The task involves classifying students into one of three classes:

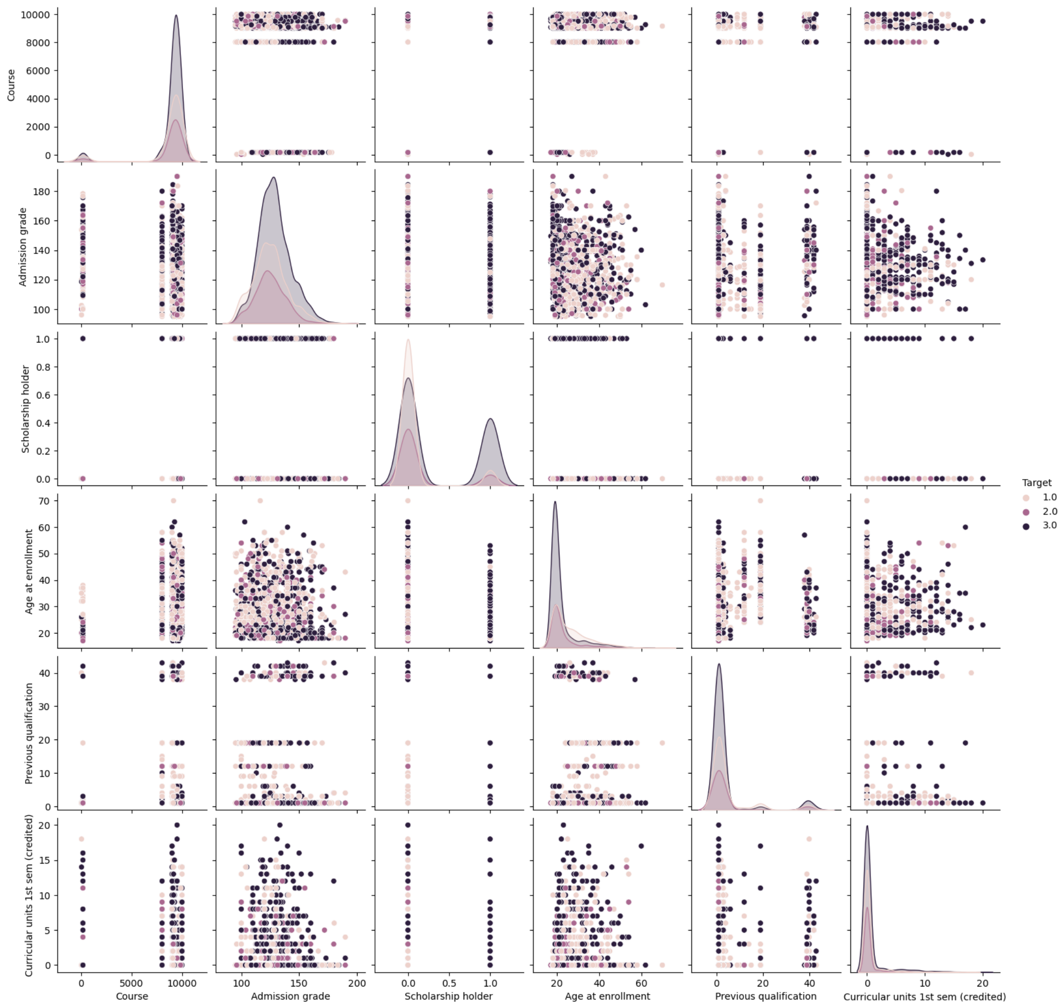
Class 1: Dropout
Class 2: Enrolled
Class 3: Graduate

To achieve this aim, following steps are required:

DATA PREPROCESSING:

1. Fetch the training file data and check if the data is clean and formatted correctly.

2. Analyze the data and plot some of the attributes and target in graph to check if both are correlated.



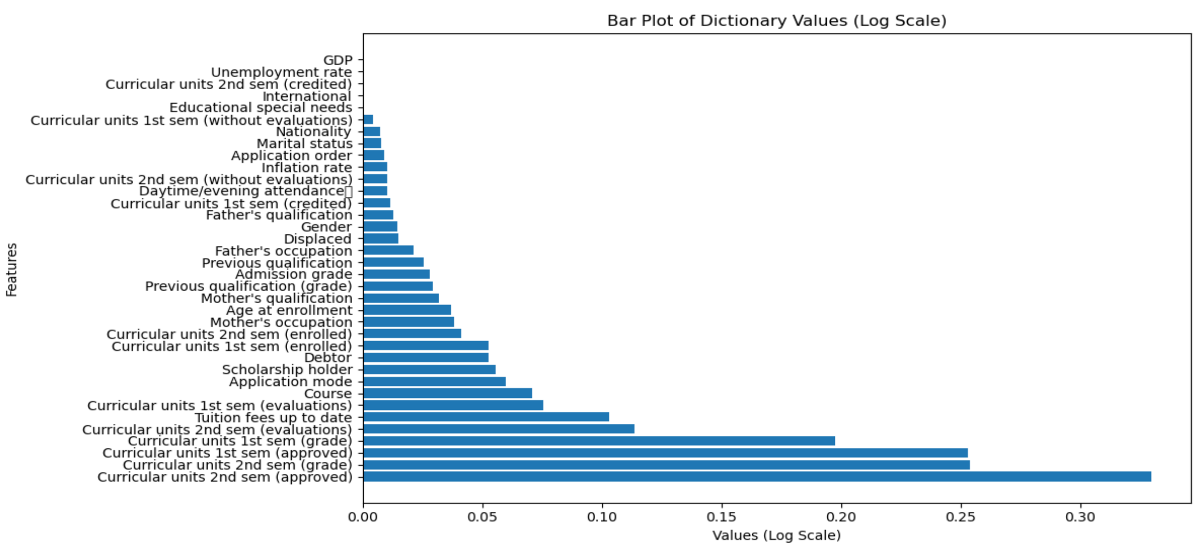
3. Split the dataset into attributes and target class.

```
# Splitting into labels and attributes
X = dataset[:, :-1]
Y = dataset[:, -1]
```

4. I trained the model with whole dataset since it is clean and formatted but accuracy is not expected so I did feature reduction, Encoding and K-fold cross validation.

5. Feature Reduction using Mutual Information technique.

```
mutual_information = mutual_info_classif(X_train, Y_train)
column_mf_mapping = {}
for i in range(len(cols)):
    column_mf_mapping[cols[i]] = mutual_information[i]
sorted_mf = dict(sorted(column_mf_mapping.items(), key=lambda
item: item[1], reverse=True))
keys = list(sorted_mf.keys())
values = list(sorted_mf.values())
plt.figure(figsize=(12,6))
plt.barh(keys,values)
plt.xlabel('Values (Log Scale)')
plt.ylabel('Features')
plt.title('Bar Plot of Dictionary Values (Log Scale)')
plt.tight_layout()
plt.show()
```



6. One Hot Encoding using pandas :

```
df_encoded = pd.get_dummies(reduced_data, columns=['Marital
status'], prefix=['Marital status'])
```

7. Normalizing data using StandardScalar() sklearn function.

BUILDING MODEL:

1. __init__(self, k): Initializes the KNN classifier with a specified value of k (number of neighbors).

2. fit_data(self, X_train, Y_train): Fits the training data to the model.

3. euclidean_distance(self, x, y): Calculates the Euclidean distance between two data points.

4. manhattan_distance(self, x, y): Calculates the Manhattan distance between two data points.

5. predict_data(self, X): Predicts the labels for a set of input data points.

6. find_k_nearest_neighbors(self, distances): Identifies the k-nearest neighbors based on calculated distances.

7. find_y_predicted(self, k_nearest_neighbors): Predicts the class label based on the k-nearest neighbors.

8. prediction(self, x): Implements the KNN pipeline to predict the label for a single input data point.

CROSS VALIDATION:

1. After experimenting with various sets of features, I have determined that the best performance is achieved when using the top 29 features.

2. Following my exploration with different values of k, I have identified that a k value of 12 provides optimal performance.

3. I tried with euclidean distance and manhattan distance. Euclidean distance gives the best result rather than Manhattan distance.

4. After performing k-fold cross-validation to select the training set, I determined that utilizing 10 folds with the 6th fold as the training set yielded the most favorable outcome.

```
def k_fold_cross_validation(X, Y, folds):

    ''' *** K- fold selecting code *** '''
    fold_size = len(X) // folds # calculating the fold size
    #initailizing the folds
    x_train_folds = []
    x_test_folds = []
    y_train_folds = []
    y_test_folds = []

    for i in range(folds):
        #calculating the indexes
        start_index = i * fold_size
        end_index = (i+1) * fold_size if i < (folds - 1) else
len(X)
        # fetch and update the values based on the index
        x_test_set = X[start_index:end_index]
        x_train_set = np.concatenate((X[:start_index],
X[end_index:]))
        y_test_set = Y[start_index:end_index]
        y_train_set = np.concatenate((Y[:start_index],
Y[end_index:]))
        x_train_folds.append(x_train_set)
        x_test_folds.append(x_test_set)
        y_train_folds.append(y_train_set)
        y_test_folds.append(y_test_set)
    return x_train_folds, x_test_folds, y_train_folds,
y_test_folds
```

TESTING & RESULTS:

1. The accuracy of a classification model can be calculated using the confusion matrix with the following formula:

Accuracy = (True Positives + True Negatives) / Total Predictions

2. I tested the data using the test set, employing the 6th fold data as the training set, and achieved an accuracy rate of 71%.