ASSIGNMENT

Problem 4: Fraud Detection in Financial Transactions

Scenario:

A financial institution wants to develop an algorithm to detect fraudulent transactions in real time.

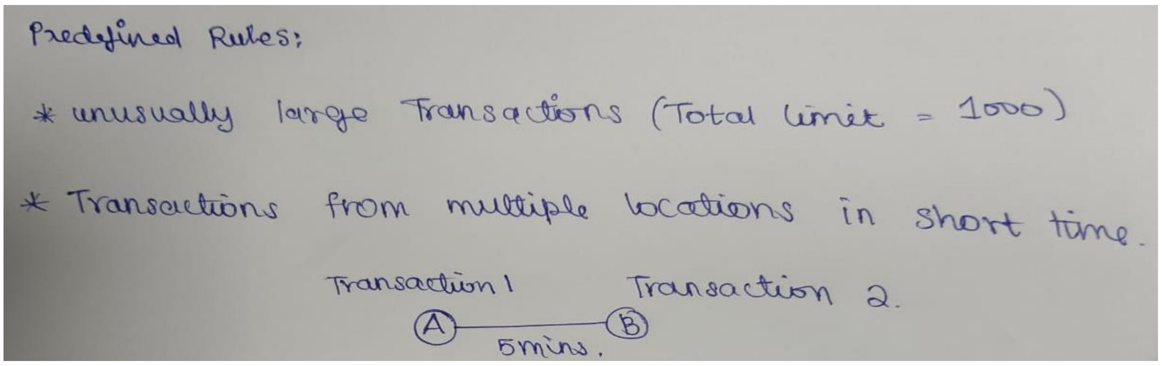
Tasks:

1. Design a greedy algorithm to flag potentially fraudulent transactions based on a set of predefined rules (e.g., unusually large transactions, and transactions from multiple locations in a short time).
2. Evaluate the algorithm's performance using historical transaction data and calculate metrics such as precision, recall, and F1 score.
3. Suggest and implement potential improvements to the algorithm.

SOLUTION:

TASK 1: Design a greedy algorithm to flag potentially fraudulent transactions based on a set of predefined rules

In this problem, I have used the Basic greedy approach and statistical formulas to predict fraud in money transactions. In addressing the problem of fraud detection in financial transactions, I have devised a greedy algorithm based on predefined rules. This algorithm flags potentially fraudulent transactions by identifying unusually large transactions and transactions occurring in multiple locations within a short timeframe.



Predefined Rules:

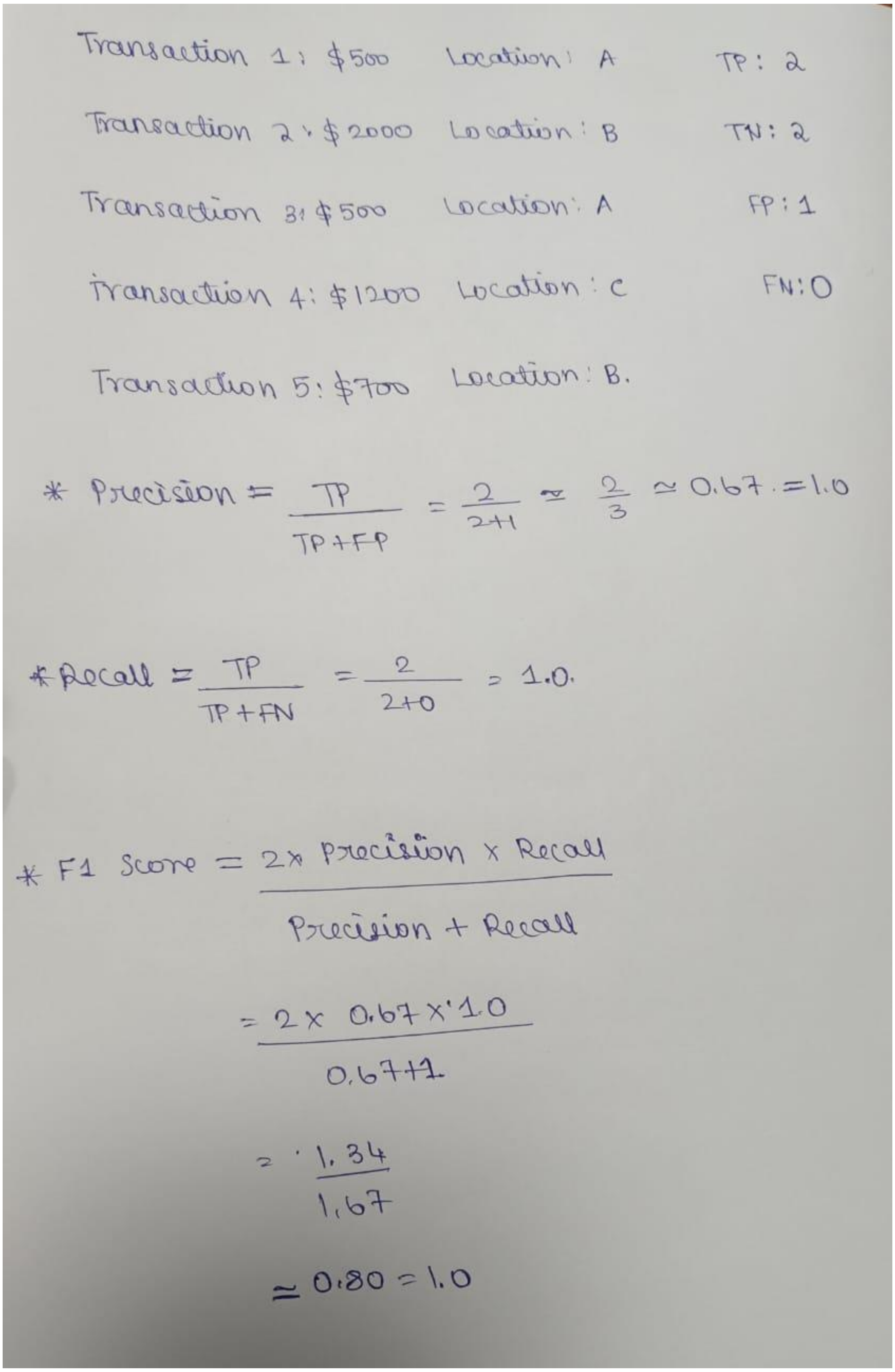
* unusually large Transactions (Total limit = 1000)

* Transactions from multiple locations in short time.

Transaction 1
Transaction 2.
A
B
5mins.

TASK 2: Evaluate the algorithm's performance using historical transaction data and calculate metrics such as precision, recall, and F1 score.

Five transactions are considered as the input data for the program. The program has predefined whether the transaction is fraudulent or not. The data contains the amount and location of the transactions. Parameters such as Precision, recall and F1 score are calculated using true positive(Transactions that are correctly predicted as fraudulent), true negative(Transactions that are correctly predicted as legitimate), false positive(Transactions that are incorrectly predicted as fraudulent) and false negative(Transactions that are incorrectly predicted as legitimate).



Transaction 1: \$500	Location: A	TP: 2
Transaction 2: \$2000	Location: B	TN: 2
Transaction 3: \$500	Location: A	FP: 1
Transaction 4: \$1200	Location: C	FN: 0
Transaction 5: \$700	Location: B.	

* Precision $= \frac{TP}{TP+FP} = \frac{2}{2+1} \approx \frac{2}{3} \simeq 0.67 = 1.0$

* Recall $= \frac{TP}{TP+FN} = \frac{2}{2+0} = 1.0.$

* F1 Score $= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

$$= \frac{2 \times 0.67 \times 1.0}{0.67+1}$$

$$= \frac{1.34}{1.67}$$

$$\simeq 0.80 = 1.0$$

IMPLEMENTATION:

```
class FraudDetection:
    def __init__(self, maxamount, location):
        self.maxamount = maxamount
        self.location = location

    def fraud(self, transaction):
        if transaction['amount'] > self.maxamount:
            return True
        rhistory = transaction['recent_transactions']
        locations = set(t['location'] for t in rhistory)
        if len(locations) > 1 and (transaction['timestamp'] - min(t['timestamp'] for t in rhistory)).seconds
< self.location:
            return True
        return False

    def evaluate(self, history):
        tp = 0
        fp = 0
        fn = 0
        tn = 0

        for transaction in history:
            prediction = self.fraud(transaction)
            actual = transaction['fraud']

            if prediction and actual:
                tp += 1
            elif prediction and not actual:
                fp += 1
            elif not prediction and actual:
                fn += 1
            elif not prediction and not actual:
                tn += 1

        precision = tp / (tp + fp) if (tp + fp) > 0 else 0
        recall = tp / (tp + fn) if (tp + fn) > 0 else 0
        f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

        return precision, recall, f1

history = [
    {'amount': 500, 'location': 'A', 'timestamp': '2024-06-27 10:00', 'fraud': False, 'recent_transactions': []},
    {'amount': 2000, 'location': 'B', 'timestamp': '2024-06-27 10:05', 'fraud': True, 'recent_transactions': [{'amount': 500, 'location': 'A', 'timestamp': '2024-06-27 10:00'}]},
    {'amount': 500, 'location': 'A', 'timestamp': '2024-06-27 10:00', 'fraud': False, 'recent_transactions': []},
    {'amount': 1200, 'location': 'C', 'timestamp': '2024-06-27 10:10', 'fraud': True, 'recent_transactions': [{'amount': 600, 'location': 'A', 'timestamp': '2024-06-27 10:05'}]},
    {'amount': 700, 'location': 'B', 'timestamp': '2024-06-27 10:15', 'fraud': False, 'recent_transactions': []},
]

detector = FraudDetection(maxamount=1000, location=300)
precision, recall, f1 = detector.evaluate(history)

print(f'Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}')
```

OUTPUT:

```
Precision: 1.00, Recall: 1.00, F1 Score: 1.00
```

PSEUDOCODE:

```
FOR each transaction in sortedhistory:
      SET prediction = self.fraud(transaction)
      SET actual = transaction['fraud']

      IF prediction AND actual:
        INCREMENT tp
      ELIF prediction AND NOT actual:
        INCREMENT fp
      ELIF NOT prediction AND actual:
        INCREMENT fn
      ELIF NOT prediction AND NOT actual:
        INCREMENT tn

    SET precision = tp / (tp + fp) IF (tp + fp) > 0 ELSE 0
    SET recall = tp / (tp + fn) IF (tp + fn) > 0 ELSE 0
    SET f1 = 2 * (precision * recall) / (precision + recall) IF (precision + recall) > 0 ELSE 0

    RETURN precision, recall, f1
```

TASK 3: Suggest and implement potential improvements to the algorithm.

POTENTIAL IMPROVEMENTS:

1. Tune the threshold values:

   The max amount and location thresholds can be adjusted to improve the accuracy of the fraud detection algorithm.

2. Use machine learning algorithms:

   Consider using machine learning algorithms like decision trees, random forests, or neural networks to improve the accuracy of the fraud detection algorithm.

3. Include additional features:

   Add more features to the transaction data, such as user behavior, IP address, and device information, to improve the accuracy of the fraud detection algorithm.

4. Use anomaly detection:

Implement anomaly detection techniques to identify unusual patterns in the transaction data that may indicate fraud.

ALTERNATIVE ALGORITHMS:

1. Multi-Stage Greedy Algorithm:

- Instead of evaluating all rules at once, it evaluates them in stages.
- This staged approach can prioritize the most critical checks first.

2. Weighted Greedy Algorithm:

- Assign weights to different rules based on their importance or historical effectiveness.
- Calculate a weighted score for each transaction based on the rules it violates.

3. Greedy Algorithm with Historical Comparison:

- Compare each transaction not just against predefined rules but also against historical data.
- Flag transactions that deviate significantly from the user's historical transaction patterns. This can be done by maintaining a rolling history of transactions and continuously updating the comparison baseline.

4. Context-Aware Greedy Algorithm:

- Incorporates additional contextual information like user profile, location, and time.
- Adjust the evaluation criteria based on context. For example, a large transaction might not be flagged if it's at a known high-spending location for the user.