Exam 1 IA

Erick Gonzalez Parada #178145

Theoretical part was handed on paper

Full code & files: https://github.com/HugeErick/ia/tree/main/exam1

⚠️ IMPORTANT: The code has no comments:

I decided to not write comments in the codes shown by this exam due to LLM's code generation confusion **ALL THE CODE IS DOCUMENTED IN A SEPARATE TXT FILE**

Exercise 1 Datasets

Dataset 1

What problem(s) appeared in this dataset?

The dataset contains missing values represented by ? and/or just straight up missing values and some empty fields and there is a evident data imbalance in the num column.

How would I identify the problem in python?

The basic answer is just compute the values to see what there is at any point and do something about but just in case I build a Problem Fixer of csv files, see below dataset questions.

How to fix this error(s)?

For the missing values one can replace the ? with NaN but of course that's not ideal either, corresponding values must be there, for the data imbalance just apply SMOTE with oversampling or undersampling.

How would this problem affect a machine learning model?

It can give wrong, false, even break or just operate poorly for the benefit of the label(s).

Is it ready to use?

no

Dataset 2

What problem(s) appeared in this dataset?

The dataset contains missing values represented by ? and/or just straight up missing values and some empty fields and there is a evident data imbalance in the num column.

How would I identify the problem in python?

The basic answer is just compute the values to see what there is at any point and do something about but just in case I build a Problem Fixer of csv files, see below dataset questions.

How to fix this error(s)?

For the missing values one can replace the ? with NaN but of course that's not ideal either, corresponding values must be there, for the data imbalance just apply SMOTE with oversampling or undersampling.

How would this problem affect a machine learning model?

It can give wrong, false, even break or just operate poorly for the benefit of the label(s).

Is it ready to use?

no

Dataset 3

What problem(s) appeared in this dataset?

Unrealistic numbers for several columns.

How would I identify the problem in python?

In this case if one know the range values from the columns it is just easy as checking each values with python.

How to fix this error(s)?

if it was a typo then don’t typo and just verify realistic values

How would this problem affect a machine learning model?

The data if some Logistic regression is applied it can perform very poorly when data is not balanced

Is it ready to use?

yes if those big numbers get removed

Dataset 4

What problem(s) appeared in this dataset?

Unrealistic numbers for several columns.

How would I identify the problem in python?

In this case if one know the range values from the columns it is just easy as checking each values with python.

How to fix this error(s)?

if it was a typo then don’t typo and just verify realistic values

How would this problem affect a machine learning model?

The data if some Logistic regression is applied it can perform very poorly when data is not balanced

Is it ready to use?

yes if those big numbers get removed

Dataset 5

What problem appeared in this dataset?

n/a

How would I identify the problem in python?

n/a

How to fix this error(s)?

n/a

How would this problem affect a machine learning model?

n/a

Is it ready to use?

yes

Code for the previous part

```
# entryToFixCSV.py
from fixCSV import validateAndFixCsv

issues, cleaned_file = validateAndFixCsv('exam1/data.csv')
for issue in issues:
    print(issue)
```

```
# fixCSV.py
import pandas as pd
import numpy as np
from collections import Counter
import re
from sklearn.utils import resample
import os

def validateAndFixCsv(filePath):
    try:
        df = pd.read_csv(filePath)
        issues = []

        def handleNullValues(dataFrame):
            nullCount = dataFrame.isnull().sum()
            if nullCount.any():
                numericCols = dataFrame.select_dtypes(include=[np.number]).column
                categoricalCols = dataFrame.select_dtypes(include=['object']).columns

                dataFrame[numericCols] = dataFrame[numericCols].fillna(dataFrame[n
                dataFrame[categoricalCols] = dataFrame[categoricalCols].fillna(dataFra
                issues.append(f"Found and fixed null values in columns: {nullCount[nul
            return dataFrame

        def checkClassBalance(dataFrame):
            categoricalCols = dataFrame.select_dtypes(include=['object']).columns
            for col in categoricalCols:
                classDistribution = Counter(dataFrame[col].dropna())
                if len(classDistribution) > 1:
                    majorityClass = max(classDistribution.values())
                    minorityClass = min(classDistribution.values())
                    if majorityClass / minorityClass > 3:
                        issues.append(f"Class imbalance detected in column {col}")
                        majorityClasses = [k for k, v in classDistribution.items() if v == maj
                        minorityClasses = [k for k, v in classDistribution.items() if v == min

                        for minClass in minorityClasses:
                            minorityDf = dataFrame[dataFrame[col] == minClass]
                            upsampledMinority = resample(minorityDf,
                                                         replace=True,
                                                         n_samples=majorityClass)
                            dataFrame = pd.concat([dataFrame[dataFrame[col] != minClass]
            return dataFrame

        def cleanSpecialCharacters(dataFrame):
            for col in dataFrame.columns:
                if dataFrame[col].dtype == 'object':
                    hasSpecialChars = dataFrame[col].astype(str).str.contains(r'[^a-zA-Z
                    if hasSpecialChars:
                        dataFrame[col] = dataFrame[col].astype(str).apply(lambda x: re.su
                        issues.append(f"Cleaned special characters in column {col}")
            return dataFrame

        def validateLabels(dataFrame):
            invalidChars = r'[^a-zA-Z0-9_]'
            originalCols = dataFrame.columns.tolist()
```

```
            newCols = [re.sub(invalidChars, '_', col) for col in originalCols]

            if originalCols != newCols:
                dataFrame.columns = newCols
                issues.append("Fixed invalid characters in column names")

            return dataFrame

        df = handleNullValues(df)
        df = checkClassBalance(df)
        df = cleanSpecialCharacters(df)
        df = validateLabels(df)

        if not issues:
            issues.append("No issues found. Data is good to go!")

        dirPath = os.path.dirname(filePath)
        fileName = os.path.basename(filePath)
        baseName, ext = os.path.splitext(fileName)
        counter = 1
        outputPath = os.path.join(dirPath, f"{baseName}_cleaned{ext}")

        while os.path.exists(outputPath):
            outputPath = os.path.join(dirPath, f"{baseName}_cleaned_{counter}{ext}"
            counter += 1

        df.to_csv(outputPath, index=False)

        return issues, outputPath

    except Exception as e:
        return [f"Error processing file: {str(e)}"], None
```

DOCUMENTATION: https://github.com/HugeErick/ia/blob/main/exam1/csv-cleaner-documentation.md

Exercise 2 Metrics

Case 1

1. **Precision (0.87):**
    - The model correctly predicts 87% of the positive cases (correct transcriptions) out of all predicted positive cases.
    - This is a mediocre precision score, indicating that the model is almost reliable when it predicts a correct transcription.
2. **Recall (0.83):**
    - The model identifies 83% of the actual positive cases (correct transcriptions) out of all actual positive cases.
    - This is a somewhat decent recall score, but there is room for improvement to reduce false negatives (missed correct transcriptions).
3. **Accuracy (0.90):**
    - The model is correct 90% of the time overall.
    - This is a “decent” accuracy score, indicating that the model performs well in general.
4. **F1-score (0.85):**
    - The F1-score balances precision and recall. A score of 0.85 is good, indicating a strong balance between precision and recall.
5. **Confusion Matrix Analysis:**
    - **True Positives (TP):** 420 (correctly predicted correct transcriptions).
    - **False Positives (FP):** 40 (incorrectly predicted correct transcriptions).
    - **False Negatives (FN):** 80 (missed correct transcriptions).
    - **True Negatives (TN):** 460 (correctly predicted incorrect transcriptions).
    - The model has a low number of false positives and false negatives, which is good. However, there is still room to reduce the number of false negatives (80) to improve recall.

Case 2

1. **Precision (0.80):**
    - The model correctly predicts 80% of the positive cases (machine failures) out of all predicted positive cases.
    - This is a mediocre precision score, but there is room for improvement to reduce false positives (incorrectly predicted failures).
2. **Recall (0.70):**
    - The model identifies 70% of the actual positive cases (machine failures) out of all actual positive cases.
    - This is a just not good enough recall score, indicating that the model misses 30% of actual failures, which could be problematic in a predictive maintenance scenario.
3. **Accuracy (0.85):**
    - The model is correct 85% of the time overall.
    - This is a strong accuracy score, but it may be misleading due to class imbalance (more non-failures than failures).
4. **F1-score (0.75):**
    - The F1-score balances precision and recall. A score of 0.75 is mediocre and could be improved, especially by increasing recall.
5. **Confusion Matrix Analysis:**
    - **True Positives (TP):** 140 (correctly predicted failures).
    - **False Positives (FP):** 90 (incorrectly predicted failures).
    - **False Negatives (FN):** 60 (missed failures).
    - **True Negatives (TN):** 710 (correctly predicted non-failures).
    - The model has a relatively high number of false positives (90) and false negatives (60). Reducing false negatives is critical in predictive maintenance to avoid missing actual failures.

Case 3

1. **Precision (0.78):**

- The model correctly predicts 78% of the positive cases (correct recommendations) out of all predicted positive cases.
- This is a horrible precision score, and there is room for improvement to reduce false positives (incorrect recommendations).

2. **Recall (0.65):**
    - The model identifies 65% of the actual positive cases (correct recommendations) out of all actual positive cases.
    - This is an awful recall score, indicating that the model misses 35% of actual correct recommendations.
3. **Accuracy (0.80):**
    - The model is correct 80% of the time overall.
    - This is a mediocre accuracy score, but it may be influenced by class imbalance (more non-recommendations than recommendations).
4. **F1-score (0.71):**
    - The F1-score balances precision and recall. A score of 0.71 is bad but could be improved, especially by increasing recall.
5. **Confusion Matrix Analysis:**
    - **True Positives (TP):** 260 (correctly predicted recommendations).
    - **False Positives (FP):** 110 (incorrectly predicted recommendations).
    - **False Negatives (FN):** 140 (missed recommendations).
    - **True Negatives (TN):** 690 (correctly predicted non-recommendations).
    - The model has a relatively high number of false positives (110) and false negatives (140). Reducing false negatives is important to improve the model's ability to recommend relevant products, the worst metrics of the 3.

Exercise 3 Search

Solution:

```
class Node:
    def __init__(self, name):
        self.name = name
        self.gCost = float('inf')
        self.hCost = float('inf')
        self.fCost = float('inf')
        self.parent = None

class Graph:
    def __init__(self):
        self.nodes = {}
        self.edges = {}
        self.heuristics = {}

    def addEdge(self, source, target, cost):
        if source not in self.nodes:
            self.nodes[source] = Node(source)
        if target not in self.nodes:
            self.nodes[target] = Node(target)

        if source not in self.edges:
            self.edges[source] = {}
        self.edges[source][target] = cost

    def addHeuristic(self, node, value):
        self.heuristics[node] = value

def aStarSearch(graph, start, goal):
    openSet = {start}
    closedSet = set()

    graph.nodes[start].gCost = 0
    graph.nodes[start].hCost = graph.heuristics[start]
    graph.nodes[start].fCost = graph.heuristics[start]
    while openSet:
        current = min(openSet, key=lambda x: graph.nodes[x].fCost)

        if current == goal:
            path = []
            cost = graph.nodes[current].gCost
            while current:
                path.append(current)
                current = graph.nodes[current].parent
            return path[::-1], cost

        openSet.remove(current)
        closedSet.add(current)

        if current in graph.edges:
            for neighbor, edgeCost in graph.edges[current].items():
                if neighbor in closedSet:
                    continue

                tentativeGCost = graph.nodes[current].gCost + edgeCost

                if neighbor not in openSet:
                    openSet.add(neighbor)
                elif tentativeGCost >= graph.nodes[neighbor].gCost:
                    continue

                graph.nodes[neighbor].parent = current
                graph.nodes[neighbor].gCost = tentativeGCost
                graph.nodes[neighbor].hCost = graph.heuristics[neighbor]
                graph.nodes[neighbor].fCost = tentativeGCost + graph.heuristics[neigh

    return None, float('inf')

networkGraph = Graph()

edges = [
    ('A', 'B', 10),
    ('A', 'C', 15),
    ('B', 'D', 12),
    ('C', 'D', 10),
    ('D', 'E', 5)
]

heuristics = [
    ('A', 20),
    ('B', 15),
    ('C', 10),
    ('D', 5),
    ('E', 0)
]

for source, target, cost in edges:
    networkGraph.addEdge(source, target, cost)

for node, value in heuristics:
    networkGraph.addHeuristic(node, value)

path, cost = aStarSearch(networkGraph, 'A', 'E')
print(f"Optimal Path: {' → '.join(path)}")
print(f"Total Cost: {cost} ms (milliseconds)")
```

DOCUMENTATION: https://github.com/HugeErick/ia/blob/main/exam1/astar-documentation.md