Predictive Analysis & Optimization Techniques - Practical: Code Answers

Author: Generated by ChatGPT
Date: 2025-09-23

Q1. Data collection from online, local drive and .csv file

Code (Python):

```
# Data collection examples using pandas
import pandas as pd

# 1. Read from local CSV
df_local = pd.read_csv('data/local_file.csv')  # path on local drive

# 2. Read from a URL (online CSV)
url = 'https://example.com/data.csv'
df_online = pd.read_csv(url)

# 3. Read Excel from local drive
df_excel = pd.read_excel('data/local_file.xlsx', sheet_name=0)

# 4. Read JSON from file or URL
df_json_local = pd.read_json('data/local.json')
df_json_online = pd.read_json('https://example.com/data.json')

# 5. Basic preview
print(df_local.head())
print(df_online.shape)
```

Q2. North-West Corner method (Transportation problem – IBFS)

Code (Python):

```
def north_west(supply, demand):
    # supply: list of supplies for rows
    # demand: list of demands for cols
    m, n = len(supply), len(demand)
    i = j = 0
    alloc = [[0]*n for _ in range(m)]
    supply = supply[:]  # copy
    demand = demand[:]
    while i < m and j < n:
        q = min(supply[i], demand[j])
        alloc[i][j] = q
        supply[i] -= q
        demand[j] -= q
        if supply[i] == 0 and i < m-1:
            i += 1
        elif demand[j] == 0 and j < n-1:
            j += 1
        else:
            # if both exhausted, advance one (prefer row)
            if supply[i] == 0 and demand[j] == 0:
                if i < m-1:
                    i += 1
                else:
                    j += 1
    return alloc

# Example (from doc):
supply = [17, 12, 16]   # O1, O2, O3
demand = [14, 8, 23]    # D1, D2, D3
alloc = north_west(supply, demand)
for row in alloc:
    print(row)
```

Q2. Least Cost Method (IBFS)

Code (Python):

```
import copy

def least_cost(costs, supply, demand):
    # costs: 2D list, supply: list, demand: list
    costs_copy = copy.deepcopy(costs)
    m, n = len(costs), len(costs[0])
    alloc = [[0]*n for _ in range(m)]
    supply = supply[:]
    demand = demand[:]
    while True:
        # find min cost cell among remaining positive supply/demand
        min_cost = None
        min_cell = None
        for i in range(m):
            for j in range(n):
                if supply[i] > 0 and demand[j] > 0:
                    if min_cost is None or costs_copy[i][j] < min_cost:
                        min_cost = costs_copy[i][j]
                        min_cell = (i, j)
        if min_cell is None:
            break
        i, j = min_cell
        q = min(supply[i], demand[j])
        alloc[i][j] = q
        supply[i] -= q
        demand[j] -= q
        # mark cell as used (optionally set cost to large)
        costs_copy[i][j] = float('inf')
    return alloc

# Example usage from doc (O1,O2,O3 supplies 20,30,25 respectively):
costs = [
    [8,6,10],
    [9,12,13],
    [14,9,16],
]
supply = [20,30,25]
demand = [30,25,20]
print(least_cost(costs, supply, demand))
```

Q2. Vogel's Approximation Method (VAM)

Code (Python):

```
import copy
from collections import defaultdict

def vogel_approx(costs, supply, demand):
    costs = copy.deepcopy(costs)
    supply = supply[:]
    demand = demand[:]
    m, n = len(costs), len(costs[0])
    alloc = [[0]*n for _ in range(m)]
    rows = set(range(m))
    cols = set(range(n))
    while any(supply) and any(demand):
        # compute penalties
        row_pen = {}
        col_pen = {}
        for i in rows:
            available = [costs[i][j] for j in range(n) if j in cols]
            if len(available) >= 2:
                s = sorted(available)
                row_pen[i] = s[1] - s[0]
            elif available:
                row_pen[i] = available[0]
        for j in cols:
            available = [costs[i][j] for i in range(m) if i in rows]
            if len(available) >= 2:
                s = sorted(available)
                col_pen[j] = s[1] - s[0]
            elif available:
                col_pen[j] = available[0]
        # choose max penalty
        all_pen = []
        if row_pen: all_pen.append(('r', max(row_pen, key=row_pen.get), row_pen[max(row_pen, key=row_p
        if col_pen: all_pen.append(('c', max(col_pen, key=col_pen.get), col_pen[max(col_pen, key=col_p
        if not all_pen:
            break
        typ, idx, _ = max(all_pen, key=lambda x: x[2])
        # find min cost in that row/col
        if typ == 'r':
            i = idx
            j = min((j for j in cols), key=lambda x: costs[i][x])
        else:
            j = idx
            i = min((i for i in rows), key=lambda x: costs[x][j])
        q = min(supply[i], demand[j])
        alloc[i][j] = q
        supply[i] -= q
        demand[j] -= q
        if supply[i] == 0:
            rows.remove(i)
        if demand[j] == 0:
            cols.remove(j)
    return alloc

# Example (from doc: 3 supplies S1,S2,S3 and 4 demands D1..D4)
costs = [
    [21,16,15,13],
    [17,18,14,23],
    [32,27,18,41],
]
supply = [11,13,19]
demand = [6,10,12,15]
print(vogel_approx(costs, supply, demand))
```

Q1. Perform Regression over the dataset

Code (Python):

```
# Example: Linear regression using scikit-learn
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# load dataset (replace with real path)
df = pd.read_csv('data/regression_dataset.csv')
# Assume last column is target
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
pred = model.predict(X_test)
```

```
print('MSE:', mean_squared_error(y_test, pred))
print('R2:', r2_score(y_test, pred))
```

Q1. Perform Classification of dataset

Code (Python):

```
# Example: RandomForest classification pipeline
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

df = pd.read_csv('data/classification_dataset.csv')
X = df.drop('target', axis=1)
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)
clf = RandomForestClassifier(n_estimators=100, random_state=7)
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
print('Accuracy:', accuracy_score(y_test, pred))
print(classification_report(y_test, pred))
```

Q1. Decision Tree operation over the dataset

Code (Python):

```
# Example: Decision Tree classifier and visualization
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split

df = pd.read_csv('data/decision_tree_dataset.csv')
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=0)
dt = DecisionTreeClassifier(max_depth=4, random_state=0)
dt.fit(X_train, y_train)
print(export_text(dt, feature_names=list(X.columns)))
```

Q2. Solve linear programming in pulp

Code (Python):

```
# Example using PuLP to solve:
# Max Z = x1 + 2x2
# s.t. x1 + 2x2 <= 20
#      x1 +  x2 <= 12
#      x1,x2 >= 0

from pulp import LpMaximize, LpProblem, LpVariable, value

prob = LpProblem('LP_example', LpMaximize)
x1 = LpVariable('x1', lowBound=0)
x2 = LpVariable('x2', lowBound=0)
prob += x1 + 2*x2
prob += x1 + 2*x2 <= 20
prob += x1 + x2 <= 12
prob.solve()
print('Status:', prob.status)
print('x1 =', value(x1))
print('x2 =', value(x2))
print('Objective =', value(prob.objective))
```

Q2. Algebraic Method for solving 2x2 game (Player A vs Player B)

Code (Python):

```python
# Payoff matrix (rows = strategies of A, cols = strategies of B)
# Example matrix from doc:
#      B1  B2
# A1   3   2
# A2   1   4
import numpy as np

A = np.array([[3,2],[1,4]], dtype=float)

def solve_2x2_game(A):
    # Solve for mixed strategy p for A (p on row1, 1-p on row2)
    a,b = A[0,0], A[0,1]
    c,d = A[1,0], A[1,1]
    denom = a - b - c + d
    if denom == 0:
        raise ValueError('Degenerate case')
    p = (d - c) / denom
    q = (d - b) / denom  # B plays column1 with probability q
    # value of the game
    V = (a*d - b*c) / denom
    return p, q, V

p, q, V = solve_2x2_game(A)
print('A plays row1 with prob p =', p)
print('B plays col1 with prob q =', q)
print('Value V =', V)
```

Q2. Two-person zero-sum game without saddle point (solve via linear programming)

Code (Python):

```python
# Use linear programming to find optimal mixed strategy for player A
# If payoff matrix has negative entries, add a constant to make all positive.

import numpy as np
from scipy.optimize import linprog

def solve_zero_sum_A(payoff):
    # maximize v subject to A*p >= v (elementwise), sum p = 1, p >= 0
    # linprog solves minimization; convert to:
    # minimize -v s.t. A*p - v >= 0 -> -A*p + v <= 0
    # We'll create variables [p..., v]
    m, n = payoff.shape
    # We minimize -v
    c = np.zeros(n+1)
    c[-1] = -1.0
    # Constraints: -A*p + v <= 0  (m constraints)
    A_ub = np.hstack([-payoff, np.ones((m,1))])
    b_ub = np.zeros(m)
    # equality sum p = 1
    A_eq = np.zeros((1,n+1))
    A_eq[0,:n] = 1
    A_eq[0,-1] = 0
    b_eq = np.array([1])
    bounds = [(0,1)]*n + [(None,None)]
    res = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method='highs')
    if res.success:
        p = res.x[:n]
        v = res.x[-1]
        return p, v
    else:
        return None

# Note: scipy required. If not available, use commercial solvers / PuLP alternative.
```

Q2. Example: Game without saddle point — pay-off matrix code snippet

Code (Python):

```python
# Example payoff matrix (A's payoffs)
payoff = np.array([[3,5,2],[4,1,6]])
# check for saddle point
row_mins = payoff.min(axis=1)
col_max_of_row_mins = row_mins.max()
col_maxs = payoff.max(axis=0)
row_min_of_col_maxs = col_maxs.min()
if col_max_of_row_mins == row_min_of_col_maxs:
    print('Saddle point exists with value', col_max_of_row_mins)
else:
    print('No saddle point; solve for mixed strategy (e.g., using LP)')
```

Q1. Data cleaning operations

Code (Python):

```python
# Data cleaning examples with pandas
import pandas as pd

df = pd.read_csv('data/noisy_data.csv')
# 1. Drop duplicates
df = df.drop_duplicates()
# 2. Handle missing values
df['col1'] = df['col1'].fillna(df['col1'].median())
df = df.dropna(subset=['important_column'])  # drop rows missing an important column
# 3. Fix data types
df['date'] = pd.to_datetime(df['date'], errors='coerce')
df['category'] = df['category'].astype('category')
# 4. Outlier handling (cap at 1st-99th percentile)
low, high = df['value'].quantile([0.01, 0.99])
df['value'] = df['value'].clip(lower=low, upper=high)
```

Q1. 2D and 3D visualization examples

Code (Python):

```python
# 2D: matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

x = np.linspace(0,10,100)
y = np.sin(x)
plt.figure()
plt.plot(x,y)
plt.title('2D line plot')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.show()

# 3D: matplotlib mplot3d
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X = np.linspace(-5,5,30)
Y = np.linspace(-5,5,30)
X,Y = np.meshgrid(X,Y)
Z = np.sin(np.sqrt(X**2 + Y**2))
ax.plot_surface(X,Y,Z, rstride=1, cstride=1)
ax.set_title('3D surface')
plt.show()
```

```
Q1. Small stock market analysis for bull or bear (NSE/BSE)

Code (Python):
```

```python
# Using yfinance (example). Replace symbol with NSE/BSE ticker like 'RELIANCE.NS' or 'TCS.NS'.
import yfinance as yf
import pandas as pd

symbol = 'RELIANCE.NS'
df = yf.download(symbol, period='6mo', interval='1d')
# Simple bull/bear rule using moving averages
df['MA20'] = df['Close'].rolling(20).mean()
df['MA50'] = df['Close'].rolling(50).mean()
latest = df.iloc[-1]
if latest['MA20'] > latest['MA50']:
    print('Short-term bullish signal (MA20 > MA50)')
else:
    print('Short-term bearish or neutral')
```

Notes

- The code snippets above are ready to run (replace file paths / data with your actual datasets).
- Some snippets require external packages (scikit-learn, pandas, yfinance, pulp, scipy). Install via pip if not present:
pip install pandas scikit-learn yfinance pulp scipy reportlab

End of file.