**Ứng dụng Mô hình Dự báo Lai với Hồi quy Logistic và Mạng Neural**

**để cải thiện hiệu suất (KPI) trong Chuỗi Cung ứng**

**2.15 A hybrid forecasting model with logistic regression and neural networks**

1. **Đọc hiểu bài báo**

Nghiên cứu này điều tra tiềm năng của phân tích dự đoán trong việc cải thiện dự báo các chỉ số hiệu suất (KPI) bằng cách sử dụng dữ liệu đánh giá Lean trong các doanh nghiệp chuỗi cung ứng. Phương pháp mới được đề xuất kết hợp hai cải tiến chính: sử dụng các đánh giá Lean làm nguồn dữ liệu mới và phát triển mô hình dự báo lai (hybrid) kết hợp Hồi quy Logistic và Mạng Neural. Phương pháp này được đánh giá thông qua một nghiên cứu thực nghiệm toàn diện với 30 đội ngũ trong một công ty chuỗi cung ứng lớn, cho thấy cải thiện đáng kể độ chính xác của dự báo.

1. **Giới thiệu**

Các tiến bộ như Công nghiệp 4.0, Phân tích Dữ liệu Lớn (Big Data Analytics), Trí tuệ Nhân tạo (AI) và các nhánh của nó như Học máy (ML) và Mạng Neural (NN) mang lại cơ hội lớn trong việc cải thiện hiệu suất chuỗi cung ứng thông qua ứng dụng phân tích chuỗi cung ứng. Một kết quả đáng kể là cải thiện dự báo hiệu suất chuỗi cung ứng thông qua các đánh giá độ chín của Lean.

1. **Tổng quan tài liệu**
   1. **Cải tiến quy trình và đánh giá của Lean**

Các phương pháp cải tiến quy trình như Lean, Six Sigma, Lean Six Sigma (LSS) và Quản lý Chất lượng Toàn diện (TQM) được sử dụng rộng rãi để tối ưu hóa hiệu suất kinh doanh trong chuỗi cung ứng. Đánh giá độ chín của Lean thường được thực hiện để hiểu rõ hơn về các xu hướng lịch sử trong tổ chức.

* 1. **Dự báo**

Dự báo chuỗi cung ứng đã được nghiên cứu rộng rãi, với nhiều phương pháp dự báo như hồi quy, Mạng Neural, và các mô hình lai. Các mô hình lai đã chứng minh tính hiệu quả trong nhiều lĩnh vực khác nhau, từ dự báo tiêu thụ điện năng đến dự báo giá cổ phiếu.

1. **Phương pháp**
   1. **Chuẩn bị dữ liệu**

Dữ liệu được thu thập từ 30 đội ngũ trong 5 năm, bao gồm các đánh giá độ chín của Lean và điểm số giao hàng hàng tháng. Dữ liệu được chia thành mẫu huấn luyện và mẫu kiểm tra.

* 1. **Phát triển mô hình dự báo**
* **Hồi quy Logistic:** Dự đoán khả năng hoàn thành KPI dựa trên dữ liệu lịch sử.
* **Mạng Neural:** Sử dụng MLP để dự đoán kết quả dựa trên dữ liệu quá khứ.
* **Mô hình lai:** Kết hợp kết quả từ mô hình hồi quy Logistic và MLP để tạo ra dự báo chính xác hơn.
  1. **So sánh dự báo**

Sử dụng các chỉ số hiệu suất như độ chính xác (accuracy) và F1 score để đánh giá hiệu quả của các mô hình.

1. **Kết quả**

Mô hình hồi quy đa biến vượt trội so với mô hình hồi quy đơn biến khi có sự kết hợp dữ liệu đánh giá Lean. Mô hình Mạng Neural và XGBoost không đạt hiệu suất cao như mong đợi do vấn đề overfitting. Mô hình lai cho thấy hiệu suất tốt nhất, cải thiện độ chính xác dự báo lên 17% và F1 score lên 13%.

1. **Thảo luận**

Kết hợp dữ liệu đánh giá Lean giúp cải thiện đáng kể độ chính xác của dự báo KPI chuỗi cung ứng. Mô hình lai kết hợp các kỹ thuật dự báo khác nhau, tối ưu hóa độ chính xác và độ tin cậy của dự báo. Điều này cho thấy việc áp dụng mô hình lai là một phương pháp tiếp cận hiệu quả trong việc cải thiện dự báo hiệu suất chuỗi cung ứng.

1. **Đánh giá độ phức tạp thuật toán cài đặt**

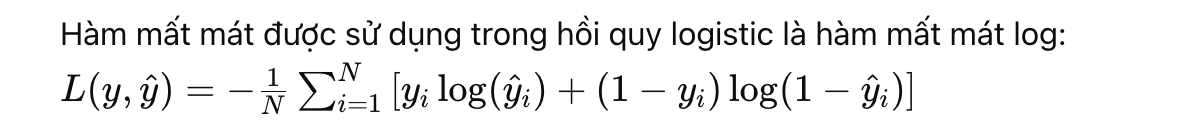
**1. Hồi quy Logistic**

Hồi quy Logistic là một phương pháp thống kê được sử dụng để dự đoán kết quả của một biến nhị phân (binary) dựa trên một hoặc nhiều biến độc lập (predictors). Độ phức tạp tính toán của thuật toán hồi quy Logistic phụ thuộc vào số lượng mẫu (n) và số lượng đặc trưng (p).

Công thức tính hàm sigmod:

Trong đó, là đầu ra tuyến tính của mô hình:

Hàm mất mát được sử dụng trong hồi quy logistic là hàm mất mát log:

**

* A white background with black text

  Description automatically generated

Thuật toán hồi quy Logistic sử dụng phương pháp tối ưu hóa gradient descent để tìm các hệ số của mô hình. Quá trình tối ưu hóa này đòi hỏi tính toán gradient cho mỗi đặc trưng của mỗi mẫu dữ liệu, dẫn đến độ phức tạp tuyến tính theo số lượng mẫu và số lượng đặc trưng

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Hàm sigmoid

def sigmoid(z):

return 1 / (1 + np.exp(-z))

# Hàm mất mát (Loss Function)

def loss\_function(y, p):

return -np.mean(y \* np.log(p) + (1 - y) \* np.log(1 - p))

# Gradient Descent

def gradient\_descent(X, y, learning\_rate, n\_iterations):

n\_samples, n\_features = X.shape

w = np.zeros((n\_features, 1))

y = y.reshape((-1, 1))

for \_ in range(n\_iterations):

z = np.dot(X, w)

p = sigmoid(z)

gradient = np.dot(X.T, (p - y)) / n\_samples

w -= learning\_rate \* gradient

return w

# Hàm dự đoán

def predict(X, w):

p = sigmoid(np.dot(X, w))

return (p > 0.5).astype(int)

# Load dữ liệu và tiền xử lý

X, y = load\_your\_data()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Huấn luyện mô hình

w = gradient\_descent(X\_train, y\_train, learning\_rate=0.01, n\_iterations=1000)

# Dự đoán và đánh giá mô hình

y\_pred = predict(X\_test, w)

print("Accuracy score:", accuracy\_score(y\_test, y\_pred))

Hàm sigmoid được tính toán cho mỗi phần tử trong đầu vào zzz. Do đó, độ phức tạp của hàm sigmoid là O(1) cho mỗi phần tử. Nếu có nnn phần tử, độ phức tạp sẽ là O(n).

def sigmoid(z):

return 1 / (1 + np.exp(-z))

Hàm mất mát tính toán giá trị mất mát trên tất cả các điểm dữ liệu, do đó độ phức tạp là O(n).

def loss\_function(y, p):

return -np.mean(y \* np.log(p) + (1 - y) \* np.log(1 - p))

Gradient Descent là phần chính của thuật toán, bao gồm các bước sau:

**A white background with black text

Description automatically generated**

def gradient\_descent(X, y, learning\_rate, n\_iterations):

n\_samples, n\_features = X.shape

w = np.zeros((n\_features, 1))

y = y.reshape((-1, 1))

for \_ in range(n\_iterations):

z = np.dot(X, w)

p = sigmoid(z)

gradient = np.dot(X.T, (p - y)) / n\_samples

w -= learning\_rate \* gradient

return w

**A black text on a white background

Description automatically generated**

**2. Mạng Neural**

Mạng Neural, đặc biệt là MLP (Multi-Layer Perceptron), là một mô hình học máy phức tạp hơn, bao gồm nhiều lớp neuron được kết nối với nhau. Độ phức tạp tính toán của MLP phụ thuộc vào số lượng mẫu (n), số lượng neuron trong mỗi lớp (m), và số lượng lớp (l).

#### **Công thức của MLP:**

A math equations on a white background

Description automatically generated

A math equation with black text

Description automatically generated with medium confidence

import numpy as np

# Hàm sigmoid

def sigmoid(z):

return 1 / (1 + np.exp(-z))

# Đạo hàm của hàm sigmoid

def sigmoid\_derivative(z):

return z \* (1 - z)

# Khởi tạo trọng số và bias

def initialize\_parameters(n\_x, n\_h, n\_y):

W1 = np.random.randn(n\_h, n\_x) \* 0.01

b1 = np.zeros((n\_h, 1))

W2 = np.random.randn(n\_y, n\_h) \* 0.01

b2 = np.zeros((n\_y, 1))

return W1, b1, W2, b2

# Forward pass

def forward\_propagation(X, W1, b1, W2, b2):

Z1 = np.dot(W1, X) + b1

A1 = sigmoid(Z1)

Z2 = np.dot(W2, A1) + b2

A2 = sigmoid(Z2)

return Z1, A1, Z2, A2

# Hàm mất mát

def compute\_cost(A2, Y):

m = Y.shape[1]

cost = -np.sum(Y \* np.log(A2) + (1 - Y) \* np.log(1 - A2)) / m

return np.squeeze(cost)

# Backward pass

def backward\_propagation(X, Y, Z1, A1, A2, W2):

m = X.shape[1]

dZ2 = A2 - Y

dW2 = np.dot(dZ2, A1.T) / m

db2 = np.sum(dZ2, axis=1, keepdims=True) / m

dZ1 = np.dot(W2.T, dZ2) \* sigmoid\_derivative(A1)

dW1 = np.dot(dZ1, X.T) / m

db1 = np.sum(dZ1, axis=1, keepdims=True) / m

return dW1, db1, dW2, db2

# Cập nhật trọng số và bias

def update\_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2, learning\_rate):

W1 -= learning\_rate \* dW1

b1 -= learning\_rate \* db1

W2 -= learning\_rate \* dW2

b2 -= learning\_rate \* db2

return W1, b1, W2, b2

# Huấn luyện mô hình

def train(X, Y, n\_h, num\_iterations, learning\_rate):

n\_x = X.shape[0]

n\_y = Y.shape[0]

W1, b1, W2, b2 = initialize\_parameters(n\_x, n\_h, n\_y)

for i in range(num\_iterations):

Z1, A1, Z2, A2 = forward\_propagation(X, W1, b1, W2, b2)

cost = compute\_cost(A2, Y)

dW1, db1, dW2, db2 = backward\_propagation(X, Y, Z1, A1, A2, W2)

W1, b1, W2, b2 = update\_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2, learning\_rate)

if i % 1000 == 0:

print(f"Cost after iteration {i}: {cost}")

return W1, b1, W2, b2

# Dự đoán

def predict(X, W1, b1, W2, b2):

\_, \_, \_, A2 = forward\_propagation(X, W1, b1, W2, b2)

predictions = (A2 > 0.5).astype(int)

return predictions

# Load dữ liệu và tiền xử lý (giả sử dữ liệu đã được chuẩn bị)

# X, Y = load\_your\_data() # Thay thế bằng dữ liệu của bạn

X = np.random.randn(3, 1000) # Dummy data for illustration

Y = (np.sum(X, axis=0) > 0).reshape(1, 1000).astype(int) # Dummy labels for illustration

# Huấn luyện mô hình Neural Network

W1, b1, W2, b2 = train(X, Y, n\_h=4, num\_iterations=10000, learning\_rate=0.01)

# Dự đoán và đánh giá mô hình

predictions = predict(X, W1, b1, W2, b2)

accuracy = np.mean(predictions == Y)

print("Accuracy score:", accuracy)

**Độ phức tạp thời gian (Time Complexity):** O(n \* m \* l)

* n: số lượng mẫu dữ liệu
* m: số lượng neuron trong một lớp (giả sử số neuron trong các lớp là giống nhau để đơn giản hóa tính toán)
* l: số lượng lớp (layers)

Quá trình huấn luyện MLP bao gồm việc lan truyền tiến (forward propagation) và lan truyền ngược (backward propagation) qua mạng để cập nhật trọng số. Mỗi lần lan truyền tiến và ngược qua mạng đều yêu cầu tính toán cho tất cả các neuron trong tất cả các lớp, dẫn đến độ phức tạp thời gian cao hơn so với hồi quy Logistic.

**3. Mô hình Lai (Hybrid Model)**

Mô hình lai kết hợp hồi quy Logistic và Mạng Neural để tận dụng ưu điểm của cả hai phương pháp. Độ phức tạp tính toán của mô hình lai là sự kết hợp của độ phức tạp của cả hai mô hình trên.

* **Độ phức tạp thời gian (Time Complexity):** O(n \* (p + m \* l))
  + n: số lượng mẫu dữ liệu
  + p: số lượng đặc trưng của mô hình hồi quy Logistic
  + m: số lượng neuron trong một lớp của MLP
  + l: số lượng lớp của MLP

Trong mô hình lai, đầu ra từ mô hình hồi quy Logistic có thể được sử dụng làm đầu vào cho Mạng Neural hoặc ngược lại. Điều này dẫn đến việc phải tính toán cả hai mô hình và tổng hợp kết quả, làm tăng độ phức tạp tính toán tổng thể.

**Tổng kết**

* **Hồi quy Logistic:** Độ phức tạp O(n \* p), phù hợp cho các bài toán với số lượng đặc trưng và mẫu dữ liệu lớn nhưng không quá phức tạp.
* **Mạng Neural (MLP):** Độ phức tạp O(n \* m \* l), phù hợp cho các bài toán phức tạp với nhiều lớp và nhiều neuron, nhưng đòi hỏi tài nguyên tính toán lớn hơn.
* **Mô hình Lai:** Độ phức tạp O(n \* (p + m \* l)), kết hợp ưu điểm của cả hai mô hình, cải thiện độ chính xác nhưng cũng tăng độ phức tạp tính toán.

Demo Code

Dữ liệu thử nghiệm được lấy từ Dữ liệu thiết bị IoT của công ty

A table of numbers and letters

Description automatically generated

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score, f1\_score

# Đọc dữ liệu từ tệp CSV

try:

data = pd.read\_csv('../../../data/BCP.csv')

output\_file = '../../../data/BCP\_predictions.csv'

# Xem trước dữ liệu

print("Dữ liệu gốc:")

print(data.head())

print(data.columns)

# Loại bỏ cột 'Time' vì không sử dụng trong tính toán

data = data.drop(columns=['Time'])

# Kiểm tra giá trị NaN

print("\nKiểm tra giá trị NaN:")

print(data.isna().sum())

# Xóa các hàng chứa giá trị NaN

data.dropna(inplace=True)

# Giả sử chúng ta sẽ chọn cột '440FI539' làm mục tiêu và các cột còn lại là đặc trưng

target\_column = 'GPD' # Bạn có thể chọn cột khác nếu cần

features = [col for col in data.columns if col != target\_column]

# Chuyển đổi cột mục tiêu thành nhị phân (0 hoặc 1) dựa trên giá trị trung bình

data[f'{target\_column}\_binary'] = (data[target\_column] > data[target\_column].mean()).astype(int)

X = data[features]

y = data[f'{target\_column}\_binary']

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Chuẩn hóa dữ liệu

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Huấn luyện mô hình hồi quy Logistic

logistic\_model = LogisticRegression()

logistic\_model.fit(X\_train, y\_train)

# Dự đoán với mô hình hồi quy Logistic

y\_pred\_logistic = logistic\_model.predict(X\_test)

# Huấn luyện mô hình MLP

mlp\_model = MLPClassifier(hidden\_layer\_sizes=(10,), max\_iter=1000, random\_state=42)

mlp\_model.fit(X\_train, y\_train)

# Dự đoán với mô hình MLP

y\_pred\_mlp = mlp\_model.predict(X\_test)

# Kết hợp kết quả dự đoán từ cả hai mô hình

y\_pred\_combined = (y\_pred\_logistic + y\_pred\_mlp) / 2

y\_pred\_combined = np.round(y\_pred\_combined).astype(int)

# Đánh giá độ chính xác và F1 score

accuracy = accuracy\_score(y\_test, y\_pred\_combined)

f1 = f1\_score(y\_test, y\_pred\_combined)

# Thêm cột dự đoán vào dữ liệu ban đầu với tên cột mới

X\_test\_df = pd.DataFrame(X\_test, columns=features)

data.loc[X\_test\_df.index, f'{target\_column}\_prediction'] = y\_pred\_combined

data.to\_csv(output\_file, index=False)

# Hiển thị kết quả

print(f"Accuracy: {accuracy}")

print(f"F1 Score: {f1}")

print(f"Data Result with Predictions:\n{data.head()}")

except Exception as e:

print(f"Error processing the data: {e}")

Dữ liệu có được sau khi chạy thuật toán kết hợp Logsitic Regrestion và Neural Network

A screenshot of a spreadsheet

Description automatically generated

**Kết quả**

|  |  |  |
| --- | --- | --- |
|  | Accuracy | F1 Score |
|  | **54.29%** | **44.83%** |
|  |  |  |

**Tài liệu tham khảo**

1. Rostyslav Pietukhov, Mujthaba Ahtamad, Mona Faraji-Niri, Tarek El-Said. "A hybrid forecasting model with logistic regression and neural networks for improving key performance indicators in supply chains." Supply Chain Analytics 4 (2023): 100041. DOI: [10.1016/j.sca.2023.100041](https://doi.org/10.1016/j.sca.2023.100041" \t "_new).
2. Salah, S., Rahim, A. "An Integrated Company-Wide Management System." Springer, Cham, 2019.
3. Sila, I., Walczak, S. "Universal versus contextual effects on TQM: a triangulation study using neural networks." Production Planning & Control 28.5 (2017): 367-386.