The **L-BFGS line search failed (code 3)** message in cuML's LogisticRegression means that the optimizer couldn't find a step size to decrease the loss function further. This often happens due to:

1. **Class imbalance** (which we already addressed with class weights)
2. **Features on different scales** (which should be handled since you used StandardScaler)
3. **Ill-conditioned data** (correlated features or redundant information)
4. **Too many iterations or too strict convergence criteria**
5. **Numerical instability from GPU floating point precision** in cuML's implementation

**Fixing the L-BFGS Issue**

Try these solutions in order:

**1. Reduce C (Regularization Parameter)**

A smaller C increases regularization, preventing instability.

log\_reg = LogisticRegression(penalty='l2', C=0.1, class\_weight={0: 1.0, 1: 1.5, 2: 3.0})

log\_reg.fit(X\_train, y\_train)

**2. Use lbfgs with More Iterations**

Increase max\_iter to give L-BFGS more time to converge.

log\_reg = LogisticRegression(penalty='l2', C=1.0, class\_weight={0: 1.0, 1: 1.5, 2: 3.0}, max\_iter=500)

log\_reg.fit(X\_train, y\_train)

**3. Try a Different Solver**

If lbfgs still fails, switch to saga (faster and handles large datasets better).

log\_reg = LogisticRegression(penalty='l2', C=1.0, class\_weight={0: 1.0, 1.5, 2: 3.0}, solver='saga')

log\_reg.fit(X\_train, y\_train)

**4. Check for Feature Correlation**

If high correlation exists, drop redundant features.

corr\_matrix = X\_train.to\_pandas().corr()

print(corr\_matrix.abs().unstack().sort\_values(ascending=False).head(10)) # Check top correlations

**5. Reduce Features (PCA or Feature Selection)**

Use PCA or feature selection to remove noise.

from cuml.decomposition import PCA

pca = PCA(n\_components=30) # Reduce to 30 dimensions

X\_train\_pca = pca.fit\_transform(X\_train)

X\_test\_pca = pca.transform(X\_test)

**What Dr. Slater Would Expect**

1. **First, reduce C** and rerun logistic regression.
2. **If it still fails, increase max\_iter** and use saga instead of lbfgs.
3. **If performance doesn’t improve, check feature correlations** and apply PCA.

**Next Steps:** Try reducing C first and see if L-BFGS converges. If not, move to the next solution.

Cujt from nb:

# import numpy as np

# # Drop columns with excessive missing values

# columns\_to\_drop = ['weight', 'payer\_code', 'medical\_specialty']

# diabetic\_data.drop(columns=columns\_to\_drop, inplace=True)

# # Fill missing categorical values

# diabetic\_data.fillna({

#     'race': 'Unknown',

#     'diag\_1': 'Missing',

#     'diag\_2': 'Missing',

#     'diag\_3': 'Missing',

#     'max\_glu\_serum': 'None',

#     'A1Cresult': 'None'

# }, inplace=True)

# # Verify missing values are handled

# print("\n✅ Missing values after handling:")

# print(diabetic\_data.isnull().sum()[diabetic\_data.isnull().sum() > 0])

# # Ensure 'admission\_type\_id' is the same data type in both datasets

# diabetic\_data['admission\_type\_id'] = diabetic\_data['admission\_type\_id'].astype(str)

# ids\_mapping['admission\_type\_id'] = ids\_mapping['admission\_type\_id'].astype(str)

# # Merge IDs mapping with diabetic\_data (if applicable)

# diabetic\_data = diabetic\_data.merge(ids\_mapping, on='admission\_type\_id', how='left')

# # Verify the merge

# print("\n✅ First few rows after merging:")

# print(diabetic\_data.head())

# from sklearn.preprocessing import LabelEncoder

# # Ensure 'readmitted' is mapped correctly

# readmitted\_mapping = {'NO': 0, '>30': 1, '<30': 2}

# diabetic\_data['readmitted'] = diabetic\_data['readmitted'].map(readmitted\_mapping)

# # Identify categorical columns

# categorical\_cols = ['race', 'gender', 'age', 'max\_glu\_serum', 'A1Cresult', 'change', 'diabetesMed']

# medication\_cols = ['metformin', 'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride',

#                    'acetohexamide', 'glipizide', 'glyburide', 'tolbutamide', 'pioglitazone',

#                    'rosiglitazone', 'acarbose', 'miglitol', 'troglitazone', 'tolazamide',

#                    'examide', 'citoglipton', 'insulin', 'glyburide-metformin',

#                    'glipizide-metformin', 'glimepiride-pioglitazone', 'metformin-rosiglitazone',

#                    'metformin-pioglitazone']

# # Include diagnosis codes as categorical

# all\_categorical = categorical\_cols + medication\_cols + ['diag\_1', 'diag\_2', 'diag\_3']

# # Apply Label Encoding

# label\_encoders = {}

# for col in all\_categorical:

#     if diabetic\_data[col].dtype == "object":  # Only encode object columns

#         le = LabelEncoder()

#         diabetic\_data[col] = le.fit\_transform(diabetic\_data[col].astype(str))

#         label\_encoders[col] = le  # Save encoders for possible inverse transformation

# # Verify all columns are now numerical

# print("\n✅ Updated column data types (should all be numeric now):")

# print(diabetic\_data.dtypes)

# # Convert 'admission\_type\_id' to numeric (shouldn't be object)

# diabetic\_data['admission\_type\_id'] = diabetic\_data['admission\_type\_id'].astype(int)

# # Drop 'description' column (unuseful for modeling)

# diabetic\_data.drop(columns=['description'], inplace=True)

# # Confirm all columns are now numeric

# print("\n Updated column data types (ALL should be numeric now):")

# print(diabetic\_data.dtypes)

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

# from sklearn.preprocessing import StandardScaler

# # Drop ID columns (not useful for predictions)

# X = diabetic\_data.drop(columns=['readmitted', 'encounter\_id', 'patient\_nbr'])

# y = diabetic\_data['readmitted']

# # Split into train (80%) and test (20%) sets

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

# # Standardize numerical features

# scaler = StandardScaler()

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

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

# # Confirm split sizes

# print("\n Training set size:", X\_train.shape)

# print(" Test set size:", X\_test.shape)

# from sklearn.linear\_model import LogisticRegression

# from sklearn.metrics import accuracy\_score, classification\_report

# # Initialize and train logistic regression model

# model = LogisticRegression(max\_iter=500, random\_state=42)

# model.fit(X\_train\_scaled, y\_train)

# # Make predictions

# y\_pred = model.predict(X\_test\_scaled)

# # Evaluate the model

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

# print(f"\n✅ Model Accuracy: {accuracy:.4f}")

# # Display classification report

# print("\n📊 Classification Report:")

# print(classification\_report(y\_test, y\_pred))

# # 8 Improve the performance

# # 8A HANDLE CLASS IMBALANCE

# from imblearn.over\_sampling import SMOTE

# from collections import Counter

# # Print class distribution before balancing

# print("\n🔍 Class distribution before SMOTE:", Counter(y\_train))

# # Apply SMOTE (Synthetic Minority Over-sampling Technique)

# smote = SMOTE(sampling\_strategy='auto', random\_state=42)

# X\_train\_resampled, y\_train\_resampled = smote.fit\_resample(X\_train\_scaled, y\_train)

# # Print class distribution after balancing

# print("\n✅ Class distribution after SMOTE:", Counter(y\_train\_resampled))

# from imblearn.over\_sampling import SMOTE

# # Apply SMOTE to balance the training dataset

# smote = SMOTE(random\_state=42)

# X\_train\_smote, y\_train\_smote = smote.fit\_resample(X\_train, y\_train)

# # Check the new class distribution

# from collections import Counter

# print("\n🔍 Class distribution after SMOTE:", Counter(y\_train\_smote))

# # 8b retrain model

# from sklearn.ensemble import RandomForestClassifier

# from sklearn.metrics import accuracy\_score, classification\_report

# # Train the model on the balanced dataset

# rf\_balanced = RandomForestClassifier(random\_state=42, n\_jobs=-1)

# rf\_balanced.fit(X\_train\_smote, y\_train\_smote)

# # Predictions on test set

# y\_pred\_balanced = rf\_balanced.predict(X\_test)

# # Evaluate performance

# accuracy\_balanced = accuracy\_score(y\_test, y\_pred\_balanced)

# print(f"\n✅ Model Accuracy After SMOTE: {accuracy\_balanced:.4f}")

# # Print classification report

# print("\n📊 Classification Report After SMOTE:")

# print(classification\_report(y\_test, y\_pred\_balanced))

# from sklearn.ensemble import RandomForestClassifier

# from sklearn.metrics import accuracy\_score, classification\_report

# # Retrain the model on the balanced dataset

# rf\_balanced = RandomForestClassifier(random\_state=42, n\_jobs=-1)

# rf\_balanced.fit(X\_train\_smote, y\_train\_smote)

# # Predictions on test set

# y\_pred\_balanced = rf\_balanced.predict(X\_test)

# # Evaluate performance

# accuracy\_balanced = accuracy\_score(y\_test, y\_pred\_balanced)

# print(f"\n✅ Model Accuracy After SMOTE: {accuracy\_balanced:.4f}")

# # Print classification report

# print("\n📊 Classification Report After SMOTE:")

# print(classification\_report(y\_test, y\_pred\_balanced))

# # Validate If the Model Is Overfitting

# feature\_importances = pd.DataFrame({'Feature': X\_train.columns,

#                                     'Importance': rf\_balanced.feature\_importances\_})

# print(feature\_importances.sort\_values(by='Importance', ascending=False).head(10))

# # Cross validate instread of fixed train to prevent model from memorizing dataset

# from sklearn.model\_selection import cross\_val\_score

# scores = cross\_val\_score(rf\_balanced, X\_train\_smote, y\_train\_smote, cv=5, scoring='accuracy')

# print("Cross-validation accuracy:", scores.mean())

# #  Check Confusion Matrix for False Positives/Negatives

# from sklearn.metrics import confusion\_matrix

# import seaborn as sns

# import matplotlib.pyplot as plt

# cm = confusion\_matrix(y\_test, y\_pred\_balanced)

# sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

# plt.xlabel('Predicted')

# plt.ylabel('Actual')

# plt.title('Confusion Matrix')

# plt.show()