Brain Stroke Streaming Model

Laura Contreras

University of Denver

COMP 4334

03/11/2023

I. Introduction

Strokes are the leading cause of long-term disabilities and death in the United States. Almost every 40 seconds a person in the US will experience a stroke and every 3.5 minutes someone will die from a stroke. In men, a stroke is the leading cause of death, while for women it is the 5th leading cause (*Stroke Facts | cdc.gov*, 2022). The most common type of stroke is an ischemic stroke, which occurs when blood is blocked from supplying to part of the brain or a brain vessel bursts. Another name for this is called a brain attack. During this healthy brain cells will start dying, resulting in a stroke (*Stroke | cdc.gov*, 2023). Symptoms can be a loss of balance, vision, droopiness in the face, and muscle weakness.

A worrisome factor of a stroke is that it increases with age and roughly 1 in 6 stroke deaths occur due to heart disease. With 1 in 3 Americans having health conditions for high blood pressure, high cholesterol, or smoking, all factors that lead to heart disease, the susceptibility to a potential stroke is high (*Stroke Facts | cdc.gov*, 2022).

The purpose of the project is to build a model that properly predicts whether a patient will be susceptible to a stroke. Because of how prevalent the disease is in the US, and globally, if a model can accurately predict the diagnosis, then doctors will be able to intervene earlier than later. By comparing different logistic regression models on their accuracy and F-1 score, an accurate model is selected.

The dataset utilized in the project comes from Kaggle (*Brain Stroke Prediction Dataset*, 2022). The data was created with over 11 features and in total there are 4981 respondents. The target variable used is the binary ‘Stroke’ attribute, where 0 is no stroke, and 1 is a yes for a stroke. There are 10 input variables that are categorical or continuous. An example of the continuous input variables is ‘avg_glucose_level’ (average glucose level in blood), and ‘BMI’ (body mass index). Some categorical variables in the dataset are ‘hypertension’ (patient has hypertension 0 or 1) and ‘heart_disease’ (patient has heart disease 0 or 1). Utilizing these attributes, a logistic model was created and hyper-tuned to various different parameters

using cross-validation to determine the best model for predicting whether a patient is susceptible to a stroke.

II. Data Preprocessing

A. Data Preparation

Before starting the model building, the dataset was read into a data frame and cleaned. A schema was created to ensure that each column was of the right data type. The age column had to be cast as a different type separately due to how the CSV was originally created and the stroke column was renamed to ‘label’ to ensure the model would work properly. The stroke data frame was then filtered to remove the ‘unknown’ smoking answer from the column ‘smoking_status’ to make sure all columns had a response. This resulted in a final data frame with 3481 respondents (Fig. 1).

B. Data Splitting

The data frame was then split on a 70/30 train test split, with ‘label’ being the target variable. From the split, a test and train stroke data set was obtained. The resulting train data frame had 2465 respondents, and the test set had 1016 respondents (Fig. 1).

```
strokeSchema = StructType([StructField('gender', StringType(), True), StructField('age', StringType(), True),
                           StructField('hypertension', LongType(), True), StructField('heart_disease', LongType(), True),
                           StructField('ever_married', StringType(), True), StructField('work_type', StringType(), True),
                          StructField('Residence_type', StringType(), True), StructField('avg_glucose_level', FloatType(), True),
                           StructField('bmi', FloatType(), True), StructField('smoking_status', StringType(), True),
                           StructField('stroke', LongType(), True)])

stroke = spark.read.format("csv").option("header", True).option("ignoreLeadingWhiteSpace", True).schema(strokeSchema).load("dbfs:///FileStore/tables/full_data_stroke.csv")
stroke = stroke.withColumnRenamed("stroke","label").withColumn("age", stroke["age"].cast(LongType()))
# We know that smoking_status contains "unknown" values. We will drop the unknown values then split the data set into training and test set
random.seed(58)
stroke = stroke.filter(f.col('smoking_status')!= 'Unknown')
```

Fig 1. Data Preparation

III. Model Building

To create the model, transformers were created to properly put the data correct type. The ‘age’ column utilized a Bucketizer to create 10 different age splits ranging from >10 to 80<. Columns

'gender', 'ever_married', 'work_type', 'Residence_type', and 'smoking_status' had a StringIndexer created to transform each response into a number. A VectorAssebler was created to place the 10 attributes into a vector so the Logistic Regression analysis can be performed. Because we are going to be hyper-tuning the Logistic Regression model, no specific parameters were given, so the default parameters were used. All of the transformers were placed into a pipeline, along with a vectorizer, and Logistic Regression Model (Fig. 2).

```
# Age data
agesplits = [-float("inf"), 10, 20, 30, 40, 50, 60, 70, 80, float("inf")]
ageBucketizer = Bucketizer(splits=agesplits, inputCol="age", outputCol="ageBucket")

# Gender
sexIndexer = StringIndexer(inputCol='gender', outputCol='sexIndex')

# ever_married
marriedIndex = StringIndexer(inputCol='ever_married', outputCol='marriedIndex')

# work_type
workIndex = StringIndexer(inputCol='work_type', outputCol = 'workIndex')

# redicence_type
residenceIndex = StringIndexer(inputCol='Residence_type', outputCol='residenceIndex')

# smoking_status
smokingIndex = StringIndexer(inputCol='smoking_status', outputCol = 'smokingIndex')
```

```
# Building the pipeline for the model
lr = LogisticRegression()

# Building pipeline
vecAssem = VectorAssembler(inputCols=['ageBucket', 'sexIndex', 'marriedIndex', 'workIndex', 'residenceIndex',
                                      'smokingIndex', 'avg_glucose_level', 'bmi', 'hypertension', 'heart_disease'], outputCol = 'features')
myStages = [ageBucketizer, sexIndexer, marriedIndex, workIndex, residenceIndex, smokingIndex, vecAssem, lr]
p = Pipeline(stages=myStages)
```

Fig 2. Pipeline Build

After building the pipeline, the training data was fit onto the pipeline, as well as transformed to obtain predictions. The accuracy and F1 were calculated since the target variable was binary and false positives/negatives are important in the prediction outcomes. A false negative or positive could prove costly in the real world if a patient is diagnosed incorrectly (Fig 3). An F1 score of 0.91126 was obtained, indicating the model is good, however, there could be some overfitting (Table 1).

```
# Looking at Logistic Model with no changed paramters
pModel = p.fit(trnStroke)
#Testing the model with testing data
Ppreds = pModel.transform(trnStroke)
print("Training Data")
Ppreds.select("label", "probability", "prediction").show(10)

#Evaluting the CV Model, also showing the best parameters by using the estimator parameter map to obtain the best model parameters
acc = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
f1 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
```

Fig 3. Pipeline Model, no tuning

```
Training Data
+-----+--------------------+----------+
|label|         probability|prediction|
+-----+--------------------+----------+
|    0|[0.99834325928448...|       0.0|
|    0|[0.99829651286408...|       0.0|
|    0|[0.99839407780579...|       0.0|
|    0|[0.99807227565883...|       0.0|
|    0|[0.99822032312717...|       0.0|
|    0|[0.99809971902501...|       0.0|
|    0|[0.99814991427746...|       0.0|
|    0|[0.99796096317805...|       0.0|
|    0|[0.99828254495270...|       0.0|
|    0|[0.99820086705974...|       0.0|
+-----+--------------------+----------+
only showing top 10 rows

Train Accuracy = 0.9399594320486815
Train F1 = 0.9112689168583318
```

Table 1. Logistic Regression (no tuning) results

IV. Model Hyperparameter Tuning and Selection

Cross-validation was performed to find the best parameters. The selected parameters were maxIter, regParam, and elasticNetParam. When building the parameter grid, the default parameters were also included to make sure it was compared to other parameters. Figure 4. shows the different parameters the CrossValidator, from the pyspark.ml.tuning package. To find the best parameters the F-1 score from the MulticlassClassificationEvaluator was utilized for the evaluator. The specific grid fit each model with parameters and the train datasets.

```
# Implementing a Cross Validation to select the best model for LR

# Creating the parameter grid for LR model,
paramGrid = ParamGridBuilder().addGrid(lr.maxIter, [15,20,30,100]).addGrid(lr.regParam, [0, 0.01, 0.1, 0.5]).addGrid(lr.elasticNetParam, [0, 0.1, 0.2, 0.5, 0.8]).build()

#Building Cross-Validation to get the best model
crossval = CrossValidator(estimator = p, estimatorParamMaps=paramGrid, evaluator=f1, numFolds=2)

# Training the model
cvModel = crossval.fit(trnStroke)

#Testing the model with testing data
preds = cvModel.transform(trnStroke)
print("Training Data")
preds.select("label", "probability", "prediction").show(10)

#Evaluting the CV Model, also showing the best parameters by using the estimator parameter map to obtain the best model parameters
print(cvModel.getEstimatorParamMaps()[np.argmax(cvModel.avgMetrics)])
```

Figure 4. Parameter Grid and Cross Validation Building

From the cross-validation, it was found that the best parameters found were maxIter (15), regParam (0.01), ElasticNet (0.1). The new LogisticRegression model was then used with these parameters, and the training dataset was fit and transformed to obtain the new F1 score and accuracy score. An F1 score of 0.91027 was obtained, indicating the model is still good. However, the F1 score is lower than the first model, but we may have minimized the overfitting of the data (Table 2).

```
Training Data
+-----+--------------------+----------+
|label|         probability|prediction|
+-----+--------------------+----------+
|    0|[0.99307960047664...|       0.0|
|    0|[0.99305170973809...|       0.0|
|    0|[0.99355033686607...|       0.0|
|    0|[0.99309415677628...|       0.0|
|    0|[0.99325865504813...|       0.0|
|    0|[0.99297760886204...|       0.0|
|    0|[0.99290406411878...|       0.0|
|    0|[0.99286103522676...|       0.0|
|    0|[0.99311147235323...|       0.0|
|    0|[0.99259137343836...|       0.0|
+-----+--------------------+----------+
only showing top 10 rows

Train Accuracy = 0.939553752535497
Train F1 = 0.9102725333077645
```

Table 2. Logistic Regression (tuning) results

V. Model Streaming Summary

To fit the test set, a stream was designed to read in 1 file per trigger. The files were created by repartitioning the testing data set into 10 partitions and then writing them to a folder. As the data was streamed into the new model and transformed, the accuracy, F-1 score, precision, and recall were evaluated (Fig 5).

```
# Source
sourceStream = spark.readStream.format("csv").option("header", True).schema(strokeSchema).option("mode", "dropMalformed").option("maxFilesPerTrigger",1).load("dbfs:///FileStore/tables/Stroke")
strokeStream = sourceStream.withColumnRenamed("stroke","label").withColumn("age", sourceStream["age"].cast(LongType()))

# Query

predsStream = BModel.transform(strokeStream).select("label", "probability", "prediction")

# Sink
sinkStream = predsStream.writeStream.outputMode("append").queryName("testpreds").format("memory").trigger(processingTime='10 seconds').start()
```

```
testpredDF = spark.sql("SELECT * FROM testpreds")

precision = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="precisionByLabel")
recall = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="recallByLabel")

print(f"Test Accuracy = {acc.evaluate(testpredDF)}")
print(f"Test F1 = {f1.evaluate(testpredDF)}")
print(f"Test Precision = {precision.evaluate(testpredDF)}")
print(f"Test Recall = {recall.evaluate(testpredDF)}")
```

Figure 5. Stream of test data set

In Table 3. the overall accuracy for the model is 0.9488 with an overall F1 score of 0.9239. The recall is 1.0 and the precision is 0.9488. With the recall being 1.0, there could be overfitting of the data and more tuning on the model is needed.

```
Test Accuracy = 0.9488188976377953
Test F1 = 0.9239004215382168
Test Precision = 0.9488188976377953
Test Recall = 1.0
```

Table 3. Test Set Evaluations

VI. Conclusion

By comparing the different parameters on the LogisiticRegression model it was found that the best parameters were maxIter (15), regParam (0.01), ElasticNet (0.1) for predicting whether a person is susceptible to a stroke or not. The model had an accuracy of 0.9488 and an overall F1 score of 0.9239.

To further improve the accuracy and F1 score, outliers could be looked at and removed on continuous variables of glucose and BMI, as well as age. In the dataset, there were some respondents who were less than 10 years old and had a smoking response, so these could have been looked at as outliers. Also, during hyperparameter tuning, expanding on different and current parameters could improve the model when implementing the CrossValidator.

References

Brain stroke prediction dataset. (2022, July 16). Kaggle. Retrieved March 11, 2023, from https://www.kaggle.com/datasets/zzettrkalpakbal/full-filled-brain-stroke-dataset

Stroke | cdc.gov. (2023, February 24). Centers for Disease Control and Prevention. https://www.cdc.gov/stroke/index.htm

Stroke Facts | cdc.gov. (2022, October 14). Centers for Disease Control and Prevention. https://www.cdc.gov/stroke/facts.htm