DS5559-Final_proj-NHAMCS-Report_2

June 28, 2020

1 Final Project: Admission Prediction from NHAMCS

1.1 Progress report: Inital model evaluation

1.1.1 DS5559: Big Data Analysis

1.1.2 Thomas Hartka(trh6u), Alicia Doan(ad2ew), Michael Langmayr(ml8vp)

Created: 6/28/20

In this report we demonstrate a logistic regression model. We use patient characteristics as input and create a model to prediction hospital admission. We transform our variables and create a vector of features. We then loop through values of reg for standard, ridge, and lasso methods. All the models use weights for the outcomes since there is class imbalance. The best hyperparameters were selected based on AUC.

The following is a summary of the best LR model:

- Type of model:
  - **Logistic Regression**
- Best hyperparameters used:
  - **Method: Ridge regression**
  - **Regularization parameter: 1.0**
- Size of the saved model:
  - **Disk usage: 52k**
- Performance metrics:
  - **Accuracy: 0.713**
  - **Area under ROC curve (AUROC): 0.755**
  - **F1 score: 0.183**
  - **Confusion matrix:**
    tn: 1473 fn: 45
    fp: 587 tp: 99

1.2 Configuration

[1]:
```
# set data directory
data_dir = "../data"
results_dir = "../results"
```

1.3 Import libraries and set up Spark

[2]:
```
# import python libraries
import os
import pandas as pd
import numpy as np
from functools import reduce
```

[3]:
```
# set up pyspark
from pyspark.sql import *
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import IntegerType
```

[4]:
```
from pyspark.ml import Pipeline
from pyspark.ml.feature import *
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

[5]:
```
spark = SparkSession.builder.getOrCreate()
```

1.4 Read in data

[6]:
```
NHAMCS = spark.read.parquet(data_dir + "/NHAMCS_processed.2007-2017")
```

1.5 Transform data

[7]:
```
# perform string indexing to prepare for OHE for residence variable
rsi = StringIndexer(inputCol="RESIDNCE", outputCol="RESINDEX")
simodel = rsi.fit(NHAMCS)
NHAMCS = simodel.transform(NHAMCS)
```

[8]:
```
# perform OHE on residence variable
rohe = OneHotEncoder(inputCol='RESINDEX', outputCol='RESONE')
NHAMCS = rohe.transform(NHAMCS)
```

[9]:
```
# assemble vector
va =␣
↪VectorAssembler(inputCols=["AGEYEAR","RESONE",'SEXMALE','ARRTIMEMIN','YEAR','PULSE','TEMPF'
↪\
                                                                  ␣
↪'RESPR','BPSYS','BPDIAS','POPCT','PAINSCALE','ALZHD','ASTHMA','CAD','CANCER',␣
↪\
                                                                  ␣
↪'CEBVD','CHF','CKD','COPD','DEPRN','DIABTYP0','DIABTYP1','DIABTYP2','EDHIV',␣
↪\
                                                                  ␣
↪'ESRD','ETOHAB','HPE','HTN','HYPLIPID','OBESITY','OSA','OSTPRSIS','SUBSTAB',␣
↪\
                                  'NOCHRON','TOTCHRON','INJURY','INJURY72'],
                          outputCol="features")

NHAMCS = va.setHandleInvalid("skip").transform(NHAMCS)
```

1.6 Train and test model

[10]:
```
# split into training and testing set
training, testing = NHAMCS.randomSplit([0.8, 0.2], 42)
```

[11]:
```
# handle class imbalance

# calculate balance ratio
balRatio = training.select("ADM_OUTCOME").where('ADM_OUTCOME == 0').count() /␣
↪training.count()

# add weights
training = training.withColumn("classWeights", when(training.ADM_OUTCOME ==␣
↪1,balRatio).otherwise(1-balRatio))
```

[12]:
```
# function for logistic regression
def lr_nhamcs (training_set, testing_set, reg_param=0, method="Standard"):
    if method=="Standard":
        lr = LogisticRegression(featuresCol="features", labelCol="ADM_OUTCOME",␣
↪weightCol="classWeights", \
                                  maxIter=10, regParam=0, elasticNetParam=0)
    elif method=="Ridge":
        lr = LogisticRegression(featuresCol="features", labelCol="ADM_OUTCOME",␣
↪weightCol="classWeights", \
                                  maxIter=10, regParam=reg_param,␣
↪elasticNetParam=0)
    elif method=="Lasso":
```

```
        lr = LogisticRegression(featuresCol="features", labelCol="ADM_OUTCOME",␣
↪weightCol="classWeights", \
                                    maxIter=10, regParam=reg_param,␣
↪elasticNetParam=1)

    # Fit the model
    admModel = lr.fit(training_set)

    # predict on testing set
    predict_test=admModel.transform(testing_set)

    # make evaluator
    evaluator=BinaryClassificationEvaluator(rawPredictionCol="rawPrediction",␣
↪labelCol="ADM_OUTCOME")

    return evaluator.evaluate(predict_test)
```

1.7 Determine best hyperparameters

[13]:
```
# test standard LR model
print("ROC-AUC for standard LR is: ", lr_nhamcs(training,testing))
```

```
ROC-AUC for standard LR is:  0.7540790183387306
```

[14]:
```
# test Ridge LR model for different values of the regularizatoin parameter
for i in np.arange(0.0, 1.1, 0.1):
    i = np.round(i,1)
    print("ROC-AUC for Ridge LR with reg_param=", i, \
          " is: ", lr_nhamcs(training,testing, i,"Ridge"))
```

```
ROC-AUC for Ridge LR with reg_param= 0.0  is:  0.7540790183387306
ROC-AUC for Ridge LR with reg_param= 0.1  is:  0.7540419363538331
ROC-AUC for Ridge LR with reg_param= 0.2  is:  0.7540756472491944
ROC-AUC for Ridge LR with reg_param= 0.3  is:  0.7542239751887837
ROC-AUC for Ridge LR with reg_param= 0.4  is:  0.754362189859765
ROC-AUC for Ridge LR with reg_param= 0.5  is:  0.7542273462783199
ROC-AUC for Ridge LR with reg_param= 0.6  is:  0.7541666666666701
ROC-AUC for Ridge LR with reg_param= 0.7  is:  0.7541026159654836
ROC-AUC for Ridge LR with reg_param= 0.8  is:  0.7540756472491945
ROC-AUC for Ridge LR with reg_param= 0.9  is:  0.7544869201726022
ROC-AUC for Ridge LR with reg_param= 1.0  is:  0.7550094390507027
```

[15]:
```
# test Lasso LR model for different values of the regularizatoin parameter
for i in np.arange(0.0, 1.1, 0.1):
    i = np.round(i,1)
    print("ROC-AUC for Lasso LR with reg_param=", i, \
```

```
        " is: ", lr_nhamcs(training,testing, i,"Lasso"))
```

```
ROC-AUC for Lasso LR with reg_param= 0.0  is:  0.7540790183387306
ROC-AUC for Lasso LR with reg_param= 0.1  is:  0.7391906014023741
ROC-AUC for Lasso LR with reg_param= 0.2  is:  0.5
ROC-AUC for Lasso LR with reg_param= 0.3  is:  0.5
ROC-AUC for Lasso LR with reg_param= 0.4  is:  0.5
ROC-AUC for Lasso LR with reg_param= 0.5  is:  0.5
ROC-AUC for Lasso LR with reg_param= 0.6  is:  0.5
ROC-AUC for Lasso LR with reg_param= 0.7  is:  0.5
ROC-AUC for Lasso LR with reg_param= 0.8  is:  0.5
ROC-AUC for Lasso LR with reg_param= 0.9  is:  0.5
ROC-AUC for Lasso LR with reg_param= 1.0  is:  0.5
```

1.8 Size of saved model

[16]:
```
# create model with best hyperparameters (Ridge, regParam=1.0)
lr = LogisticRegression(featuresCol="features", labelCol="ADM_OUTCOME",␣
 ↪weightCol="classWeights", \
                        maxIter=10, regParam=1.0, elasticNetParam=0)

admModel = lr.fit(training)
```

[17]:
```
# save model
admModel.write().overwrite().save("../models/001-log_regress-no_RFV")
```

[18]:
```
# get size on disk
!du -h ../models/001-log_regress-no_RFV
```

```
28K     ../models/001-log_regress-no_RFV/data
20K     ../models/001-log_regress-no_RFV/metadata
52K     ../models/001-log_regress-no_RFV
```

1.9 Get evaluation metrics

[19]:
```
# predict on testing set
predict_test=admModel.transform(testing)
```

[20]:
```
# calculate AUC
evaluator=BinaryClassificationEvaluator(rawPredictionCol="rawPrediction",␣
 ↪labelCol="ADM_OUTCOME")
print("ROC-AUC:", evaluator.evaluate(predict_test))
```

```
ROC-AUC: 0.7550094390507027
```

[21]:

```
# calculate accuracy
print("F1 score:",evaluator.setMetricName("areaUnderPR").evaluate(predict_test))
```

```
F1 score: 0.18282251646509748
```

[22]:

```
# calculate accuracy
correct = predict_test.where('prediction == ADM_OUTCOME').count()
total = predict_test.count()

print("Accuracy:", correct/total)
```

```
Accuracy: 0.7132486388384754
```

[24]:

```
# compute confusion matrix
tp = predict_test.where('prediction == 1 and ADM_OUTCOME==1').count()
fp = predict_test.where('prediction == 1 and ADM_OUTCOME==0').count()
tn = predict_test.where('prediction == 0 and ADM_OUTCOME==0').count()
fn = predict_test.where('prediction == 0 and ADM_OUTCOME==1').count()

print("\nConfusion Matrix:")
print('tn:',tn,' fn:',fn)
print('fp:',fp, '  tp:',tp,)
```

```

Confusion Matrix:
tn: 1473  fn: 45
fp: 587   tp: 99
```

[ ]:

```
# convert to PDF
!jupyter nbconvert --to pdf `pwd`/DS5559-Final_proj-NHAMCS-Report_2.ipynb
```

[ ]:

```

```