classification_wisc_breast_cancer_solutions

June 10, 2020

0.0.1 University of Virginia

0.0.2 DS 5559: Big Data Analytics

0.0.3 Classification of Wisconsin Breast Cancer Database

0.0.4 Last updated: Feb 20, 2020

Instructions

In this project, you will work with the Wisconsin Breast Cancer dataset. You will train a logistic regression model to predict the diagnosis. First, you will work through this example. Then you will make modifications and run the code, collecting results at the bottom of the notebook.

The following experiments should be conducted: 1. Three features were used in the model. Build the model using all features. Before training the model, apply scaling to the features using the StandardScaler transformer. Then train the model and compute the accuracy on the test set. Additionally, compute and show the confusion matrix.

2. Repeat step (1), including an intercept
3. Repeat step (1), using randomSplit([0.7, 0.3])
4. Repeat step (2), using randomSplit([0.7, 0.3])
5. Compare and discuss the results of (1) vs (2). Compare and discuss the results of (3) vs (4).

Total Possible Points: 10

[50]:
```
# load modules
from pyspark.sql import SparkSession
from pyspark.mllib.classification import LogisticRegressionWithLBFGS,␣
 ↪LogisticRegressionModel
from pyspark.mllib.regression import LabeledPoint
from pyspark.ml.feature import VectorAssembler
from pyspark.mllib.linalg import Vectors
from pyspark.sql.functions import col
from pyspark.mllib.evaluation import MulticlassMetrics

import os
```

[51]:
```
# param init (you will need to update data_path)
infile = 'wisc_breast_cancer_w_fields.csv'
```

```
spark = SparkSession \
    .builder \
    .appName("Wisc BRCA") \
    .getOrCreate()
```

[52]:
```
# read in data
df = spark.read.csv(infile, inferSchema=True, header = True)
```

[17]:
```
# use some of the fields as features
assembler = VectorAssembler(inputCols=["f1", "f2", "f3"], outputCol="features")
transformed = assembler.transform(df)
```

[18]:
```
# convert to RDD
dataRdd = transformed.select("diagnosis", "features").rdd.map(tuple)
```

[24]:
```
# look at some data
dataRdd.take(2)
```

[21]:
```
# map label to binary values, then convert to LabeledPoint
lp = dataRdd.map(lambda row:(1 if row[0]=='M' else 0, Vectors.dense(row[1])))␣
 ↪ \
                    .map(lambda row: LabeledPoint(row[0], row[1]))
```

[25]:
```
# look at some data
lp.take(2)
```

[26]:
```
# Split data approximately into training (60%) and test (40%)
training, test = lp.randomSplit([0.6, 0.4], seed=314)
```

[28]:
```
# count records in datasets
(training.count(), test.count(), lp.count())
```

[ ]:
```
(training.count()/lp.count(), test.count()/lp.count(), lp.count()/lp.count())
```

[29]:
```
# Build the model
model = LogisticRegressionWithLBFGS.train(training)
```

[ ]:
```
# Evaluating the model on test data
labelsAndPreds_te = test.map(lambda p: (p.label, model.predict(p.features)))
accuracy_te = 1.0 * labelsAndPreds_te.filter(lambda pl: pl[0] == pl[1]).count()␣
 ↪/ test.count()
print('model accuracy (test): {}'.format(accuracy_te))
```

SOLUTIONS

For parts 1-4, compute and show for the test set: (1) accuracy (2) confusion matrix.
Each part is worth 2 POINTS.

[54]:
```
## Enter solution for Part 1

# use all of the fields as features
assembler = VectorAssembler(inputCols=[i for i in df.columns if i[0]=='f'],␣
↪outputCol="features")
transformed = assembler.transform(df)

from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scalerModel = scaler.fit(transformed)
scaledData = scalerModel.transform(transformed)

# convert to RDD
dataRdd = scaledData.select("diagnosis","scaledFeatures").rdd.map(tuple)

# map label to binary values, then create LabeledPoints
lp = dataRdd.map(lambda row: LabeledPoint(1 if row[0]=='M' else 0, Vectors.
↪dense(row[1])))

# Split data approximately into training (60%) and test (40%)
training, test = lp.randomSplit([0.6, 0.4], seed=314)

# Build the model
model = LogisticRegressionWithLBFGS.train(training)

# Evaluating the model on test data

# make sure predictions are floats
labelsAndPreds_te = test.map(lambda p: (p.label, float(model.predict(p.
↪features))))
accuracy_te = 1.0 * labelsAndPreds_te.filter(lambda pl: pl[0] == pl[1]).count()␣
↪/ test.count()
print('model accuracy (test): {}'.format(accuracy_te))

metrics = MulticlassMetrics(labelsAndPreds_te)
labelsAndPreds_te.take(3)
print("Confusion Matrix:\n{}".format(metrics.confusionMatrix().toArray()))
```

```
model accuracy (test): 0.9624413145539906
Confusion Matrix:
[[135.   4.]
 [  4.  70.]]
```

[55]:
```
## Enter solution for Part 2
```

```
assembler = VectorAssembler(inputCols=[i for i in df.columns if i[0]=='f'],␣
 ↪outputCol="features")
transformed = assembler.transform(df)

from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scalerModel = scaler.fit(transformed)
scaledData = scalerModel.transform(transformed)

dataRdd = scaledData.select("diagnosis","scaledFeatures").rdd.map(tuple)

lp = dataRdd.map(lambda row: LabeledPoint(1 if row[0]=='M' else 0, Vectors.
 ↪dense(row[1])))

training, test = lp.randomSplit([0.6, 0.4], seed=314)

model = LogisticRegressionWithLBFGS.train(training, intercept=True)

labelsAndPreds_te = test.map(lambda p: (p.label, float(model.predict(p.
 ↪features))))
accuracy_te = 1.0 * labelsAndPreds_te.filter(lambda pl: pl[0] == pl[1]).count()␣
 ↪/ test.count()
print('model accuracy (test): {}'.format(accuracy_te))

metrics = MulticlassMetrics(labelsAndPreds_te)
labelsAndPreds_te.take(3)
print("Confusion Matrix:\n{}".format(metrics.confusionMatrix().toArray()))
```

```
model accuracy (test): 0.9671361502347418
Confusion Matrix:
[[135.   3.]
 [  4.  71.]]
```

[56]:
```
## Enter solution for Part 3

assembler = VectorAssembler(inputCols=[i for i in df.columns if i[0]=='f'],␣
 ↪outputCol="features")
transformed = assembler.transform(df)

from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scalerModel = scaler.fit(transformed)
scaledData = scalerModel.transform(transformed)

dataRdd = scaledData.select("diagnosis","scaledFeatures").rdd.map(tuple)
```

```
lp = dataRdd.map(lambda row: LabeledPoint(1 if row[0]=='M' else 0, Vectors.
 ↪dense(row[1])))

training, test = lp.randomSplit([0.7, 0.3], seed=314)

model = LogisticRegressionWithLBFGS.train(training)

labelsAndPreds_te = test.map(lambda p: (p.label, float(model.predict(p.
 ↪features))))
accuracy_te = 1.0 * labelsAndPreds_te.filter(lambda pl: pl[0] == pl[1]).count()␣
 ↪/ test.count()
print('model accuracy (test): {}'.format(accuracy_te))

metrics = MulticlassMetrics(labelsAndPreds_te)
labelsAndPreds_te.take(3)
print("Confusion Matrix:\n{}".format(metrics.confusionMatrix().toArray()))
```

```
model accuracy (test): 0.9433962264150944
Confusion Matrix:
[[98.  6.]
 [ 3. 52.]]
```

[57]:
```
## Enter solution for Part 4

assembler = VectorAssembler(inputCols=[i for i in df.columns if i[0]=='f'],␣
 ↪outputCol="features")
transformed = assembler.transform(df)

from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scalerModel = scaler.fit(transformed)
scaledData = scalerModel.transform(transformed)

dataRdd = scaledData.select("diagnosis","scaledFeatures").rdd.map(tuple)

lp = dataRdd.map(lambda row: LabeledPoint(1 if row[0]=='M' else 0, Vectors.
 ↪dense(row[1])))

training, test = lp.randomSplit([0.7, 0.3], seed=314)

model = LogisticRegressionWithLBFGS.train(training, intercept=True)

labelsAndPreds_te = test.map(lambda p: (p.label, float(model.predict(p.
 ↪features))))
```

```
accuracy_te = 1.0 * labelsAndPreds_te.filter(lambda pl: pl[0] == pl[1]).count()␣
 ↪/ test.count()
print('model accuracy (test): {}'.format(accuracy_te))

metrics = MulticlassMetrics(labelsAndPreds_te)
labelsAndPreds_te.take(3)
print("Confusion Matrix:\n{}".format(metrics.confusionMatrix().toArray()))
```

```
model accuracy (test): 0.9371069182389937
Confusion Matrix:
[[98.  7.]
 [ 3. 51.]]
```

Enter solution for Part 5 **Compare and discuss the results of (1) vs (2).**
Looking at the results between 1 and 2, they appear similar; adding the intercept appears to improve the model predictive ability. However, it is worth noting that only one seed has been used and this hasn't been cross-validated. Until one of both of those is performed, it is tough to say one way or another if adding an intercept actually improves the model.

Compare and discuss the results of (3) vs (4).
Looking at the results between 3 and 4, where we are adding additional data to our model (via using a 70-30 split as opposed to a 60-40), it appears that the addition of an intercept actually hurts the model predictive ability.

```
[ ]:
```