Chapter 13: PCA and Logistic Regression

In [1]:
```
import findspark
findspark.init()
```

In [2]:
```
from pyspark.sql import SparkSession
```

In [3]:
```
spark = SparkSession.builder.appName('PCA_sonar').getOrCreate()
```

In [4]:
```
df = spark.read.csv('sonar_all_data.txt',inferSchema=True,header=False)
```

In [33]:
```
str(df.schema.names)
```

Out[33]:
```
"['_c0', '_c1', '_c2', '_c3', '_c4', '_c5', '_c6', '_c7', '_c8', '_c9', '_c10',
'_c11', '_c12', '_c13', '_c14', '_c15', '_c16', '_c17', '_c18', '_c19', '_c20',
'_c21', '_c22', '_c23', '_c24', '_c25', '_c26', '_c27', '_c28', '_c29', '_c30',
'_c31', '_c32', '_c33', '_c34', '_c35', '_c36', '_c37', '_c38', '_c39', '_c40',
'_c41', '_c42', '_c43', '_c44', '_c45', '_c46', '_c47', '_c48', '_c49', '_c50',
'_c51', '_c52', '_c53', '_c54', '_c55', '_c56', '_c57', '_c58', '_c59', 'labe
l']"
```

In [6]:
```
df = df.withColumnRenamed("_c60","label")
```

In [7]:
```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import PCA
```

In [8]:
```
assembler = VectorAssembler(
    inputCols=['_c%d' % i for i in range(60)],
    outputCol="features")
output = assembler.transform(df)
```

In [9]:
```
output.select("features").show(1, truncate=False)
```

```
+--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------------+
|features
|
+--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------------+
|[0.02,0.0371,0.0428,0.0207,0.0954,0.0986,0.1539,0.1601,0.3109,0.2111,0.1609,0.
1582,0.2238,0.0645,0.066,0.2273,0.31,0.2999,0.5078,0.4797,0.5783,0.5071,0.4328,
0.555,0.6711,0.6415,0.7104,0.808,0.6791,0.3857,0.1307,0.2604,0.5121,0.7547,0.85
37,0.8507,0.6692,0.6097,0.4943,0.2744,0.051,0.2834,0.2825,0.4256,0.2641,0.1386,
0.1051,0.1343,0.0383,0.0324,0.0232,0.0027,0.0065,0.0159,0.0072,0.0167,0.018,0.0
084,0.009,0.0032]|
+--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------------+
only showing top 1 row
```

Scale feature if we need

In [10]:
```
standardizer = StandardScaler(withMean=True, withStd=True,
                              inputCol='features',
                              outputCol='std_features')
model = standardizer.fit(output)
output = model.transform(output)
```

Label to String

In [11]:
```
indexer = StringIndexer(inputCol="label", outputCol="label_idx")
indexed = indexer.fit(output).transform(output)
```

In [12]:
```
final_data = indexed.select(['std_features', 'label', 'label_idx'])
```

In [13]:
```
final_data.show(3)
```

```
+--------------------+-----+---------+
|        std_features|label|label_idx|
+--------------------+-----+---------+
|[-0.3985897356694...|    R|      1.0|
|[0.70184498705605...|    R|      1.0|
|[-0.1289179854363...|    R|      1.0|
+--------------------+-----+---------+
only showing top 3 rows
```

PCA

In [14]:
```
pca = PCA(k=15, inputCol="std_features", outputCol="pca")
model = pca.fit(final_data)
```

In [15]:
```
model.explainedVariance
```

Out[15]:
```
DenseVector([0.2035, 0.189, 0.0855, 0.0568, 0.0501, 0.0406, 0.0328, 0.0305, 0.0
257, 0.0249, 0.0208, 0.019, 0.0175, 0.0154, 0.0143])
```

In [16]:
```
percent = model.explainedVariance
type(percent)
```

Out[16]:
```
pyspark.ml.linalg.DenseVector
```

In [17]:
```
percent.values.sum()
```

Out[17]:
```
0.8261807898020073
```

In [18]:
```
transformed = model.transform(final_data)
```

In [19]:
```
transformed.show(3)
```

```
+--------------------+-----+---------+--------------------+
|        std_features|label|label_idx|                 pca|
+--------------------+-----+---------+--------------------+
|[-0.3985897356694...|    R|      1.0|[-1.9165444107164...|
|[0.70184498705605...|    R|      1.0|[0.47896904316845...|
|[-0.1289179854363...|    R|      1.0|[-3.8499400285258...|
+--------------------+-----+---------+--------------------+
only showing top 3 rows
```

In [20]:
```
final_data = transformed.select("label_idx","pca")
```

Logistic Regression

In [21]:
```
train_data,test_data = final_data.randomSplit([0.8,0.2])
```

In [22]:
```
from pyspark.ml.classification import LogisticRegression
```

In [23]:
```
logistic = LogisticRegression(featuresCol='pca',
                              labelCol='label_idx',
                              predictionCol='prediction')
```

In [24]:
```
# Fit the model to the data and call this model logisticModel
logisticModel = logistic.fit(train_data)
```

In [25]:
```
# Create predictions for the testing data and show confusion matrix
test_model = logisticModel.transform(test_data)
test_model.groupBy('label_idx', 'prediction').count().show()
```

```
+---------+----------+-----+
|label_idx|prediction|count|
+---------+----------+-----+
|      1.0|       1.0|   14|
|      0.0|       1.0|    2|
|      1.0|       0.0|   10|
|      0.0|       0.0|   19|
+---------+----------+-----+
```

In [26]:
```
# Calculate the elements of the confusion matrix
TN = test_model.filter('prediction = 0 AND label_idx = prediction').count()
TP = test_model.filter('prediction = 1 AND label_idx = prediction').count()
FN = test_model.filter('prediction = 0 AND label_idx != prediction').count()
FP = test_model.filter('prediction = 1 AND label_idx != prediction').count()
```

In [27]:
```
# Calculate precision and recall
precision = TP / (TP + FP)
recall = TP / (TP + FN)
print('precision = {:.2f}\nrecall    = {:.2f}'.format(precision, recall))
```

```
precision = 0.88
recall    = 0.58
```

In [28]:
```
acc =(TP+TN)/test_model.count()
acc
```

Out[28]:
```
0.7333333333333333
```