### 

Big Data Analytics with Spark

Guyland Gore

Shannon Lamberson

Mayur Vira

Pranathi Donthineni

### 

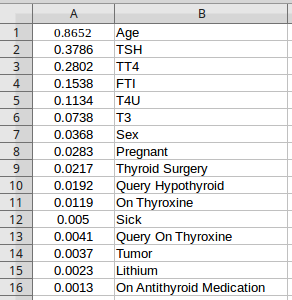
### 

#### 

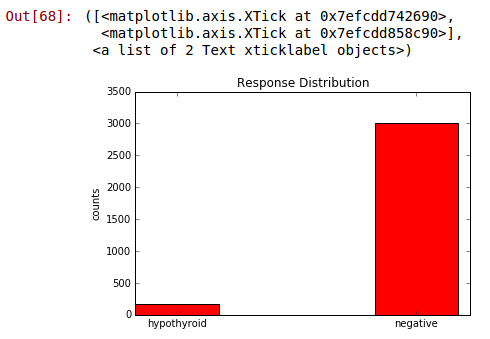
We obtained a dataset composed of three thousand rows that related to individuals diagnosed with hypothyroidism. Cleaning the data involved: removing records with missing values, deletion of attributes, correcting for imbalance, and vectorizing feature values. There were a number of variables we could choose as features and they related to the patient in several ways: some were characteristics of the individual such as age, others were related to hormone levels, and still others related to the patient’s history like whether or not they’d undergone thyroid surgery in the past. Our goal is to determine if any (and which) of the variables will help us determine with some certainty that a patient has hypothyroidism, detail which features we chose, run a series of classification algorithms, and describe our findings.

One of the first things we learned is that setting up the analytical environment can be troublesome. After several failed attempts to get the VM to run MLLib imports we began to experiment with alternative strategies including the Data Science Experience and DataBricks. Eventually we settled on the DSE, and after some trouble in getting the dataset imported we were finally on track and ready for some analytics.

First, we wanted to make some determinations on what features were most important while screening for false predictors. Since we had no subject matter expert, we used feature selection techniques to screen all features with Spark:



So our six features became: age, TSH, TT4, FTI, T4U, and F3. Ironically, we found that “on antithyroid medication” was a horrible predictor. We realized that while someone taking this medication has hypothyroidism, there were enough people who had the condition but weren’t yet prescribed this medicine to bias the feature so we excluded it. Also, we had to vectorize all values and run the above code twice, once for nominal features and once for numeric.



For the analysis we decided to focus on classification and chose to use the Random Forest technique because random forests are accurate, fast, and can control for unbalanced datasets. Our dataset targets were highly imbalanced with ‘negative’ results for hypothyroidism making up close to all of the results so controlling for imbalance was important. This was the “Correcting for Imbalance” part of cleaning.

We employed the random forest with the following code (edited for brevity):

trainingData, testData = allData.randomSplit([0.8,0.2], seed = 0)

rf = RF(labelCol='predict', featuresCol='tests', numTrees=50)  
fit = rf.fit(trainingData)  
transformed = fit.transform(testData)

results = transformed.select(['probability','predict'])

results\_collect = results.collect()  
results\_list = [(float(i[0][0]),1.0-float(i[1])) for i in results\_collect]  
scoreAndLabels = sc.parallelize(results\_list)

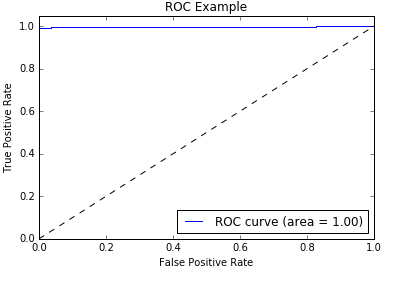
metrics = metric(scoreAndLabels)  
print("The ROC score is (@numTrees=50):", metrics.areaUnderROC)

('The ROC score is (@numTrees=50):', 0.9974782084315551)

…

plt.show()

We ran the first experiment with all 17 features and got a ROC curve of .997. To ensure we weren’t seeing overfitting from noise, we reran it with just the six features suggested by the feature selection and got a ROC curve of .994. This is the “Deletion of Attributes” part of cleaning.



The curve suggests a correct prediction in almost every case. At first this seems unbelievable and indicative of a false predictor. However, since random forests “correct for decision trees' habit of overfitting to their training set” [1], and because the information gain we saw from our selected variables capped at 86% prediction value, we felt confident that we had no false predictors and that our findings were valid.

***Notes:***

We were happily surprised to find such high efficacy in the random forest algorithm. It was fast and returned results that are undeniably significant. Our team discovered that the random forest algorithm is easy enough to employ yet configurable to a very high degree. We were amazed at how many additional techniques could be applied with this one algorithm. Additionally, we were reminded of how complex data science can be. Many hours we spent on just trying to get data loaded or get an algorithm to run with no errors.

This exercise helped each of us see just how powerful Spark can be if properly wielded. Understanding techniques for last mile computing and Spark’s machine learning library has helped all of us see how to create value by finding insights in data.

Please note, in the feature selection code, the URL given is an IP address. Guy did in fact set up a webserver on an old laptop, create a directory to act as a web page, drop our file into that directory, enable port forwarding on his router, and then use his public IP address in DataBricks in order to make use of the read\_csv method as part of the environment. Otherwise, using the default data loading method in DataBricks gave an incompatible data type for running ExtraTreesClassifier.fit().

Sources Used

[1] - <https://en.wikipedia.org/wiki/Random_forest>

<http://machinelearningmastery.com/feature-selection-machine-learning-python/>

***Code:***

***Feature Selection***

**from** pandas **import** read\_csv

**from** sklearn.ensemble **import** ExtraTreesClassifier

url = "<http://192.136.208.3:8080/fileupload/file.csv>" #URL of webserver we build to host our file publicly

names = ['age', 'sex', 'on\_thyroxine', 'query\_on\_thyroxine', 'on\_antithyroid\_medication', 'thryoid\_surgery', 'query\_hypothyroid', 'query\_hyperthyroid', 'preg', 'sick', 'tumor', 'lithium', 'TSH', 'T3', 'TT4', 'T4U', 'FTI', 'target']

dataframe = read\_csv(url, names=names)

**print** dataframe.values

array = dataframe.values

#Run once on cols 0:11 (categorical), then a second time (not shown) on 12:17 for feature selection on numeric values. Note, when we ran all 17 through Weka to confirm our findings we had identical results.

X = array[:,0:11]

Y = array[:,17]

model = ExtraTreesClassifier()

model.fit(X, Y)

**print**(model.feature\_importances\_)

***Random Forest***

In [65]:

**from** **pyspark.sql** **import** SQLContext  
sqlContext = SQLContext(sc)  
**from** **pyspark.sql** **import** SQLContext  
sqlContext = SQLContext(sc)  
  
**def** set\_hadoop\_config\_with\_credentials(name):  
 hconf = sc.\_jsc.hadoopConfiguration()  
 hconf.setBoolean('.public', True)  
  
name = ‘project’  
set\_hadoop\_config\_with\_credentials(name)  
  
df\_data\_1 = sqlContext.read.format('com.databricks.spark.csv')\  
 .options(header='true', inferschema='true')\  
 .load("swift://BigDataFinalProject." + name + "/ThyroidCleanedData3.csv")  
df\_data\_1.take(5)

Out[65]:

[Row(age=72, TSH=30.0, T3=0.6, TT4=15.0, T4U=1.48, FTI=10.0, on\_thyroxine=u'f', query\_on\_thyroxine=u'f', thyroid\_surgery=u'f', query\_hypothyroid=u'f', query\_hyperthyroid=u'f', pregnant=u'f', sick=u'f', goitre=u'f', Result=u'hypothyroid'),  
 Row(age=15, TSH=145.0, T3=1.7, TT4=19.0, T4U=1.13, FTI=17.0, on\_thyroxine=u't', query\_on\_thyroxine=u'f', thyroid\_surgery=u'f', query\_hypothyroid=u'f', query\_hyperthyroid=u'f', pregnant=u'f', sick=u'f', goitre=u'f', Result=u'hypothyroid'),  
 Row(age=24, TSH=0.0, T3=0.2, TT4=4.0, T4U=1.0, FTI=0.0, on\_thyroxine=u'f', query\_on\_thyroxine=u'f', thyroid\_surgery=u'f', query\_hypothyroid=u'f', query\_hyperthyroid=u'f', pregnant=u'f', sick=u'f', goitre=u'f', Result=u'negative'),  
 Row(age=24, TSH=430.0, T3=0.4, TT4=6.0, T4U=1.04, FTI=6.0, on\_thyroxine=u'f', query\_on\_thyroxine=u'f', thyroid\_surgery=u'f', query\_hypothyroid=u'f', query\_hyperthyroid=u'f', pregnant=u'f', sick=u'f', goitre=u'f', Result=u'hypothyroid'),  
 Row(age=77, TSH=7.3, T3=1.2, TT4=57.0, T4U=1.28, FTI=44.0, on\_thyroxine=u'f', query\_on\_thyroxine=u'f', thyroid\_surgery=u'f', query\_hypothyroid=u'f', query\_hyperthyroid=u'f', pregnant=u'f', sick=u'f', goitre=u'f', Result=u'hypothyroid')]

In [66]:

**from** **pyspark.ml** **import** Pipeline  
**from** **pyspark.ml.classification** **import** DecisionTreeClassifier  
**from** **pyspark.ml.classification** **import** RandomForestClassifier **as** RF  
**from** **pyspark.ml.clustering** **import** KMeans  
**from** **pyspark.ml.feature** **import** StringIndexer, VectorIndexer, VectorAssembler, SQLTransformer  
**from** **pyspark.ml.evaluation** **import** MulticlassClassificationEvaluator, BinaryClassificationEvaluator  
**from** **pyspark.ml.tuning** **import** CrossValidator, ParamGridBuilder  
**import** **pandas** **as** **pd**  
**import** **numpy** **as** **np**  
**import** **functools**  
**from** **pyspark.ml.feature** **import** OneHotEncoder

In [67]:

cols\_select = ['age', 'TSH','T3','TT4','T4U','FTI','on\_thyroxine','query\_on\_thyroxine', 'thyroid\_surgery','query\_hypothyroid',  
 'query\_hyperthyroid','pregnant','sick','goitre' ,'Result']  
  
  
df = tableData.select(cols\_select)

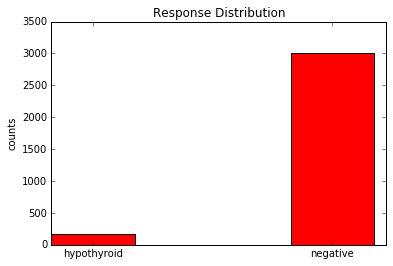
In [68]:

**from** **matplotlib** **import** pyplot **as** plt  
%**matplotlib** inline  
**print**("Histograms: ")  
  
responses = df.groupBy('Result').count().collect()  
categories = [i[0] **for** i **in** responses]  
counts =[i[1] **for** i **in** responses]  
  
ind = np.array(range(len(categories)))  
width = 0.35  
plt.bar(ind, counts, width=width, color ='r')  
  
plt.ylabel('counts')  
plt.title('Response Distribution')  
plt.xticks(ind + width/2., categories)

Histograms:

Out[68]:

([<matplotlib.axis.XTick at 0x7efcdd742690>,  
 <matplotlib.axis.XTick at 0x7efcdd858c90>],  
 <a list of 2 Text xticklabel objects>)



In [69]:

cols\_select = ['age', 'TSH','T3','TT4','T4U','FTI','on\_thyroxine','query\_on\_thyroxine', 'thyroid\_surgery','query\_hypothyroid',  
 'query\_hyperthyroid','pregnant','sick','goitre','Result']  
  
df = df.select(df.age.cast('int'),  
 df.TSH.cast('float'),  
 df.T3.cast('float'),  
 df.TT4.cast('float'),  
 df.T4U.cast('float'),  
 df.FTI.cast('float'),  
 \*cols\_select[6:])

In [70]:

**for** col **in** df.columns[6:-2]:  
 **print**(col,df.select(col).distinct().count())

('on\_thyroxine', 2)  
('query\_on\_thyroxine', 2)  
('thyroid\_surgery', 2)  
('query\_hypothyroid', 2)  
('query\_hyperthyroid', 2)  
('pregnant', 2)  
('sick', 2)

In [71]:

#vectorize values, change categories to numeric by replacing with an int

column\_vec\_in = ['on\_thyroxine','query\_on\_thyroxine', 'thyroid\_surgery','query\_hypothyroid',  
 'query\_hyperthyroid','pregnant','sick','goitre' ]  
  
column\_vec\_out= ['on\_thyroxine\_vec','query\_on\_thyroxine\_vec', 'thyroid\_surgery\_vec','query\_hypothyroid\_vec',  
 'query\_hyperthyroid\_vec','pregnant\_vec','sick\_vec','goitre\_vec']  
  
indexers = [StringIndexer(inputCol=x, outputCol=x+'\_tmp') **for** x **in** column\_vec\_in]  
  
encoders = [OneHotEncoder(dropLast=False, inputCol =x+"\_tmp", outputCol=y)  
 **for** x,y **in** zip(column\_vec\_in, column\_vec\_out)]  
  
tmp =[[i,j]**for** i,j **in** zip(indexers, encoders)]  
tmp = [i **for** sublist **in** tmp **for** i **in** sublist]

In [73]:

cols\_now = ['age',   
 'TSH',  
 'T3',  
 'TT4',  
 'T4U',  
 'FTI',  
 'on\_thyroxine\_vec',  
 'query\_on\_thyroxine\_vec',   
 'thyroid\_surgery\_vec',  
 'query\_hypothyroid\_vec',  
 'query\_hyperthyroid\_vec',  
 'pregnant\_vec',  
 'sick\_vec',  
 'goitre\_vec']  
  
  
assembler\_features = VectorAssembler(inputCols=cols\_now, outputCol='tests')  
labelIndexer = StringIndexer(inputCol='Result', outputCol = 'predict')  
tmp +=[assembler\_features, labelIndexer]  
pipeline = Pipeline(stages=tmp)

In [74]:

allData = pipeline.fit(df).transform(df)  
allData.cache()  
trainingData, testData = allData.randomSplit([0.8,0.2], seed = 0)  
**print**("Distribution of HypoThyroid and Neg in trainingData is: ", trainingData.groupBy('predict').count().take(3))

('Distribution of HypoThyroid and Neg in trainingData is: ', [Row(predict=1.0, count=136), Row(predict=0.0, count=2386)])

In [75]:

rf = RF(labelCol='predict', featuresCol='tests', numTrees=50)  
fit = rf.fit(trainingData)  
transformed = fit.transform(testData)

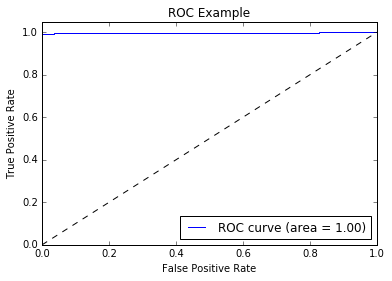
In [77]:

**from** **pyspark.mllib.evaluation** **import** BinaryClassificationMetrics **as** metric  
results = transformed.select(['probability','predict'])  
results\_collect = results.collect()  
results\_list = [(float(i[0][0]),1.0-float(i[1])) **for** i **in** results\_collect]  
scoreAndLabels = sc.parallelize(results\_list)  
  
metrics = metric(scoreAndLabels)  
**print**("The ROC score is (@numTrees=50):", metrics.areaUnderROC)

('The ROC score is (@numTrees=50):', 0.9974782084315551)

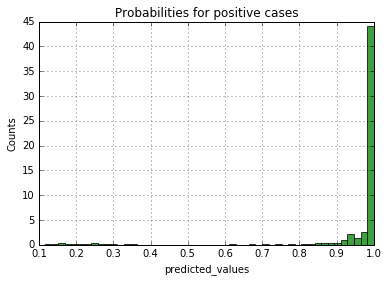
In [83]:

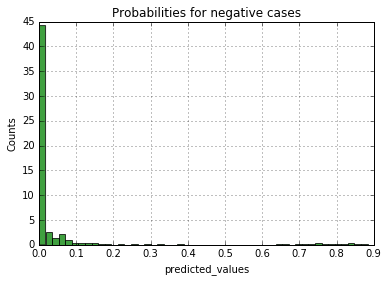
**from** **sklearn.metrics** **import** roc\_curve, auc  
  
fpr = dict()  
tpr = dict()  
roc\_auc = dict()  
  
y\_test = [i[1] **for** i **in** results\_list]  
y\_score = [i[0] **for** i **in** results\_list]  
  
fpr, tpr, \_ = roc\_curve(y\_test, y\_score)  
roc\_auc = auc(fpr, tpr)  
  
%**matplotlib** inline  
plt.figure()  
plt.plot(fpr, tpr, label = 'ROC curve (area = **%0.2f**)' % roc\_auc)  
plt.plot([0,1],[0,1], 'k--')  
plt.xlim([0.0,1.0])  
plt.ylim([0.0,1.05])  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('ROC Example')  
plt.legend(loc="lower right")  
plt.show()



In [85]:

all\_probs = transformed.select("probability").collect()  
pos\_probs = [i[0][0] **for** i **in** all\_probs]  
neg\_probs = [i[0][1] **for** i **in** all\_probs]  
  
**from** **matplotlib** **import** pyplot **as** plt  
%**matplotlib** inline  
  
*#pos*  
plt.hist(pos\_probs, 50, normed =1, facecolor ='green', alpha=0.75)  
plt.xlabel('predicted\_values')  
plt.ylabel('Counts')  
plt.title('Probabilities for positive cases')  
plt.grid(True)  
plt.show()  
  
*#neg*  
plt.hist(neg\_probs, 50, normed =1, facecolor ='green', alpha=0.75)  
plt.xlabel('predicted\_values')  
plt.ylabel('Counts')  
plt.title('Probabilities for negative cases')  
plt.grid(True)  
plt.show()





In [90]:

**from** **numpy.random** **import** randint  
**from** **pyspark.sql.functions** **import** udf  
**from** **pyspark.sql.types** **import** IntegerType  
  
RATIO\_ADJUST = 2.0  
  
counts = trainingData.select('Result').groupBy('Result').count().collect()  
higherBound = counts[0][1]  
THESHOLD\_TO\_FILTER = int(RATIO\_ADJUST \* float(counts[1][1]/ counts [0][1] \*higherBound))  
  
randGen = **lambda** x: randint(0, higherBound) **if** x == "hypothyroid" **else** -1  
  
udfRandGen = udf(randGen, IntegerType())  
trainingData = trainingData.withColumn("randIndex", udfRandGen("Result"))  
df\_subsample = trainingData.filter(trainingData['randIndex'] <THESHOLD\_TO\_FILTER)  
df\_subsample = df\_subsample.drop('randIndex')  
  
**print**("Distribution of Pos and Neg cases of the down-sample training data are: **\n**", df\_subsample.groupBy('Result').count().collect())

('Distribution of Pos and Neg cases of the down-sample training data are: \n', [Row(Result=u'hypothyroid', count=136), Row(Result=u'negative', count=2386)])