Below, there are some scripts for executing classification in PySpark. Download those scripts to your Spark environment (or cut and paste if you need to). The following quiz will ask a series of question using some observations with these scripts.

[NOTE: These scripts were created for Spark MLLIB 1.3.0. They should run OK on Cloudera VM, but I do get an occasional crash or error on my Cloudera VM that I don't get when I run on my cluster. Be careful with spacing in the python commands, and sometimes you might have to try rerunning.]

1. Review the scripts:

There are 3 scripts:

'doweathclass.py' will create Weather data for predicting if someone will 'Play' tennis. The data is hard coded as a list of lists, put into a dataframe and then mapped into an RDD of labeled point vectors. Notice that the categorical variables are recoded as binary indicator variable (eg outlook='sunny' 'overcast' or 'rainy' is replace by 3 variables sunny (1 or 0), overcast (1 or 0), rainy (1 or 0)).

doweathclass.py (https://d18ky98rnyall9.cloudfront.net/...

'doweathclass_naivebayes.py' will execute a Naive Bayes classifier on the RDD of labeled points. There is also some lines of code to get a confusion matrix.

doweathclass_naivebayes.py (https://d18ky98rnyall9.clo...

'doweathclass_dectree.py' will execute a Decision Tree classifier on the RDD, and produce a confusion matrix as well.

doweathclass_dectree.py (https://d18ky98rnyall9.cloudfr...

2. Execute Naive Bayes classification:

In your Cloudera VM open a terminal, and then from Unix prompt install numpy (if had not already)

```
> sudo easy_install numpy==1.4.1
```

And then enter:

```
> PYSPARK_DRIVER_PYTHON=ipython pyspark
```

from Spark prompt

```
>>> exec(open('doweathclass.py').read())
```

After lots of INFO messages, check what variables are there:

```
>>> dir()
```

Look for the datax_rdd:

```
>>> datax_rdd.take(5)
```

You can also see what rdd functions are available:

```
>>> dir(datax_rdd)
```

3. Execute the NaiveBayes script:

```
>>> exec(open('doweathclass_naivebayes.py').read())
```

You should see the confusion matrix printed out and the percent correct.

What is the approximate percent correct? (save the answer for the quiz)

Look at the confusion matrix and write down the numbers for the quiz.

4. In spark, run the decision tree script:

```
>>> exec(open('doweathclass_dectree.py').read())
```

Look at the confusion matrix that is output, save the numbers.

5. The decision tree function returns a decision tree object. (Review the code). Check out the object; enter just the object name:

```
>>> dt_model

>>> dir(dt_model)
```

Execute the following to see the decision tree:

```
>>>print dt_model.toDebugString()
```

The first line returned should tell you the number of nodes and the depth of the tree.

Observe how the number of nodes is related to the decision tree.

6. Let's try adding a useless variable. In the doweathclass.py script add a new variable that is constant, after the 'play' column, as follows (it is short enough to edit by hand)

From Unix prompt

```
> gedit doweathclass.py
```

```
rawdata=[['sunny',85,85,'FALSE',0,    1],
         ['sunny',80,90,'TRUE',0,     1],
         ['overcast',83,86,'FALSE',1, 1],
         ['rainy',70,96,'FALSE',1,    1],
...etc
```

Now to process this rawdata list I have to change the rest of code a bit as follows. First change the dataframe creation to include a 'mydummy' field:

```
data_df=sqlContext.createDataFrame(rawdata,
['outlook','temp','humid','windy','play','mydummy']) #<--add field
```

Next, change the function that creates labeled points. The function should have 1 more line of code (see the 'add this' marked line) that just includes the new column in the labeled point vector.

```
#make RDD of labeled vectors
def newrow(dfrow):
    outrow = list(out2index.get((dfrow[0])))  #get dictionary entry
    outrow.append(dfrow[1])   #temp
    outrow.append(dfrow[2])   #humidity
    if dfrow[3]=='TRUE':      #windy
        outrow.append(1)
    else:
        outrow.append(0)
    outrow.append(dfrow[5])  # <---- add this
    return (LabeledPoint(dfrow[4],outrow))
```

Now rerun the NaiveBayes and DecisionTree scripts, and observe the impact on the percent correct.

7. Lets add a new test point.

7.1. Create an numpy array with the following values:

[1,0,0,68,79,0,1]

```
>>> newpoint  = np.array([ ... ])
```

This for the outlook binary indicator variables, sunny,overcast,rainy,

the temperature = 68,

the humidy = 79,

windy = 0

useless-constant dummy variable = 1

7.2. Now run that test point through the NaiveBayes and DecisionTree model using the predict functions.

(sorry for the slight inconsistency in my model result naming conventions)

```
>>> my_nbmodel.predict( ....)

>>> dt_model.predict( ... )
```

7.3. You should see that the NaiveBayes and Decision Tree gave different answers. It is very difficult to say exactly why the answer are different, without recreating the entire algorithm calculations. However, we can do a quick analysis as follows;

Print the decision tree out and look at how the test point is labeled in that tree. For example, observe which variables are used to get the prediction node for this test point.

.