In [1]:
```
%%javascript
/*********************************************************************
**********************
Known Mathjax Issue with Chrome - a rounding issue adds a border to the
 right of mathjax markup
https://github.com/mathjax/MathJax/issues/1300
A quick hack to fix this based on stackoverflow discussions:
http://stackoverflow.com/questions/34277967/chrome-rendering-mathjax-equ
ations-with-a-trailing-vertical-line
*********************************************************************
**********************/

$('.math>span').css("border-left-color","transparent")
```

In [ ]:
```
%reload_ext autoreload
%autoreload 2
```

MIDS w261 - Machine Learning At Scale

Assignment - HW13

Name: Alex Smith
Class: Section 2, e.g., Summer 2016
Email: aksmith@ischool.berkely.edu
Week: 14

Table of Contents

1 Instructions

Back to Table of Contents

- Homework submissions are due by Tuesday, 08/23/2016 at 11AM (West Coast Time).
- Prepare a single Jupyter notebook (not a requirment), please include questions, and question numbers in the questions and in the responses. Submit your homework notebook via the following form:
  - Submission Link - Google Form (https://docs.google.com/forms/d/1ZOr9RnIe_A06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiis/viewform?usp=send_form)

Documents:

- IPython Notebook, published and viewable online.
- PDF export of IPython Notebook.

2 Useful References

Back to Table of Contents

- Lecture 12 (Async)
- Chapter 17 on decision Trees, https://www.dropbox.com/s/5ca98ah5chqlcmn/Data_Science_from_Scratch%20%281%29.pdf?dl=0 (https://www.dropbox.com/s/5ca98ah5chqlcmn/Data_Science_from_Scratch%20%281%29.pdf?dl=0) [Please do not share this PDF]
- Karau, Holden, Konwinski, Andy, Wendell, Patrick, & Zaharia, Matei. (2015). Learning Spark: Lightning-fast big data analysis. Sebastopol, CA: O'Reilly Publishers.
- Hastie, Trevor, Tibshirani, Robert, & Friedman, Jerome. (2009). The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Stanford, CA: Springer Science+Business Media. **(Download for free here (http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf))**
- Ryza, Sandy, Laserson, Uri, Owen, Sean, & Wills, Josh. (2015). Advanced analytics with Spark: Patterns for learning from data at scale. Sebastopol, CA: O'Reilly Publishers.

HW13.1 Build a decision to predict whether you can play tennis or not

Back to Table of Contents

Decision Trees

Write a program in Python (or in Spark; this part is optional) to implement the ID3 decision tree algorithm. You should build a tree to predict PlayTennis, based on the other attributes (but, do not use the Day attribute in your tree.). You should read in a space delimited dataset in a file called dataset.txt and output to the screen your decision tree and the training set accuracy in some readable format. For example, here is the tennis dataset. The first line will contain the names of the fields:

```
Day outlook temperature humidity wind playtennis
d1 sunny hot high FALSE no
d2 sunny hot high TRUE no
d3 overcast hot high FALSE yes
d4 rainy mild high FALSE yes
d5 rainy cool normal FALSE yes
d6 rainy cool normal TRUE no
d6 overcast cool normal TRUE yes
d7 sunny mild high FALSE no
d8 sunny cool normal FALSE yes
d9 rainy mild normal FALSE yes
d10 sunny mild normal TRUE yes
d11 overcast mild high TRUE yes
d12 overcast hot normal FALSE yes
d12 rainy mild high TRUE no
```

The last column is the classification attribute, and will always contain contain the values yes or no.

For output, you can choose how to draw the tree so long as it is clear what the tree is. You might find it easier if you turn the decision tree on its side, and use indentation to show levels of the tree as it grows from the left. For example:

```
outlook = sunny
|  humidity = high: no
|  humidity = normal: yes
outlook = overcast: yes
outlook = rainy
|  windy = TRUE: no
|  windy = FALSE: yes
```

You don't need to make your tree output look exactly like above: feel free to print out something similarly readable if you think it is easier to code.

You may find Python dictionaries especially useful here, as they will give you a quick an easy way to help manage counting the number of times you see a particular attribute.

Here are some FAQs that I've gotten in the past regarding this assignment, and some I might get if I don't answer them now.

Should my code work for other datasets besides the tennis dataset? Yes. We will give your program a different dataset to try it out with. You may assume that our dataset is correct and well-formatted, but you should not make assumptions regrading number of rows, number of columns, or values that will appear within. The last column will also be the classification, and will always contain yes or no values.

Is it possible that some value, like "normal," could appear in more than one column? Yes. In addition to the column "humidity", we might have had another column called "skycolor" which could have values "normal," "weird," and "bizarre."

Could "yes" and "no" appear as possible values in columns other than the classification column? Yes. In addition to the classification column "playtennis," we might have had another column called "seasonalweather" which would contain "yes" and "no."

Save the data to a file

In [2]:
```
%%writefile tennis.txt
Day outlook temperature humidity wind playtennis
d1 sunny hot high FALSE no
d2 sunny hot high TRUE no
d3 overcast hot high FALSE yes
d4 rainy mild high FALSE yes
d5 rainy cool normal FALSE yes
d6 rainy cool normal TRUE no
d6 overcast cool normal TRUE yes
d7 sunny mild high FALSE no
d8 sunny cool normal FALSE yes
d9 rainy mild normal FALSE yes
d10 sunny mild normal TRUE yes
d11 overcast mild high TRUE yes
d12 overcast hot normal FALSE yes
d12 rainy mild high TRUE no
```

```
Writing tennis.txt
```

Load in the data

In [2]:
```
def loadData(path2file):
    """takes a filepath and loads in the data. the
    first row of data MUST be the labels for the
    data. returns the dictionary of the data and a
    list of all the labels"""

    # create a list for storing the labels
    labels = []

    # create a dictionary for storing the
    # data for each record
    records = []

    # open the file
    with open(path2file,'r') as myfile:

        # read in the lines
        for index_i,line in enumerate(myfile.readlines()):

            # split the line by spaces
            line = line.strip('\n').split(' ')

            # if it's the first line, then
            # save the line to the list of labels
            if index_i == 0:
                labels = line[1:]

            # otherwise add a record to the dictionary
            else:

                # set up the label for that data
                label = line[-1]
                if label == 'yes':
                    label = True
                else:
                    label = False

                # gather the attributes
                attributes = line[1:-1]

                # create a new dictionary for the record
                record = {}

                # loop through the attributes
                # and add each attribute to the
                # dictionary for that day
                for index_j,attribute in enumerate(attributes):
                    record[labels[index_j]] = attribute

                # append the record with its label to the list
                info = record,label
                records.append(info)

    # return the labels and a dictionary of
    # all the data
    return labels[:-1],records
```

In [3]:
```
# load in the data
labels,data = loadData('tennis.txt')

# preview the data
for line in data:
    print line
```

```
({'outlook': 'sunny', 'temperature': 'hot', 'wind': 'FALSE', 'humidit
y': 'high'}, False)
({'outlook': 'sunny', 'temperature': 'hot', 'wind': 'TRUE', 'humidity':
 'high'}, False)
({'outlook': 'overcast', 'temperature': 'hot', 'wind': 'FALSE', 'humidi
ty': 'high'}, True)
({'outlook': 'rainy', 'temperature': 'mild', 'wind': 'FALSE', 'humidit
y': 'high'}, True)
({'outlook': 'rainy', 'temperature': 'cool', 'wind': 'FALSE', 'humidit
y': 'normal'}, True)
({'outlook': 'rainy', 'temperature': 'cool', 'wind': 'TRUE', 'humidit
y': 'normal'}, False)
({'outlook': 'overcast', 'temperature': 'cool', 'wind': 'TRUE', 'humidi
ty': 'normal'}, True)
({'outlook': 'sunny', 'temperature': 'mild', 'wind': 'FALSE', 'humidit
y': 'high'}, False)
({'outlook': 'sunny', 'temperature': 'cool', 'wind': 'FALSE', 'humidit
y': 'normal'}, True)
({'outlook': 'rainy', 'temperature': 'mild', 'wind': 'FALSE', 'humidit
y': 'normal'}, True)
({'outlook': 'sunny', 'temperature': 'mild', 'wind': 'TRUE', 'humidit
y': 'normal'}, True)
({'outlook': 'overcast', 'temperature': 'mild', 'wind': 'TRUE', 'humidi
ty': 'high'}, True)
({'outlook': 'overcast', 'temperature': 'hot', 'wind': 'FALSE', 'humidi
ty': 'normal'}, True)
({'outlook': 'rainy', 'temperature': 'mild', 'wind': 'TRUE', 'humidit
y': 'high'}, False)
```

Write a function to compute entropy among the different possible partitions

In [23]:
```
import math

def entropy(class_probabilities):
    """given a list of class probabilities, compute the entropy"""

    # create a value to hold entropy
    entrop_sum = 0.0

    # loop through each probability
    for prob in class_probabilities:

        # calculate the entropy for each
        # probability
        if prob:   # ignore zero probabilities
            entrop_part = -prob * math.log(prob,2)

            # calculate the sum of the entropies
            entrop_sum = entrop_sum + entrop_part

    return entrop_sum
```

In [54]:
```
from collections import Counter

def classProbabilities(labels):
    """takes as input a bunch of labels
    and returns the probabilities for
    each label"""

    # set the total labels as the length
    # of the label array
    total_count = len(labels)

    # create an array to store the
    # probabilities of each subset
    probs = []

    # covert the labels into a special
    # dictionary of type counter
    for count in Counter(labels).values():

        # take the partial probability
        # and append it to the main array
        prob_partial = float(count) / float(total_count)
        probs.append(prob_partial)

    return probs
```

In [55]:
```
def dataEntropy(labeled_data):

    # create an array to hold the labels
    labels = []

    # loop through the data to get the labels
    # and append them to our labels array
    for _,label in labeled_data:
        labels.append(label)

    # calculate the probabilities based on the
    # labels
    probabilities = classProbabilities(labels)

    # return the entropy of these probabilities
    return entropy(probabilities)
```

In [56]:
```
def partitionEntropy(subsets):
    """find the entropy from this partition of data
    into subsets. subsets is a list of lists of labeled data"""

    # calculate the total count as the sum of the
    # number of elements in each subset of subsets
    total_count = 0
    for subset in subsets:
        total_count = total_count + len(subset)

    # set a variable for the total entropy
    total_entropy = 0.0

    # loop through each subset
    for subset in subsets:

        # calculate the partial entropy
        partial_entropy = dataEntropy(subset) * len(subset) / total_coun
t

        # add to the total entropy
        total_entropy = total_entropy + partial_entropy

    return total_entropy
```

In [57]:
```
from collections import defaultdict

def partitionBy(data,label):
    """takes as input a list of attributes
    and labels and outputs the groups and
    their corresponding values"""

    # create a special type of dictionary,
    # a list dictionary that only adds unique values
    groups = defaultdict(list)

    # loop through input
    for inpt in data:
        key = inpt[0][label]
        groups[key].append(inpt)

    # return the groups
    return groups
```

In [62]:
```
# test out the partitionBy function
partitionBy(data,'outlook')
```

Out[62]:

```
defaultdict(list,
            {'overcast': [({'humidity': 'high',
                'outlook': 'overcast',
                'temperature': 'hot',
                'wind': 'FALSE'},
               True),
              ({'humidity': 'normal',
                'outlook': 'overcast',
                'temperature': 'cool',
                'wind': 'TRUE'},
               True),
              ({'humidity': 'high',
                'outlook': 'overcast',
                'temperature': 'mild',
                'wind': 'TRUE'},
               True),
              ({'humidity': 'normal',
                'outlook': 'overcast',
                'temperature': 'hot',
                'wind': 'FALSE'},
               True)],
             'rainy': [({'humidity': 'high',
                'outlook': 'rainy',
                'temperature': 'mild',
                'wind': 'FALSE'},
               True),
              ({'humidity': 'normal',
                'outlook': 'rainy',
                'temperature': 'cool',
                'wind': 'FALSE'},
               True),
              ({'humidity': 'normal',
                'outlook': 'rainy',
                'temperature': 'cool',
                'wind': 'TRUE'},
               False),
              ({'humidity': 'normal',
                'outlook': 'rainy',
                'temperature': 'mild',
                'wind': 'FALSE'},
               True),
              ({'humidity': 'high',
                'outlook': 'rainy',
                'temperature': 'mild',
                'wind': 'TRUE'},
               False)],
             'sunny': [({'humidity': 'high',
                'outlook': 'sunny',
                'temperature': 'hot',
                'wind': 'FALSE'},
               False),
              ({'humidity': 'high',
                'outlook': 'sunny',
                'temperature': 'hot',
                'wind': 'TRUE'},
               False),
              ({'humidity': 'high',
```

```
  'outlook': 'sunny',
  'temperature': 'mild',
  'wind': 'FALSE'},
 False),
({'humidity': 'normal',
  'outlook': 'sunny',
  'temperature': 'cool',
  'wind': 'FALSE'},
 True),
({'humidity': 'normal',
  'outlook': 'sunny',
  'temperature': 'mild',
  'wind': 'TRUE'},
 True)]})
```

In [58]:
```
def partitionEntropyBy(data, label):
    """computes the entropy corresponding to the given partition"""

    # gathers up the partions of the data
    partitions = partitionBy(data, label)

    return partitionEntropy(partitions.values())
```

In [59]:
```
# test out by printing the entropies for making each of many divisions
for label in labels:
    print label, partitionEntropyBy(data,label)
```

```
outlook 0.693536138896
temperature 0.911063393012
humidity 0.788450457308
wind 0.892158928262
```

Build the tree

In [66]:

```
from functools import partial

def buildTreeID3(inputs, split_candidates=None):
    """builds an decision tree based on the ID3 algorithm,
    takes as input, some data and potential split_candidates
    where the split candidates are none in the initialization"""

    # if this is our first pass,
    # all keys of the first input are split candidates
    if split_candidates is None:
        split_candidates = inputs[0][0].keys()

    # count True's and False's in the inputs
    # as the total number of inputs minutes
    # the number of true labels
    num_inputs = len(inputs)
    num_trues = 0
    for item,label in inputs:
        if label==True:
            num_trues = num_trues + 1
    num_falses = num_inputs - num_trues

    # if there are no trues present, return
    # that this leaf node should be false
    if num_trues == 0:
        return False

    # if there are no falses present, return
    # that this leaf node should be true
    if num_falses == 0:
        return True

    # if we don't have any split candidates left,
    # then return the majority class for the leaf
    if not split_candidates:
        return num_trues >= num_falses

    # otherwise, split on the best attribute
    # we take the minimum of the entropies for
    # each potential split. the partial function
    # helps us out by running the partitionEntropyBy
    # function on the inputs and the split_candidates
    best_attribute = min(split_candidates,
                         key=partial(partitionEntropyBy, inputs))

    # generate the partitions for the best attribute split
    partitions = partitionBy(inputs, best_attribute)

    # gather the new candidates for the subtrees
    # create a blank array to hold the new candidates
    new_candidates = []

    # loop through each candidate
    for attribute in split_candidates:

        # provided it's not the attribute on which we've already split
        if attribute != best_attribute:
```

```
            new_candidates.append(attribute)

    # create an empty dictionary to hold the subtrees
    subtrees = {}

    # loop through each of the partitions
    for attribute_value, subset in partitions.iteritems():

        # build a subtree and add it
        subtrees[attribute_value] = buildTreeID3(subset,new_candidates)

    # return the default case, if we don't have a
    # particular instance of an attribute in our
    # tree, then return the majority case
    subtrees[None] = num_trues > num_falses

    # return the tree
    return (best_attribute, subtrees)
```

In [89]:
```
# build the tree
mytree = buildTreeID3(data)
buildTreeID3(data)
```

Out[89]:
```
('outlook',
 {None: True,
  'overcast': True,
  'rainy': ('wind', {None: True, 'FALSE': True, 'TRUE': False}),
  'sunny': ('humidity', {None: False, 'high': False, 'normal': True})})
```

Develop a classification for the tree

In [90]:
```
def classify(tree, inpt):
    """classify the input using the given decision tree"""

    # if this is a leaf node, return its value
    if tree in [True, False]:
        return tree

    # otherwise this tree consists of an attribute to split on
    # and a dictionary whose keys are values of that attribute
    # and whose values of are subtrees to consider next

    # grab the first attribute and the
    # tree that we start at
    attribute, subtree_dict = tree

    # find the value for this attribute
    # among the data for our first input
    subtree_key = inpt.get(attribute)

    # if the subtree_key is not in the labels
    # already seen by the tree, then we'll use
    # the None key which returns the majority case
    if subtree_key not in subtree_dict:
        subtree_key = None

    # we then dig further, into the next
    # level of tree by grabbing the subtree
    # for that division
    subtree = subtree_dict[subtree_key]

    # we'll recursively keep digging until we've
    # gotten to a leaf node
    return classify(subtree, inpt)
```

In [100]:
```
# test the classifier
print "I play tennis if it's sunny..."
classify(mytree,{'outlook':'sunny'})
```

```
I play tennis if it's sunny...
```

Out[100]:
```
False
```

HW13.1.1 What is the classification accuracy of the tree on the training data?

In [103]:
```
# create counters to measure accuracy
total = 0
correct = 0

# loop through each of the training examples
for example,label in data:

    # classify the example
    predict = classify(mytree,example)

    # increment the counters
    total = total + 1
    if predict == label:
        correct = correct + 1

# generate the accuracy
print "Model Accuracy:",float(correct)/float(total)*100,"%"
```

```
Model Accuracy: 100.0 %
```

The model is eerily 100% accurate on the training data. This implies that we've overfit the training data.

HW13.1.2 Is it possible to produce some set of correct training examples that will get the algorihtm to include the attribute Temperature in the learned tree, even though the true target concept is independent of Temperature? if no, explain. If yes, give such a set.

Yes, it is possible to produce some set of correct training examples that include the attribute *temperature* in the learned tree. Even though temperature is independent of the target value in reality, it could happen by chance, that high and low temperatures during some week perfectly correlate with playing or not playing tennis. I have provided one such example below where playing tennis always paired with mild days and not playing tennis is always paired with hot days:

```
({'outlook': 'sunny', 'temperature': 'hot', 'wind': 'FALSE', 'humidity': 'hi
gh'}, False)
({'outlook': 'sunny', 'temperature': 'hot', 'wind': 'TRUE', 'humidity': 'hig
h'}, False)
({'outlook': 'overcast', 'temperature': 'mild', 'wind': 'FALSE', 'humidity':
 'high'}, True)
({'outlook': 'rainy', 'temperature': 'mild', 'wind': 'FALSE', 'humidity': 'h
igh'}, True)
({'outlook': 'rainy', 'temperature': 'mild', 'wind': 'FALSE', 'humidity': 'n
ormal'}, True)
({'outlook': 'rainy', 'temperature': 'hot', 'wind': 'TRUE', 'humidity': 'nor
mal'}, False)
({'outlook': 'overcast', 'temperature': 'mild', 'wind': 'TRUE', 'humidity':
 'normal'}, True)
({'outlook': 'sunny', 'temperature': 'hot', 'wind': 'FALSE', 'humidity': 'hi
gh'}, False)
({'outlook': 'sunny', 'temperature': 'mild', 'wind': 'FALSE', 'humidity': 'n
ormal'}, True)
({'outlook': 'rainy', 'temperature': 'mild', 'wind': 'FALSE', 'humidity': 'n
ormal'}, True)
({'outlook': 'sunny', 'temperature': 'mild', 'wind': 'TRUE', 'humidity': 'no
rmal'}, True)
({'outlook': 'overcast', 'temperature': 'mild', 'wind': 'TRUE', 'humidity':
 'high'}, True)
({'outlook': 'overcast', 'temperature': 'mild', 'wind': 'FALSE', 'humidity':
 'normal'}, True)
({'outlook': 'rainy', 'temperature': 'hot', 'wind': 'TRUE', 'humidity': 'hig
h'}, False)
```

HW13.1.3 Now, build a tree using only examples D1–D7. What is the classification accuracy for the training set? what is the accuracy for the test set (examples D8–D14)? explain why you think these are the results.

In [106]:
```
# separate out the data
dataPart1 = data[0:7]
dataPart2 = data[7:]

# build the tree
partD1D7 = buildTreeID3(dataPart1)
partD1D7
```

Out[106]:
```
('outlook',
 {None: True,
  'overcast': True,
  'rainy': ('wind', {None: True, 'FALSE': True, 'TRUE': False}),
  'sunny': False})
```

In [110]:
```
# use the tree to predict examples D8-D14
# create counters to measure accuracy
total = 0
correct = 0

# loop through each of the training examples
for example,label in dataPart1:

    # classify the example
    predict = classify(partD1D7,example)

    # increment the counters
    total = total + 1
    if predict == label:
        correct = correct + 1

# generate the accuracy
print "Model Accuracy on training
set:",float(correct)/float(total)*100,"%"
```

```
Model Accuracy on training set: 100.0 %
```

In [111]:
```
# use the tree to predict examples D8-D14
# create counters to measure accuracy
total = 0
correct = 0

# loop through each of the training examples
for example,label in dataPart2:

    # classify the example
    predict = classify(partD1D7,example)

    # increment the counters
    total = total + 1
    if predict == label:
        correct = correct + 1

# generate the accuracy
print "Model Accuracy on test set:",float(correct)/float(total)*100,"%"
```

```
Model Accuracy on test set: 71.4285714286 %
```

My accuracy falls to a mere 70%. I have clearly overfit the few exmaples in the training data (where I had an accuracy of 100%). This is because I extend each leaf node, till I'm 100% certain of the predicted class.

HW13.1.4 In this case, and others, there are only a few labelled examples available for training (that is, no additional data is available for testing or validation). Suggest a concrete pruning strategy, that can be readily embedded in the algorithm, to avoid over fitting. Explain why you think this strategy should work.

We are overfitting the training data with our model. There are a couple of a quick solutions that would help us not overfit the training data so badly. (1) We could end the subtree construction once we reach a certain number of examples in the node. For example, once we have 3 examples for a given node, we could stop building any further nodes and just take the majority class of that node with the mixed exmaples. (2) We could end the subtree construction once all the entropies fall below a certain value. Rather than continually taking the minimum entropy, once all entropies fall below a certain value, we decide that there is little value to keep splitting and simply take the majority class.

HW13.2 Regression Tree (OPTIONAL Homework)

Back to Table of Contents

Implement a decision tree algorithm for regression for two input continous variables and one categorical input variable on a single core computer using Python.

- Use the IRIS dataset to evaluate your code, where the input variables are: Petal.Length Petal.Width Species and the target or output variable is Sepal.Length.
- Use the same dataset to train and test your implementation.
- Stop expanding nodes once you have less than ten (10) examples (along with the usual stopping criteria).
- Report the mean squared error for your implementation and contrast that with the MSE from scikit-learn's implementation on this dataset (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html))

HW13.3 Predict survival on the Titanic using Python (Logistic regression, SVMs, Random Forests)

Back to Table of Contents

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this challenge, you need to review (and edit the code) in this notebook (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/kmbgrkhh73931lo/Titanic-EDA-LogisticRegression.ipynb) to do analysis of what sorts of people were likely to survive. In particular, please look at how the tools of machine learning are used to predict which passengers survived the tragedy. Please share any useful graphs/analysis you come up with via the group email. For example, pick the top two most important variables and plot the separating hyperplane in this 2D space that is generated using an SVM (or logistic regression model or plot both; are they similar?) that is learnt using those two features only. Comment on your observations. Please feel free to come up other graphs/analysis (e.g., clustering the passengers).

For more details see:

- https://www.kaggle.com/c/titanic (https://www.kaggle.com/c/titanic)

HW13.4 Heritage Healthcare Prize (Predict # Days in Hospital next year)

Back to Table of Contents

1. Introduction Back to Table of Contents

The Heritage Health Prize (HHP) was a data science challenge sponsored by The Heritage Provider Network. It took place from April 4, 2011 to April 4, 2013. For information on the winning entries, please see here.

Please see the following notebooks for more background and candidate solutions

- Spark Map-Reduce + MMLlib solution (with optional extensions) See Notebook (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/v52cxipe7yftf97/HeritageHealthPrizeUnitTestNotebook_Sp Map-Reduce.ipynb)
- Spark SQL + MLLib solution (with optional extensions): Notebook (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/s2wxg6g982oho5m/HeritageHealthPrizeUnitTestNotebook

Please look at section 7 in both notebooks complete any one or more the suggested next steps. Included here are excerpts from section 7 in both of those notebooks. E.g.,

- Please complete the EDA extensions using inspiration from the Titanic Notebook from above.
- **Complete Section 3.B: EDA-0. Gather information to see what transformations may need to be done on the data.** Answer questions about each raw DataFrame. In general, is the data in good shape? For example, in each of the Target DataFrames (df_target_Y1, df_target_Y2, df_target_Y3), what values does DaysInHospital take on? Are they all integers? What values does ClaimsTruncated take on? Are they all integers? In the Claims DataFrame (df_claims), how many different ProviderIDs are there? How many different PrimaryConditionGroups are there? What are their values? What values can the CharlesonIndex take on? Are they integers? In the Drug Count DataFrame (df_drug_count), what values can DrugCount take on? Are they all integers? Given this information, what transformations are needed?
- **Complete Section 3.D: EDA-1. Create tables and graphs to display information about the transformed DataFrames.** For inspiration, see the Titanic notebook discussed above. Answer questions about each DataFrame. For example, in each of the Target DataFrames (df_target_Y1, df_target_Y2, df_target_Y3), what is the minimum, maximum, mean, and standard deviation of DaysInHospital? In the Claims DataFrame, group by MemberID and Year and count the number of records. What is the minimum, maximum, mean, and standard deviation of the count? Do the same for the Drug Count and Lab Count DataFrames, etc.

- **Please generate ensemble of DT model using 100 trees with 8 nodes and report the Loss** Try additional models. See possibilities here (e.g. Decision Tree Regressor, Gradient-Boosted Trees Regressor, Random Forest Regressor). See an example here. Tune their hyperparameters. Try different feature selections. Try a two-step model.

In [ ]:

In [ ]: