Homework 5

[ 100 points - due by 11:59 pm, Sunday, March 5, 2017 ]

Submit these files to the CS submission system at the usual place by 11:59. You may work on your own or with 1-2 partners on the programming portions of this assignment. (The reading/response is individual only.) Groups larger than 3, please split into smaller groups! Remember that partners need to work in the same physical location, share composition time equally (or each compose on their own machines) and be fully equal owners and producers of their work. *Have fun treeing (and foresting)!*  [cs35 homepage](https://www.cs.hmc.edu/~dodds/cs35/)

**Downloads**

There's one (zipped) starter file to download -- grab it at the start of class & follow along:

* [The zip file, hw5.zip, to start all of this week's problems…](https://drive.google.com/open?id=0BwPWh-3AmiLxbVVNSmdaeHJtWG8) (now updated!)

Note for example starter file iris5.py:

* Line 179 and 180: dtree should be rforest
* ImputeLearn code giving TypeError (probably just delete this from starter code, sort of confusing)

**Submission**

**Overview** Again we ask you to submit a zipped archive named **hw5.zip** and we've standardized the filenames:

**digits5.py** [**lab** problem] askin you to use decision trees and random forests to model the digits

dataset, digits5.csv (a slightly altered version of last week's data) … In particular, building from the in-class iris.py example (provided), you should find (a) the values of max\_depth (for trees and random forests) and the value of n\_estimators (for random forests) via cross-validation. Provide one decision-tree image and an image of the importance of the various pixels (feature importance). **Extra**: draw some of your own digits, reduce them to 8x8 and see whether they're correctly classified!

**titanic5.py** This asks you to extend the Titanic-dataset analysis using decision trees and random forests.

For this problem, you should experiment with different ways of *imputing* age data for passengers whose outcomes are known (but whose ages are not). Compare mean-filling, median-filling, kNN,and RF as methods to impute data. Then, as with the digits dataset, use cross-validation to find reasonable parameters for a single DT and for RF. Include both a decision-tree image and the feature importances of your columns from RF.

**owndata5.py** Here, you should choose a dataset that interests you and that is suitable for predictive modeling.

Run (at least) three different experiments that predict one feature based on the other features available, using a decision tree, a random forest, and a neural network. Again, share at least one decision tree image for each of your two experiments and share the RF feature importances for each of your two experiments.

If you're not sure what data to use, consider the ["old-movies MovieLens" data](http://www.gregreda.com/2013/10/26/using-pandas-on-the-movielens-dataset/), add your own scores to it, and let us know what your model predicts you'll like -- and how much… . (**Extra**: use kNN and/or imputation, too, if you'd like/as appropriate).

As usual, submit your reading response in its own spot at the [submission site](http://cicero.cs.hmc.edu/).

As always, extra-credit is available for posting code and a write-up of any one of these problems to your GitHub repository (be sure to let us know you've done this -- and provide a direct link)

**Problem 0**: Machine learning and social media? *Cambridge Analytica* [5 pts]

This week's reading offers warnings/counterpoints to last week's ML optimism.

Read this [NYTimes article](https://www.nytimes.com/2016/11/20/opinion/the-secret-agenda-of-a-facebook-quiz.html?_r=0) on the application of machine-learning to social-media data and, in particular, the possible influence that [Cambridge Analytica](https://cambridgeanalytica.org/) had on 2016's presidential elections and Brexit votes. Some believe that the data analysis and targeting was [decisive](https://motherboard.vice.com/en_us/article/how-our-likes-helped-trump-win). Others, [not so much](https://www.bloomberg.com/view/articles/2016-12-08/no-big-data-didn-t-win-the-u-s-election). Then, consider the quote near the end of this [source research](http://www.pnas.org/content/110/15/5802.full):

*"the predictability of individual attributes from digital records of behavior may have considerable negative implications, because it can easily be applied to large numbers of people without obtaining their individual consent and without them noticing. Commercial companies, governmental institutions, or even one’s Facebook friends could use software to infer attributes such as intelligence, sexual orientation, or political views that an individual may not have intended to share."*

Do you agree with these potential negative implications? In your opinion, what *balance of responsibility* -- among individuals, social media sites (e..g, Facebook), and ML practitioners -- do you feel would best suit the situation? As with each week's reading, responses should carefully considered, but need not be very long: a 4-5 sentence paragraph is wonderful.

**[Lab problem] Problem 1: DTs/RFs for handwritten-digits data**

[30 pts; EC available for classifying *your own* handwritten digits...]

* This problem asks you to run/write your code in the **digits5.py** file.
* There aren't any unknowns this time -- unless you try the EC with your own digits
* This problem works with the digits5.csv dataset, similar to last week's digits.csv (but without any unknowns).
* [**Overall goals/tasks**] You'll notice that the digits5.py file has less starter code than iris5.py (our in-class examples). As with last week, you should use the iris5.py example to create both a DT and an RF model for the digits dataset -- and then share some of the insights those models provide. In more detail:
  + Remember that there are 64 input columns (X) and one output column (y) in the digits5.csv dataset. This week, all of the correct labels are provided.
  + First, build a DT model of the dataset and create an image of a model decision tree with a max\_depth of **4**. Include a screenshot as digitsDT.png (or any type)
  + Then, just as with last week, add loops to the cross-validation step so that the scripts is finding the average of at least 10 different cross-validation runs for each set of model parameters. Be sure to use the score for the *\_testing\_* portion of those cv runs (for both the DT and RF modeling).
  + Then, again similar to last week, find the model parameters that produce the best cross-validation results:
    - Find **max\_depth** for the DT model
    - Find **max\_depth** and **n\_estimators** for the RF model
    - Be strategic about the values you check -- you *won't* want to check n\_estimators from 1 to 200 striding by 1 each time! But, you *will* probably want to stride by 1 in your max\_depth search.
  + When you've found a reasonable choice for "best" parameters, create one DT and one RF model for all of the data. Include in a comment how well your best DT and RF did at the predictions for the digits (was it as good as kNN last time?)
  + **Include** the values (pasted into the bottom of your file) of the feature\_importances\_ from both your DT and from the RF. The feature\_importances\_ are, themselves, an image! (though you'll just see the floating-point pixel values…)
* ***EC Option***: For up to +10 points extra-credit, use your mouse and a drawing program OR take a photo of ten of your own hand-drawn digits. Then, run and report how well your DT/RF models classify your own handwriting! To get an image named digit.png resized into 8x8 pixels and then extract the 64 values (from 0 to 15), use this bit of code:
  + **Code!** Save the code below as **three.py**. Here's **three.png**: three.png

**#**

**# three.py**

**#**

**from PIL import Image**

**im = Image.open( "three.png" )**

**print("im is of size", im.size)**

**im\_resized = im.resize( (8,8), Image.BICUBIC )**

**ir = im\_resized**

**for row in range(8):**

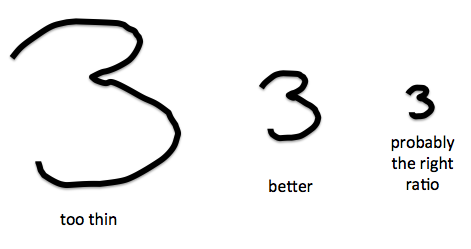
**for col in range(8):**

**r, g, b, a = ir.getpixel( (col,row) )**

**pixstring = "{0:3d} ".format(r) # cool Python formatting!**

**print( pixstring, end="")**

**print()**

* + As you draw your own, do be sure to draw thick enough lines (and/or small enough squares) in order to preserve the digit data all the way down to 8x8!
  + Image guide to thicknesses of digits: 

**Problem 2: DTs/RFs (and imputing) for Titanic data**

[30 pts; EC for comparing results between imputed vs. non-imputed source data]

* This problem asks you to run/write your code in the **titanic5.py** file.
* In addition, you'll include your model's estimates for the 20 missing-but-known ages (imputed)
* This problem asks you to model - and impute - a slightly different version of the titanic dataset, in titanic5.py. You may want to use your titanic.py file from last week to help.
* The goal here is to take advantage of DTs' and RFs' *modeling* capabilities -- built from the digits5 example (and from iris5), you should again
  + First, drop columns which don't have meaning relative to survival -- or which have too few data elements to be worthwhile
  + Then, use df. dropna() to get rid of rows with missing data
  + Following the previous problem's example, create a DT and RF models (for the target survival variable) with parameters found using 10x cross-validation. Include in your file (in the triple-quoted string at the bottom)
    - The average cross-validated test-set accuracy for your best DT model
    - The average cross-validated test-set accuracy for your best RF model
      * RF usually can get above 80% on the titanic data -- though *not* usually to 85%
    - A screenshot/image of the best DT (include this as titanicDT.png) In fact, this won't be in the triple-quoted string, but…
    - *do* include, in your comment, a brief summary of what the first two layers of the DT "ask" about a line of data...
    - Also, do include the feature\_importances\_ from your RF model
  + Next, *impute* the missing ages within the Titanic dataset. In order to do this, you'll need to redefine the target ('age') and features (include 'survived'). Also, since this is a floating-point value, not a category, you will want to use a RandomForestRegressor (instead of RandomForestClassifier). Make sure to remove dropna() so that the rows are not deleted.
    - Note: if you use the imputing code in the iris5.py file, you will need to be sure that all of the data is *numeric.* The line

#df['irisname'] = df['irisname'].map(transform) at the top of iris5.py will do this, when commented "back in"...

* + For the top 30 rows, the actual ages are known -- but were not provided. Include your 30 imputed ages using RF-based imputation. (How well did it do?)
  + [Here is a Google sheet with the full rows of data](https://docs.google.com/spreadsheets/d/1aB68MT_PK0VALjQl-660EE7AbqhwbRzFK5tBpYz-TUA/edit#gid=0) (including ages) for the first thirty rows. (In fact a couple of the ages *weren't* known, but that's totally ok.)

***EC Option***: For up to +2 points extra-credit, try imputing the ages with kNN (again, use a Regressor!), too -- which technique was more accurate? re-run your DT and RF analyses on the inputed data (with the new ages) -- and compare these with the original models: Which did better? By how much?

**Problem 3: DTs/RFs/NNs for your own dataset**

[35 pts; EC for anything above and beyond!]

* This problem asks you to run/write your code in the **owndata.py** file.
* For this problem, choose your *own dataset* to analyze/model with the DT, RF & NN. Don't include the data if it's huge (please do include at least a small subset, however, with which we can run your **owndata.py** file!)
* Make sure you select a dataset that interests you -- and that is suitable for predictive modeling: there needs to be something that can be predicted from other features. (However, it need *not* be causal -- correlations are all that ML *really* deals with!)
* Run (at least) three different experiments -- in precisely the styles above. In each one, predict a different feature based on the other features available, using a decision tree DT, a random forest RF, and a neural network NN.
* For DT and RF, share at least one decision tree image for each of your two experiments and share the RF feature importances (paste them in the comments at the bottom) for each of your two experiments. Please name your decision-tree images owndata1.png and owndata2.png (though with any image type)
* For NN, share the best parameters of your MLP in comments at the bottom
* Be sure to give a brief overview (3-4 sentences) of your dataset choice, what you did with it, and any insights (obvious or non-obvious) that emerged.
* If you're not sure what data to use, you may want to revisit our [page of dataset pages](https://docs.google.com/document/d/1dr2_Byi4I6KI7CQUTiMjX0FXRo-M9k6kB2OESd7a2ck/edit). Or, consider the ["old-movies MovieLens" data](http://www.gregreda.com/2013/10/26/using-pandas-on-the-movielens-dataset/), and add your own scores to it. From there, you can see what movies your model predicts *you* will like -- and how much you'll like it! (Then, tell us whether it's true, as well!)

**Extra**: For up to +5 points extra-credit, create a comparison of models with kNN and/or use imputation or other prediction approaches - whatever's appropriate for your dataset. Let us know if you take one of these routes in your overall comments!

**Extra-credit: Deep Learning with TensorFlow**

[??? pts]

TensorFlow is an open-source software library for Machine Intelligence. It was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well. This extra credit assignment will introduce you to some of TensorFlow's basic deep learning capabilities.

To install TensorFlow, type the following in the terminal:

MAC

* conda create -n tensorflow python=3.5
* source activate tensorflow
* conda install -c conda-forge tensorflow
* pip install --ignore-installed --upgrade $TF\_BINARY\_URL
* conda install ipython

To change prompt back:

* source deactivate

WINDOWS

* conda create -n tensorflow python=3.5
* activate tensorflow
* pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.2.1-cp35-cp35m-win\_amd64.whl
* conda install ipython

To change prompt back:

* deactivate

Take a look at the following tutorials to get a feel for how TensorFlow works. The first video just sets up the neural network, and converting what you already have learned about NNs into TensorFlow syntax (you can skip most of his talking in this video if you have a good understanding of NNs already). The second video shows how to use TensorFlow to train your model based on the data.

**To do:**

Follow along with the videos and submit the code in **tf\_tutorial.py** that trains a NN based on the minst data. The minst dataset is a database of handwritten digits that has a training set of 60,000 examples, and a test set of 10,000 examples. The particulars of the data are further explained at the beginning of the first video.

As you follow along (or after you have completed the code), add comments throughout explaining what each section of code is doing. (Note that the "Batch" training is basically the cross validation you have seen using SciKit.)

<https://www.youtube.com/watch?v=BhpvH5DuVu8>

<https://www.youtube.com/watch?v=PwAGxqrXSCs>

The link below is a written tutorial using the same minst dataset. The lines of code within are very similar to those in the video. It contains some helpful explanations and descriptions if you are confused by parts of the video!

<https://www.tensorflow.org/get_started/mnist/beginners>

He does make a few errors along the way, but he fixes them towards the end. Here they are so that you don't have to watch him debug his code.

* tf.initialize\_all\_variables() -> tf.global\_variables\_initializer() (Deprecated method)
* hm\_epochs -> range(hm\_epochs)
* '+' -> ',' when defining l1-l3 because you already added with tf.add()
* n\_nodes\_hl<1-3> -> [n\_nodes\_hl<1-3>] when setting 'biases' in hidden\_1-3\_layer because random\_normal takes in an array

Feel free to check out parts 5 and beyond from these sentdex tutorials to learn more about using TensorFlow!

**Extra-credit: Showing off your results…**

[up to +5 pts extra-credit...]

* As with each week, you're invited to include both your source code and a short write-up of one of the week's problems within your GitHub repo(s). Images and other visuals, of course, are welcome. If you do this, let us know (and provide a direct link :-)