Homework 4

[ 100 points - due by 11:59 pm, Sunday, February 26, 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 neighboring!*  [cs35 homepage](https://www.cs.hmc.edu/~dodds/cs35/)

**Downloads**

Starter files to download -- grab them at the start of class & follow along:

* [The zip file, hw4.zip, to start all of this week's problems…](https://drive.google.com/open?id=0BwPWh-3AmiLxM3JrMDNqVWZ4ZEk)
* [Boston](https://github.com/ScriptingBeyondCS/CS-35/tree/master/week_4)

cross-validation deprecated in favor of model\_selection module

Change appropriate lines to:

* from sklearn.model\_selection import train\_test\_split
* cv\_data\_train, cv\_data\_test, cv\_target\_train, cv\_target\_test = \

train\_test\_split(train, train\_labels, test\_size=0.25)

**Submission**

Again we ask you to submit a zipped archive named hw4.zip -- again, we've standardize the filenames -- this week we ask for these three files:

**iris.py** [**lab** problem] which introduces pandas, nearest neighbors, and cross-validation (using iris.csv)

include your answers to the 9 "mystery irises" inside your hw4pr1.py file, at the bottom...

**digits.py** kNN with the "digits" dataset -- include your "mystery digits" answers at the bottom of hw4pr2.py

Extra credit: use the nearest neighbors to "fill in" the missing pixel data… :-)

**titanic.py** kNN with the "titanic" dataset -- again, include your "mystery passenger" answers at the bottom

Extra: visualize some of the titanic data using matplotlib

Have your own dataset of interest? Feel free to use it -- this will be the final problem of \_next week's\_ hw, as well.

**boston.py** Use neural networks (multilayer perceptrons) to predict the median value of houses in Boston based on a number of factors.

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**: AI and ML today: *The Great AI Awakening* [5 pts]

This week's reading is a recent [NYTimes article](https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html?_r=0) on the "Great AI Awakening" It's a long article, so please feel free to stop after the prologue (just before Part I: Learning Machine - of course, if you're like me, you may find yourself reading further regardless... :-) The article gives context of some of the recent developments in AI, and especially machine-learning (statistical-learning). After reading over at least the first part of that NYTimes article, consider the quote near the end of the prologue:

*"Once a machine can translate fluently between two natural languages, the foundation has been laid for a machine that*

*that might one day "understand" human language well enough to engage in a plausible conversation."*

What do you think of this premise? Would a capable translation machine be able to converse naturally? Or, is it, perhaps, an important prerequisite to a natural "personal assistant." Or does your intuition tell you that translation and conversation might turn out to be relatively unrelated? 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: kNN for Iris data**

[30 pts]

* This problem asks you to run/write your code in the **iris.py** file.
* In addition, you'll submit your model's answers to the 9 unknown flowers at the bottom of your file
* This week's problems introduce two more libraries in Python: the Pandas library for data-handling (it makes some tasks easier) and the scikit-learnlibrary for machine learning, which implements many (most?) machine learning algorithms. You'll be using the nearest-neighbors algorithm this week. It's one of the most fundamental approaches to machine learning.
* The data file for this problem is iris.csv, which is in the starter files folder.
* [**Overall goals/tasks**] Your overall tasks for this problem are to
  + Get more comfortable with pandas, numpy, scikit-learn, and kNN. Out of the box, the iris.py file should run, train a kNN model, and classify nine flowers (nine it already knows). Read over the code, follow its steps and print statement, and experiment!
  + Write a loop that changes the single cross-validation step into 10-fold cross validation (and reports the \_average\_ success score for both training and testing cross-validation through those 10 iterations)
    - **Hint**: this would loop the lines 92 to 102 ten times, accumulating the testing-data scores each time (and then, at the end, dividing by 10)
  + Write another loop that iterates through the **n\_neighbors** model parameter, also called *k*, in order to determine which one performs best (based on its cross-validation scores)
    - **Hint**: this would loop around the code from lines 89 to 102 (which includes the previously-looped code!) -- this time, changing the number of neighbors each time and finding the number that maximizes the average testing-data scores. A common approach is to try a variety of odd values for the number of neighbors, e.g., with a loop like this one: for k in [1,3,5,7,9,11,15,21,32,42,51,71,91]
  + Once you've found a "good" k value (the "best" may change from run to run because of the randomness involved), comment out the cross-validation (the nested loops) and create a single model with the "good" value of k (the number of neighbors)
  + Also, change the data-handling at the top of the file so that your tuned model (with your chosen k), uses *all* of the labeled data in its training and then predicts the classifications of the 9 unknown irises at the top of the file (they're excluded by the starter code). Then, paste those 9 identifications into the bottom of your file, along with a short comment about how this lab went overall.

*Note: We have also included the file knn.py with the starter files. This is the knn algorithm written from scratch, without any methods from the sklearn library. If you do not understand what the sklearn methods are actually doing, it may be helpful to read through the knn.py code. (Though it does the same thing, knn.py is much less efficient than sklearn methods and can take up to a few minutes to run, depending on the size of the data set.)*

* The next two problems will ask you to go further with kNN! Remember that, if you join one of the labs, the lab incentive applies: credit for this problem, even if you don't make it to the end of all of the above challenges…

**Problem 2: kNN for handwritten-digits data**

[30 pts; with "image fill" EC available!]

* This problem asks you to run/write your code in the **digits.py** file.
* In addition, you'll submit your model's answers to the 22 unknown digits at the bottom of your file
* This problem asks you to use a larger dataset, the hand-written digits in digits.csv. Each digit is 8 pixels by 8 pixels, with grayscale values from 0 to 16. That file also includes 22 "unknown" digits. The columns indexed from 0 to 63 have the pixel data (rowwise). Column index 64 has the label, which is 0-9 or "-1" for the "unknown" ditits.
  + Also, the first 10 have the bottom three rows of pixels "erased"
* [**Overall goals/tasks**] You'll notice that the digits.py file has less starter code than iris.py -- you should use the example (copying and editing as you go…) from iris.py into digits.py to create a kNN model for the digits dataset, tune it for k, and predict the labels of the unknown digits. In more detail:
  + First, run the digits.py code as provided. Run several times the plotting function that allows you to show any digits as an 8x8 image by changing the "row" value. By looking at the unknown images, you'll be able to recognize what (most) of them are - the partially erased ones are sometimes tougher.
  + One piece at a time, write (or paste/alter) code from the iris.py example so that you can create a knn model for the digits data.
    - There will be 64 input columns (X) and one output column (y)
  + Again, use at least 10-fold cross validation (with the \_average\_ success score for those 10 iterations) to determine a k (**n\_neighbors**) that works well.
  + Also, feel free to use cross-validation to help you *reweight* columns (a form of feature engineering). For example, you might multiply columns whose pixels aren't important by a fraction less than 1.0; you might multiply those whose pixels you feel are more likely to be important by a coefficient greater than 1.0. Here, the idea is to use your intuition -- and the results of the cross-validation.
    - **Float Note!** In order to multiply a column by a fraction or a float, e.g., 0.1, you'll need to convert the data to floats (they'll be ints by default). There are many ways to do this, but the easiest is to make just one pixel in the csv file a float (add a .0 to the end -- any row but the top and any column but the final one) and you'll be set!
  + [Predict the full-data unknowns] Then, with your k, predict the digit types of the 12 unknown (but full-data) digits -- paste those 12 digit-types in order in the triple-quoted string at the bottom of digits.py.
  + [Predict the partial-data unknowns] Next, build a knn model (same or different) that predicts the digit types of the 10 unknown digits (with partially-erased-data) digits -- paste those 10 digit-types in the triple-quoted string, too.
  + Also paste those into the bottom of your file, along with a short comment about how you handled the partially-erased data challenge -- and how the digits challenge went overall.
* ***EC Option***: For up to +10 points extra-credit, have your code predict *what pixels are missing from the 10 "partly-erased" digits* at the top of the digits.csv file. There are many ways to approach this… The key library-function resource is

distances, indices = knn.kneighbors(X)

which is linked/demoed at the [top of this page](http://scikit-learn.org/stable/modules/neighbors.html), and allows you to find the indices of the k nearest neighbors from within the data set. From there, you could, for example, average the pixels those neighbors have in order to estimate the missing data!

Alternatively, you could build a predictor for *each* of the pixels that's missing (use loops rather than 24 separate calls!) Either way, if you do this, be sure to call it out -- and to include at least three side-by-side screenshots of the filled-in results! Or, more ambitiously, create an image/screenshot of all 10 of the partially-erased digits, now with the pixels filled in!

**Problem 3: kNN for Titanic data**

[35 pts; with graphics EC available!]

* This problem asks you to run/write your code in the **titanic.py** file.
* In addition, you'll submit your model's answers to the 42 unknown outcomes at the bottom of your file
* This problem asks you to model an even more intricate dataset with kNN: the survival data of some of the passengers on the Titanic.
* In titanic.csv, many of the problems with real data sets are illustrated. Using the experience from iris.py and digits.py, you will first want to
  + Drop any columns with too few valid values (using the file's example)
  + There will be rows with missing data - drop those (there's a df.dropna() call there already -- it does this task)
  + Convert any columns with string (object) data to floating-point or int data
  + Be sure not to use the 'survived' column in the input data (if you get accuracy rates over 90%, chances are you've used it!) For example, you can create y\_data\_full using df( 'survived' ).values and then drop that column before creating X\_data\_full.
  + As with the prior two datasets, use at least 10-fold cross validation (with the \_average\_ success score for those 10 iterations) to determine a k (**n\_neighbors**) that works as well as possible.
  + Also, your results will *substantially benefit* from using cross-validation to help you *reweight* columns: some of those columns are more important than others. For example, you don't want male/female to be 0/1 and the fare to be measured in dollars: the gender difference will have too little impact!
  + Measure how high you're able to get the cross-validation (testing) score as you change column weights and *k*  - this is a much more difficult dataset, and it's not easy to get the score above 0.7 (and much harder to get it above .75 or .8! I've never seen anything above .85...)
  + [Predict the full-data unknowns] Then, with your k, predict whether the 42 "unknown-outcome" passengers survived or perished! Paste your results at the bottom of the file, along with a note about how the overall process went.
* ***EC Options***: (For up to +5 points) Add graphs/graphics! Visualize the Titanic data in anyway you'd like (and you're welcome to use/follow the many online guides that can help with this -- the Titanic data is a very popular dataset!) As usual, this is wide open -- the visualizations can range from histograms of ages or survival rates depending on gender or other variables.

**Problem 4: Neural Networks**

[??? pts]

In this problem, you will be analyzing data from a Boston housing tract in 1978. You will use a neural network with three hidden layers of equal size to predict the median value (MV) of each of the first 20 houses.

First, paste the following lines in the terminal to install from scikit:

* pip install cython
* pip install git+https://github.com/scikit-learn/scikit-learn.git

You will now use the scikit **MLPRegressor** to predict the house prices.

* If you haven't already, download the necessary files [here](https://github.com/ScriptingBeyondCS/CS-35/tree/master/week_4).
* Grab the data from boston.py and create divide up your data into training and testing.
* Add loops to the cross-validation step so that the script 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.
* Then loop through hidden layer sizes of 1, 5, 10, 20, 40, 80, and 120 and use cross validation scores to determine the best size.
* Finally run the MLP with the best size on the entire dataset to predict the unknown house prices.

**Extra**: Play with the parameters (especially number of hidden layers and the size of each) to get the highest cross-validation testing score. You can also use other activation functions, solving algorithms, learning rates, etc. (documentation link below). If you get over 0.865 , add a comment at the bottom of your code specifying the parameters you used.

<http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html>

**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). If you do, let us know (and provide a direct link :-)