CS 3430: S20: SciComp with Py
Final Exam (aka Midterm 3)

Vladimir Kulyukin
Department of Computer Science
Utah State University

April 23, 2020

Instructions

1. This exam has 10 problems worth a total of 30 points; you may use your class notes, class slides and screencasts on Canvas, reading materials distributed in class, class code samples, and, most importantly, your homework solutions in solving these problems; **you may not use any other materials while working on your solutions**.
2. You'll submit your solutions in the included file `midterm03_s20.py`. Don't copy and paste the problem statements from this pdf into `midterm03_s20.py`. **Remember to write your name and A-number at the top of the file**.
3. I've written the test runs for some problems as unit tests in `cs3430_s20_midterm03_uts.py` (also included). Use that file to test your coding solutions.
4. You may not talk to anyone during this exam orally, digitally, in writing, by phone, by sign language, or through telepathy. Should I detect telepathic communication, I'll punish you by assigning your final score from the interval [0, 1] with a random number generator. **You must work on your own**.
5. You may use your Python IDE for numerical calculations.
6. **If you use your homework solutions, you must include the homework files you use into your submission**. For example, if, in `midterm03_s20.py`, you use an import statement

   ```
   from cs3430_s20_hw_x import y
   ```

   to import some function/class method `y` from `cs3430_s20_hw_x.py`, then you must include `cs3430_s20_hw_x.py` into your submission. If you fail to do so and your code doesn't pass a unit test due to a failed import, I'll be able to give only partial credit at best.
7. You have 33 hours to complete this exam. You solutions in `midterm03_s20.py` are due in Canvas by 11:59pm on April 24, 2020. If you have multiple files to submit, because of your imports, zip everything up in `midterm03_s20.zip` and submit the zip.
8. I thank you all for taking my class and wish you all best of luck and, as always, Happy Hacking!

Problem 1 (3 points)

Use the images in the folder `hinx` for this problem. The directory `hinx/img` contains 10 images (5 png's and 5 jpeg's) that you'll index. The directory `hinx/pck` is the directory where you'll pickle your dictionaries mapping image paths to histograms.

Implement the function `hinx_img(imgp, hist_index, color_space, num_bins=8)` that takes a path to an image (`imgp`), a dictionary (`hist_index`), a string specification of the color space (`color_space`), which is either `'rgb'` or `'hsv'`, and a number of bins in each color channel (`num_bins`) in the histogram (either 8 or

16). The number of bins parameter is used in the OpenCV `cv2.calcHist()` function. The function `hinx_img` places the key-value pair `(imgp, norm_hist)` in `hist_index`. Depending on the value of `color_space`, the `norm_hist` is either the normalized and flattened RGB histogram of the image in `imgp` or the normalized and flattened HSV histogram of the same image.

Implement the function `hinx_img_dir(imgdir, hist_index, color_space, num_bins, pick_file)` that uses `hinx_img()` to index all jpg and png images in the directory `imgdir` in the dictionary `hist_index`. The `color_space` and `bin_size` arguments are as in `hinx_img`. The argument `pick_file` specifies the path to a pickle file where `hist_index` is persisted.

Here's a test run.

```
hist_index = {}
hinx_img_dir('hinx/img', hist_index, 'rgb', 8, 'hinx/pck/rgb_hist8.pck')
assert len(hist_index) == 10
```

This test generates the following output and creates the pickle file `hinx/pck/rgb_hist8.pck`.

```
Indexing hinx/img/...
indexing hinx/img/img10.jpg
hinx/img/img10.jpg indexed
indexing hinx/img/img6.jpg
hinx/img/img6.jpg indexed
indexing hinx/img/img7.jpg
hinx/img/img7.jpg indexed
indexing hinx/img/img5.png
hinx/img/img5.png indexed
indexing hinx/img/img9.jpg
hinx/img/img9.jpg indexed
indexing hinx/img/img1.png
hinx/img/img1.png indexed
indexing hinx/img/img8.jpg
hinx/img/img8.jpg indexed
indexing hinx/img/img3.png
hinx/img/img3.png indexed
indexing hinx/img/img4.png
hinx/img/img4.png indexed
indexing hinx/img/img2.png
hinx/img/img2.png indexed
indexing finished
```

Here's another test run that creates the pickle file `hinx/pck/hsv_hist16.pck`.

```
hist_index = {}
hinx_img_dir('hinx/img/', hist_index, 'hsv', 16, 'hinx/pck/hsv_hist16.pck')
assert len(hist_index) == 10
```

Problem 2 (3 points)

Implement the functions `find_sim_rgb_images(imgpath, num_bins, hist_index, hist_sim)` and `find_sim_hsv_images(imgpath, num_bins, hist_index, hist_sim)`. The function `find_sim_rgb_images()` finds and displays the top 2 images most similar to the image in `imgpath` by computing the similarity scores between this image and all images in a given histogram dictionary `hist_index` in the RGB space. The parameter `num_bins` should be 8 or 16. Use the following strings as values of `hist_sim` (i.e., the histogram similarity metric): `correl` – for correlation, `chisqr` – for chi square, `inter` – for intersection, and `bhatta` – for bhattacharrya. The function `find_sim_hsv_images()` implements the same functionality in the HSV space.

Below are a few test runs on the 8-bin histogram index in the RGB space.

```
hist_index = load_hinx('hinx/pck/rgb_hist8.pck')
assert len(hist_index) == 10
imgpath = 'hinx/test/test1.jpg'
print(imgpath)
top_matches = find_sim_rgb_images(imgpath, 8, hist_index, 'inter')
print('similarity metric: inter')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
print('similarity metric: correl')
top_matches = find_sim_rgb_images(imgpath, 8, hist_index, 'correl')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
print('similarity metric: chisqr')
top_matches = find_sim_rgb_images(imgpath, 8, hist_index, 'chisqr')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
print('similarity metric: chisqr')
top_matches = find_sim_rgb_images(imgpath, 8, hist_index, 'bhatta')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
del hist_index
```

Here's my output.

```
hinx/test/test1.jpg
similarity metric: inter
hinx/img/img6.jpg --> 2.6241015265858323
hinx/img/img10.jpg --> 1.9212639209071085
similarity metric: correl
hinx/img/img6.jpg --> 0.46133169483187464
hinx/img/img10.jpg --> 0.25405138915865694
similarity metric: chisqr
hinx/img/img9.jpg --> 21.13220309863486
hinx/img/img8.jpg --> 25.122591590634556
similarity metric: bhatta
hinx/img/img6.jpg --> 0.4639969990443348
hinx/img/img10.jpg --> 0.6143115372112586
```

Now a few test runs on the 16-bin histogram index in the RGB space.

```
hist_index = load_hinx('hinx/pck/rgb_hist16.pck')
assert len(hist_index) == 10
imgpath = 'hinx/test/test1.jpg'
print(imgpath)
top_matches = find_sim_rgb_images(imgpath, 16, hist_index, 'inter')
print('similarity metric: inter')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
print('similarity metric: correl')
top_matches = find_sim_rgb_images(imgpath, 16, hist_index, 'correl')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
print('similarity metric: chisqr')
top_matches = find_sim_rgb_images(imgpath, 16, hist_index, 'chisqr')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
print('similarity metric: bhatta')
top_matches = find_sim_rgb_images(imgpath, 16, hist_index, 'bhatta')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
del hist_index
```

Here's my output.

```
hinx/test/test1.jpg
similarity metric: inter
hinx/img/img6.jpg --> 3.6126040376209403
hinx/img/img7.jpg --> 2.3348736308337266
similarity metric: correl
hinx/img/img6.jpg --> 0.3850049182824728
hinx/img/img7.jpg --> 0.14619678849832404
similarity metric: chisqr
hinx/img/img6.jpg --> 200.98192012921248
hinx/img/img8.jpg --> 263.624892014442
similarity metric: bhatta
hinx/img/img6.jpg --> 0.5509965847243383
hinx/img/img7.jpg --> 0.6917197799035861
```

Let's do a few runs on the 8 bin histogram index in the HSV space.

```
hist_index = load_hinx('hinx/pck/hsv_hist8.pck')
assert len(hist_index) == 10
imgpath = 'hinx/test/test1.jpg'
print(imgpath)
top_matches = find_sim_hsv_images(imgpath, 8, hist_index, 'inter')
print('similarity metric: inter')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
print('similarity metric: correl')
top_matches = find_sim_hsv_images(imgpath, 8, hist_index, 'correl')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
print('similarity metric: chisqr')
top_matches = find_sim_hsv_images(imgpath, 8, hist_index, 'chisqr')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
print('similarity metric: bhatta')
top_matches = find_sim_hsv_images(imgpath, 8, hist_index, 'bhatta')
for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
del hist_index
```

Here's my output.

```
hinx/test/test1.jpg
similarity metric: inter
hinx/img/img6.jpg --> 2.797683347072706
hinx/img/img7.jpg --> 1.5187115583273112
similarity metric: correl
hinx/img/img6.jpg --> 0.4555350986629035
hinx/img/img7.jpg --> 0.09559583768980101
similarity metric: chisqr
hinx/img/img7.jpg --> 223.9691658927892
hinx/img/img8.jpg --> 241.78071688749355
similarity metric: bhatta
hinx/img/img6.jpg --> 0.4958741918177958
hinx/img/img7.jpg --> 0.687309668232793
```

Problem 3 (3 points)

1. Write the function `bsubst(a, n, b, m)` that uses back substitution to solve the linear system $ax = b_1, b_2, ..., b_m$, where a is an $n \times n$ upper-triangular matrix and b is an $n \times m$ matrix of vectors $b_1, b_2, ..., b_m$.

2. Write the function `fsubst(a, n, b, m)` that uses forward substitution to solve the linear system $ax = b_1, b_2, ..., b_m$, where a is an $n \times n$ lower-triangular matrix and b is an $n \times m$ matrix of vectors $b_1, b_2, ..., b_m$.

A couple of test runs.

```
A = np.array([[1, 3, -1],
              [0, 2, 6],
              [0, 0, -15]],
             dtype=float)
b = np.array([[-4, 11],
              [10, -5],
              [-30, 50]],
             dtype=float)
x = bsubst(A, 3, b, 2)
>>> print(x)
[[  1.         -14.83333333]
 [ -1.           7.5       ]
 [  2.          -3.33333333]]

A = np.array([[1, 0, 0],
              [2, 1, 0],
              [-1, 3, 1]],
             dtype=float)
b = np.array([[-4, -10],
              [2, 4],
              [4, 8]],
             dtype=float)
x = fsubst(A, 3, b, 2)
>>> print(x)
[[ -4. -10.]
 [ 10.  24.]
 [-30. -74.]]
```

Problem 4 (3 points)

Recall that the simplex tableau is a 2-tuple that consists of the dictionary with the in-variables and the tableau's matrix (i.e., `(in_vars, m)`). Implement the function `do_simplex(tab)` that takes a tableau represented as a 2-tuple, runs the simplex algorithm on the tableau, and returns two values: the modified tableau and the boolean variable that is `True` when the returned tableau contains a solution and `False` when the returned tableau does not contain a solution (i.e., the problem formulated by the initial tableau has no solution).

Here's a test run that uses the simplex algorithm to solve the following SMP. Maximize $p = 10x_0 + 6x_1 + 2x_2$ subject to

1. $x_0 \geq 0$;
2. $x_1 \geq 0$;
3. $x_2 \geq 0$;
4. $2x_0 + 2x_1 + 3x_3 \leq 160$;
5. $5x_0 + x_1 + 10x_3 \leq 100$.

```
in_vars = {0:3, 1:4}
m = np.array([[2,    2,   3, 1, 0, 160],
              [5,    1,  10, 0, 1, 100],
              [-10, -6,  -2, 0, 0, 0]],
             dtype=float)
tab = (in_vars, m)
tab, solved = do_simplex(tab)
```

```
assert solved is True
>>> display_solution_from_tab(tab)

x1 = 75.0
x0 = 5.0
p = 500.0
```

Use `do_simplex()` to solve the following SMP and state your answer in the comments in `midterm03_s20.py` next to your definition of `do_simplex()`. Maximize $p = 4x_0 + 10x_1 + 8x_2$ subject to

1. $x_0 \geq 0$;
2. $x_1 \geq 0$;
3. $x_2 \geq 0$;
4. $15x_0 + 5x_1 + 4x_3 \leq 6500$;
5. $10x_0 + 8x_1 + 4x_3 \leq 11000$;
6. $12x_0 + 4x_1 + 4x_3 \leq 6000$;

Problem 5 (4 points)

Implement the function `dir_corr(fixed, dancer)` that takes two 2D numpy matrices, `fixed` and `dancer`, and returns a 2D numpy array `C` with their correlation coefficients.

Here's a test run. Let's compute the direct correlation matrix C12 between M1 and M2 and the direct correlation matrix C21 between M2 and M1.

```
M1 = np.array([[162., 118., 111., 133.],
               [ 64.,  37.,  33., 165.],
               [ 38.,   4., 107.,  86.],
               [ 98.,  35.,  67., 107.]])
M2 = np.array([[28, 29, 86, 45, 42],
               [46,  7,  7, 36, 90],
               [27, 42, 22, 44, 79],
               [25,  4, 77, 29, 38],
               [41, 30, 26, 95, 99]],
              dtype=float)
C12 = dir_corr(M1, M2)
>>> print(C12)
[[16038. 27072. 26411. 31640. 25703. 11626.  8541.  5453.]
 [12492. 18925. 28560. 40359. 37193. 20432.  9610. 10090.]
 [18992. 23718. 36741. 55046. 39328. 30146. 17035. 11242.]
 [30782. 36220. 40629. 63554. 53785. 26528. 22834. 17110.]
 [19290. 24040. 49226. 64454. 51047. 36925. 18242. 16311.]
 [13850. 13239. 27584. 41626. 29047. 23831. 17536. 11465.]
 [10416.  8556. 15918. 22846. 23326. 13439.  9321.  7330.]
 [ 4116.  5880. 12817. 13361. 14336. 12125.  4979.  2996.]]
C21 = dir_corr(M2, M1)
>>> print(C21)
[[ 2996.  4979. 12125. 14336. 13361. 12817.  5880.  4116.]
 [ 7330.  9321. 13439. 23326. 22846. 15918.  8556. 10416.]
 [11465. 17536. 23831. 29047. 41626. 27584. 13239. 13850.]
 [16311. 18242. 36925. 51047. 64454. 49226. 24040. 19290.]
 [17110. 22834. 26528. 53785. 63554. 40629. 36220. 30782.]
 [11242. 17035. 30146. 39328. 55046. 36741. 23718. 18992.]
 [10090.  9610. 20432. 37193. 40359. 28560. 18925. 12492.]
 [ 5453.  8541. 11626. 25703. 31640. 26411. 27072. 16038.]]
```

Problem 6 (2 points)

Compute the NRA approximations for the following values, numbers of iterations, and initial guesses.

1. $\sqrt{333}$ for 15 iterations and an initial guess of 1;
2. $\sqrt{11}$ for 5 iterations and an initial guess of 1;
3. $10^{1/5}$ for 20 iterations and an initial guess of 1;
4. $11^{1/3}$ for 7 iterations and an initial guess of 1;

Write your answers to problem 6 as the value of the appropriate variables in `midterm03_s20.py` and state how you've computed them. You may simply import a function/class method from a module and then use that function/ class method to compute the value. For example,

```
from x import y
midterm03_s20_problem_6_1 = y(...)
```

You may not use scientific calculators/MATLAB/Octave/etc. You have to use your implementation of the NRA from a previous assignment or write it from scratch. If you import, remember to include the imported module into your submission.

Problem 7 (2 points)

Use the central divided difference (CDD) formulas of the specified order to compute the approximations of the derivatives of the following functions at given points and values of h.

1. $f(x) = e^{2.5x}$, $f'(0.41)$, $h = 0.00001$, CDD of order 2 = ?;
2. $f(x) = e^{3.5x}$, $f'(0.51)$, $h = 0.00001$, CDD of order 4 = ?;
3. $f(x) = e^{4.5x}$, $f'(0.61)$, $h = 0.00001$, CDD of order 2 = ?;
4. $f(x) = e^{5.5x}$, $f'(0.71)$, $h = 0.00001$, CDD of order 4 = ?;

State which functions, methods, and classes and from which assignments you used to compute the values. Write your answers to problem 7 as the value of the variables in `midterm03_s20.py` and state how you've computed them. You may simply import a function/class method from a module and then use that function/class method to compute the value. For example,

```
from x import y
midterm03_s20_problem_7_1 = y(...)
```

You may not use scientific calculators/MATLAB/Octave/etc. You have to use your implementation of the CDD from a previous assignment or write it from scratch. If you import, remember to include the imported module into your submission.

Problem 8 (4 points)

1. Write the function `build_huffman_tree_from_text(txtstr)` that takes a string `txtstr` and builds a Huffman tree from the frequencies of each character in the string `txtstr`.
2. Use your solutions to Assignment 12 to write the function `encode_moby_dick_ch03()` that produces the Huffman tree binary encoding of Chapter 3 from "Moby Dick" given in `moby_dick_ch03.txt` and saves the encoding in the current directory. This function should produce two files – `moby_dick_ch03.bin` and `moby_dick_ch03_pb.txt`. In your comments next to `encode_moby_dick_ch03()`, subtract the sum of the sizes of `moby_dick_ch03.bin` and `moby_dick_ch03_pb.txt` from the size of `moby_dick_ch03.txt` and state how many bytes your encoding saved.

Here's a test run and my output for part 1 of this problem.

```
>>> from HuffmanTreeNode import HuffmanTreeNode
>>> from HuffmanTree import HuffmanTree
>>> txt = 'aababbcdefghaaaaa'
>>> ht = learn_huffman_tree_from_text(txt)
>>> HuffmanTree.displayHuffmanTree(ht)
{'d', 'e', 'a', 'h', 'c', 'b', 'g', 'f'}:17
        {'a'}:8
        {'d', 'b', 'e', 'c', 'g', 'f', 'h'}:9
                {'d', 'c', 'e', 'f'}:4
                        {'d', 'c'}:2
                                {'c'}:1
                                {'d'}:1
                        {'e', 'f'}:2
                                {'e'}:1
                                {'f'}:1
                {'b', 'g', 'h'}:5
                        {'g', 'h'}:2
                                {'g'}:1
                                {'h'}:1
                        {'b'}:3

>>> for c in txt:
        enc = ht.encodeSymbol(c)
        print('encoding({}) = {}'.format(c, enc))
        dec = ht.decode(enc)
        assert c == dec
        assert enc == ht.encodeSymbol(ht.decode(enc))

encoding(a) = 0
encoding(a) = 0
encoding(b) = 111
encoding(a) = 0
encoding(b) = 111
encoding(b) = 111
encoding(c) = 1000
encoding(d) = 1001
encoding(e) = 1010
encoding(f) = 1011
encoding(g) = 1100
encoding(h) = 1101
encoding(a) = 0
encoding(a) = 0
encoding(a) = 0
encoding(a) = 0
encoding(a) = 0
```

Problem 9 (4 points)

Use the files `train_data.csv` and `test_data.csv` for this problem. The file `train_data.csv` contains 123 discretized training examples generated from the IRIS dataset. The file `test_data.csv` contains 27 discretized test examples generated from the IRIS dataset.

1. Implement the function `learn_bin_id3_dt_from_csv_file(csv_fp, target_attrib)` that takes a path to a csv file of training examples, applies the binary ID3 algorithm to them with the target attribute specified by the second argument `target_attrib`, and returns the root of the learned binary ID3 decision tree (i.e., the object of the `id3_node` class in `bin_id3.py` you used in Assignment 11).

2. Implement the function `classify_csv_file_with_bin_id3_dt(dt_root, csv_fp, target_attrib)` that takes the root of the binary ID3 tree returned by `learn_bin_id3_dt_from_csv_file()` (`dt_root`),

a path to a csv file with test examples (`csv_fp`) and the target attribute `target_attrib`, and returns the average classification accuracy of `dt_root` on the test examples in `csv_fp`.

Here's a test run of `learn_bin_id3_dt_from_csv_file()` and my output. Your tree may look different but it shouldn't be deeper.

```
>>> dt_root = learn_bin_id3_dt_from_csv_file('train_data.csv', 'Class')
>>> assert not dt_root is None
>>> display_bin_id3_node(dt_root)

PetLen
        PetLen1
                No
        PetLen2
                No
        PetLen0
                Yes
        PetLen4
                No
        PetLen3
                No
```

Here's a test run of `classify_csv_file_with_bin_id3_dt()` and my output. Your classification accuracy doesn't have to be 100%, but it should definitely be $\geq 90\%$, because it's a pretty simple dataset.

```
>>> dt_root = learn_bin_id3_dt_from_csv_file('train_data.csv', 'Class')
>>> assert not dt_root is None
>>> acc = classify_csv_file_with_bin_id3_dt(dt_root, 'test_data.csv', 'Class')
>>> print('classification accuracy = {}'.format(acc))
classification accuracy = 1.0
```

Problem 10 (2 points)

Briefly explain what the missing attribute value problem is in the context of the ID3 algorithm and how you handled it in Assignment 11. Don't write an essay. Two-three short paragraphs will suffice. Write your answer inside the multi-line comment in `midterm03_s20.py`.