# Midterm - 10-605

Oct 17, 2017

**10-605** **Name:**_____
**Fall 2017**
**Midterm Exam Answer Key** **Andrew ID:**_____
**Time Limit: 80 Minutes**

---

Grade Table (for teacher use only)

| Question | Points | Score |
|---|---|---|
| 1 | 12 | |
| 2 | 10 | |
| 3 | 14 | |
| 4 | 12 | |
| 5 | 10 | |
| 6 | 6 | |
| 7 | 8 | |
| 8 | 4 | |
| 9 | 10 | |
| 10 | 12 | |
| 11 | 2 | |
| Total: | 100 | |

Known Notes:

| *Label* | *Content* |
|---|---|
| Fall |  |
| Fall |  |
| Fall |  |
| Winter |  |
| Winter |  |

Unknown Note A:

| *Label* | *Content* |
|---|---|
| ? |  |

Unknown Note B:

| *Label* | *Content* |
|---|---|
| ? |  |

Figure 1: Buttercup's Notes

1. (12 points) TA: Tao Lin **Naïve Bayes** Buttercup loves to take notes in Emoji. In the past, she manually labeled each note with one category. After taking 10-605/805, she decided to build a Naïve Bayes classifier to label the notes automatically. Above are five notes with known labels and two notes with unknown labels.

(a) (2 points) Under the Naïve Bayes model, what is the probability of seeing the Note **A** if you know the label is **Fall**, using Laplace smoothing with $\alpha = 1$ (as in the assignments, i.e., estimating $P(x) = \frac{k+\alpha}{N+\alpha d}$, where $d$ is the number of possible outcomes). Circle the correct answer.

A. $\frac{3+1}{6+5}$

B. $\frac{3+1}{6+3}$

C. $\frac{2+1}{3+5}$

D. $\frac{2+1}{3+3}$

(b) (4 points) What is the probability of seeing the Note **B** if you know the label is **Fall**? What about **Winter**? Use Laplace smoothing with $\alpha = 1$, and write your answer as a product of fractions, as above.

$P(\text{Note B}|\text{Fall}) = \frac{3+1}{6+5} \times \frac{0+1}{6+5} = \frac{4}{11} \times \frac{1}{11} = \frac{4}{121}$

$P(\text{Note B}|\text{Winter}) = \frac{1+1}{5+5} \times \frac{3+1}{5+5} = \frac{2}{10} \times \frac{4}{10} = \frac{8}{100} = \frac{2}{25}$

(c) (4 points) Which label would be assigned to the Note **B** by a Naïve Bayes classifier? How is this related to the probabilities in question (b)? Use Laplace smoothing with $\alpha = 1$.

Winter. (2 points)

$\frac{P(Fall|NoteB)}{P(Winter|NoteB)} = \frac{P(NoteB|Fall)P(Fall)/P(NoteB)}{P(NoteB|Winter)P(Winter)/P(NoteB)} < 1$ (2 points)

(d) (2 points) How many event counters do we need to keep if we only want to compute $P$(Note B|Winter)?

A. 2

B. 3 ("X=ANY,Y=Winter", "X=umbrella,Y=Winter", "X=snowman,Y=Winter")

C. 4 (1 point, additional "X=snowflake,Y=Winter")

D. 5

2. (10 points) TA: Bo Chen **SGD short answer questions**. Circle the correct answer.

(a) In logistic regression, we model $p(y = 1|x) = \text{sigmoid}(w^T x)$. If some sparse feature $x_j$ takes value of {-1, 0} and only occurs in positive examples, the corresponding weight $w_j$ will get smaller in absolute value after many iterations. **True** **False**

(b) The hashing trick is mainly used to **reduce memory consumption** / **enable parallelism**.

(c) With non-lazy L2 regularization, SGD would apply weight decay to features in an example **before** / **after** / **both before and after** it computes the prediction.

(d) Unregularized SGD using the hash trick becomes more likely to overfit as the hash table becomes smaller. **True** **False**

(e) Using lazy L2 regularization, if the learning rate changes between epoch $i$ and $i+1$, it is necessary to apply weight to decay to all the parameters and the end of epoch $i$. **True** **False**

3. (14 points) TA:Yifan Yang **Map-reduce operations.** You work for the startup Twitler, where a user has a list of followers and a registration date. Consider an example input below. Each row contains the user name, registration date with format YYYYMMDD and then a non-empty list of followers of that user.

```
Alice,20120802,Nick,Mike,Bob
Nick,20150903,Alice
Mike,20110101,Alice,Bob
Bob,20170901,Nick,Alice
```

You suspect that a lot of spam accounts were added on or after Sept 1, 2017. As part of your test, you want to get, for each user, the set of all followees (not followers) that registered before this date. For example:

```
Alice:Nick,Mike
Bob:Alice,Mike
Mike:Alice
Nick:Alice
```

Please fill the blanks of the following Guinea Pig code, which performs this task (except for the final formatting stage).

```
class GetOldFollowers(Planner):
    users = ReadLines('tmp.txt') \
     | Map(by=lambda line:line.strip().split(",")) \
     | Map(by=lambda parts:(parts[0],parts[1],parts[2:]))

    oldUsers = Filter(users,by= __lambda (u,d,f): d < "20170901" )__

     | Map(by=lambda (user,regDate,followers):user)

    followerPairs = __FlatMap/Flatten__(users, \

           by=lambda(user,regDate,followers):[(f,user) for f in followers])

    oldFollowers = Join( Jin(followerPairs,by=__lambda (f,u) : u)__,

                         Jin(__oldUsers__, by=__lambda u: u__)

     | Map(by=lambda ((f,u),_):(f,u))
    result = Group(oldFollowers,

                   by=lambda(f,u): __f__

                   retaining= __lambda (f,u) : u)__
```

4. (12 points) TA: Anant & Ning Dong **Map-reduce and joins.** You work for a small company (<1000 employees) full of famous people, where every employee has a Wikipedia page. Internal employee data resides in a single file with each line having the format `(employee name, employee details)`. Assume all the Wikipedia data related to famous personalities is available to you in a huge file with each line containing `(personality name, wikipedia data)`.

   (a) (3 points) You decide to combine the internal company data with the external Wikipedia data. Fortunately, the names in both databases are very clean so you can use a hard join to match the names. Should you use a map-side or a reduce-side join in order to combine the data? Justify your choice using one sentence.

   Map-side: Smaller (< 1000 line relation) will fit in memory and map-side avoids doing a shuffle-sort

   (b) (3 points) A famous professional networking company, LinkedOut, decides to use your algorithm against the Wikipedia data, but LinkedOut has over a billion user account details. Should they use a map-side or a reduce-side join? Justify your answer in one sentence.

   Reduce-side join is better for two large relations since Wikipedia data may not fit in memory.

   The huge company LinkedOut knows that some spammers operate by finding profiles for famous people on Wikipedia, finding a LinkedOut profile with exactly the same name, and then making a slightly modified copy of that profile. LinkedOut wants to locate these spam profiles as part of a quality improvement initiative, but unfortunately, there have been a lot of such initiatives since they were acquired by Microsloth, so their chief scientists have their hands full. Help them out by circling the **best** answers in the following plan. You may assume that the same Wikipedia dataset is available to them, and that is has a few million lines (personalities) in it.

   (c) (2 points) We first begin by identifying the LinkedOut profiles that belong to famous personalities. This can be done using a _____ of the LinkedOut name data with the Wikipedia name data.

      A. hard join

B. soft join

C. concatenation

(d) (2 points) We can now represent each of these famous profiles as a _____ containing _____ values.

A. graph, PageRank

B. vector, TF-IDF

C. matrix, co-occurrence

(e) (2 points) A self-join is just joining a table to itself. We can think of the collection of filtered profiles (with each profile having the representation from the previous part) as a table. One row corresponds to one profile. In order to identify the **nearly-duplicate** rows, we can perform a _____ with _____.

A. self hard join, exact name as the key

B. self soft join, with cosine similarity as a measure of closeness

C. PageRank computation, with an appropriate random restart value for each profile

5. (10 points) TA: Minxing Liu

**Perceptrons**

(a) (2 points) When deriving the error bound of the Perceptron algorithm, we make two assumptions. Fill in the blanks below to complete the assumptions.

"Margin": The training data can be separated with some vector $\mathbf{u}$ with margin $\gamma > 0$, e.g.,

$$\exists \mathbf{u}, ||\mathbf{u}|| = 1 : \forall (\mathbf{x}_i, y_i) \text{ given as examples, } (\mathbf{u} \cdot \mathbf{x}_i) y_i \quad \underline{> \text{ or } \geq} \quad \gamma$$

"Radius": The training data is "near the origin" with radius R, e.g.,

$$\forall \mathbf{x}_i \text{ given as examples, } ||\mathbf{x}_i|| \quad \underline{< \text{ or } \leq} \quad R$$

(b) (6 points) We have 4 training data points ($\mathbf{x}_i$, $y_i$), which are ([4, 0], 1), ([1, 1], -1), ([0, 1], 1), ([-2, -2], 1).

(1) After training on the data, you will get some weight vectors $\mathbf{w}_i$, which will be used for predicting new data points. The weight vectors used for Perceptron and Average Perceptron are different. Please write them down below. (Write something like $\mathbf{w}_1$=[1 1], $\mathbf{w}_2$=[2 3], etc.)

Weight vector(s) for Perceptron (2 pts):

[1, -2]

Partial credits will be provided if you get close.

Weight vector(s) for Average Perceptron (2 pts):

$[\frac{11}{4}, \frac{-3}{4}]$

Partial credits will be provided if you get close.

(2) Despite often achieving high accuracy, the voted perceptron is rarely used in practice. In one sentence, give one reason why. (2 pts)

1. Voted perceptron has to store all the weight vectors along the way, thus it's memory inefficient and computational inefficient.

2. Voted perceptron is hard to do efficient lazy update as the average perceptron.

Either can get full score. Some deduction will be made if you didn't explain the reason clearly.

(c) (2 points) Suppose we have 4 training examples, ($\mathbf{x}_i$, $y_i$), where $y_i \in \{-1, 1\}$. After training, the weight vector $\mathbf{w} = \mathbf{x}_1 - \mathbf{x}_4$. Circle the correct answer regarding the predictions on the training data.

The prediction for $\mathbf{x}_2$ is **false positive/false negative/correct/cannot determine (both are correct)**.

The prediction for $\mathbf{x}_4$ is **false positive/false negative/correct/cannot determine.**

6. (6 points) TA: Ning Dong **Perceptron variants** Fill in the blank in the algorithms below.

The structured perceptron algorithm:
**Inputs:**
training data $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_m, \mathbf{y}_m)$
**Initialize:**
$\mathbf{w} = 0$
**Learning:**
for each example $(\mathbf{x}_i, \mathbf{y}_i)$:
$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}'} F(\mathbf{x}_i, \mathbf{y}') \cdot \mathbf{w}$

$\mathbf{w} = \mathbf{w} + F(\mathbf{x}_i, \mathbf{y}_i) - F(\mathbf{x}_i, \hat{\mathbf{y}})$

The ranking perceptron algorithm:
**Inputs:**
training data $(\mathbf{x}_{1,1}, \ldots, \mathbf{x}_{1,k_1}; j_1), \ldots, (\mathbf{x}_{m,1}, \ldots, \mathbf{x}_{m,k_m}; , j_m)$
**Initialize:**
$\mathbf{w} = 0$
**Learning:**
for each example $(\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,k_i}; j_i)$

$\hat{j} = \arg\max_{j'} \mathbf{x}_{i,j'} \cdot \mathbf{w}$

$\mathbf{w} = \mathbf{w} + \mathbf{x}_{i,j_i} - \mathbf{x}_{i,\hat{j}}$

7. (8 points) TA: Chen Hu **Distributed SGD for MF**

In distributed matrix factorization, we decompose a matrix into blocks, and group a subset of blocks into a stratum. Gemulla et al in their paper define a stratum as *valid* if all blocks are interchangeable, and *efficient* if it is as large as it can be, without compromising interchangeability.

Given a $m \times n$ matrix, we decompose it into blocks of size $d \times d$:

(a) (2 points) if $m = 64$, $n = 64$, $d = 16$, how many blocks are in an valid and efficient strata?

Maximum 4 blocks in an valid and efficient strata.

(b) (4 points) if $m = 64$, $n = 64$, $d = 16$, how many valid and efficient strata can be constructed? (Assume that the blocks are fixed.)

4! = 24 (you will get 3 points if you give answer 4 where you assume each block can only be used once)

(c) (2 points) for $m = 4$, $n = 4$, $d = 1$, draw a valid and efficient stratum which is **not** the same as the main diagonal.

For example, $\begin{bmatrix} X & . & . & . \\ . & . & X & . \\ . & X & . & . \\ . & . & . & X \end{bmatrix}$

8. (4 points) TA: Chen Hu **DSGD for Matrix Factorization**

Consider the following Figure:

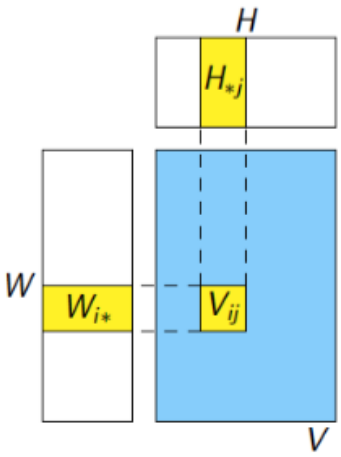


Figure 2: Matrix Factorization

V is the original $m \times n$ matrix, with i as the row index and j as the column index. W and H are the matrices used to approximate V matrix. We seek to *squared loss* with the *L1 norm regularization*:

$$L_{L1} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (V_{i,j} - [WH]_{i,j})^2 + \lambda(||W||_F^1 + ||H||_F^1)$$

where for a matrix $M$

$$||M||_F^1 \equiv \sum_{i,j} |M_{i,j}|$$

Please derive the gradients of the $L_{L1}$ objective function with respect to each entry in the H matrix, using k to denote the column index for W and row index for H.

$\frac{\partial}{\partial H_{k,j}} L_{L1} = \sum_{i=0}^{m-1} (-2W_{i,k}(V_{i,j} - [WH]_{i,j})) + \lambda sign(H_{k,j})$

9. (10 points) TA: Rose **PageRank and Map-Reduce**

Recall that PageRank for $N$ pages is computed using the below equation:

$$\mathbf{v}^{(t+1)} = \alpha\mathbf{u} + (1-\alpha)\mathbf{v}^{(t)}W$$

where, $\alpha$ $(0 < \alpha < 1)$ is the probability that the page hopper will jump to a randomly chosen page, $\mathbf{u} = (\frac{1}{N}, ..., \frac{1}{N})$ is a uniform distribution over the $N$ nodes, $\mathbf{v}^{(t)}$ is a vector representing the PageRank values of the nodes at iteration $t$, and $W$ is a matrix where $w_{ij}$ is the probability of the page hopper moving from page $i$ to page $j$. Now suppose we have a topic of interest, like `Cooking`, `Sports` etc, and we also know for each topic $k$ a "seed set" of pages that are primarily about that topic. We denote this set as $S_k$. We can compute a Topic-Specific PageRank vector $\mathbf{v}_k$ that upweights pages that are more relevant to topic $k$ by modifying the standard PageRank computation to:

$$\mathbf{v}_k^{(t+1)} = \alpha\mathbf{u} + \beta\mathbf{v}_k^{(t)}W + \gamma\mathbf{z}_k$$

where, $0 < \alpha, \beta, \gamma < 1$ are constants such that $\alpha + \beta + \gamma = 1$, and $z_k = (z_1, z_2, ...z_N)$ is uniform over the seed nodes for $C_k$, i.e., it is defined so that

$$z_i = \begin{cases} 1/|S_k| & \text{if } i \in S_k \\ 0 & \text{otherwise} \end{cases}$$

In class we presented a map-reduce implementation of PageRank, which was similar (but not the same) as the one below.

```
p.prevGraph = ReadLines(TMPFILE) | ...
p.outBoundPageRankMessages =
   Flatten(p.prevGraph,
           by=lambda(url,pagerank,outlinks):
                        map(lambda dst:(dst,pagerank/len(outlinks),
                            outlinks)
p.inBoundMessages =
    Group(p.outBoundPageRankMessages,
          by=lambda(dst,deltaPageRank):dst,
          retaining=lambda(dst,deltaPageRank):deltaPageRank),
          reducingTo=ReduceToSum())
p.newPageRank = ????
p.newRankedGraph =
    Join(Jin(p.prevGraph, by=lambda(url,pageRank,outlinks),url),
         Jin(p.newPageRank, by=lambda(url,newPageRank):url)) \
    | Map(by=lambda((url,oldPageRank,outlinks),(_,newPageRank)):
                    (url,newPageRank,outlinks))
p.serializedRankedGraph = ....
```

(a) (3 points) If we defined p.newPageRank=p.inBoundMessages, what update would we be implementing?

$$\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)}W$$

(b) (3 points) In one or two sentences, how should we modify the values in p.inBoundMessages to implement Topic-Specific PageRank?

You have to explain in words, the code you wrote in part (c).

(c) (4 points) Write GuineaPig code to correctly compute p.newPageRank from p.inBoundMessages. Assume that SEEDS is a globally-defined Python set that holds the seed set $S_k$, and that ALPHA, BETA, and GAMMA are the appropriate constants. Hint: It's ok if you want to define an auxiliary function, but you should be able to do this without defining any additional views.

$$\begin{aligned} P.newPageRank = & ReplaceEach(p.inBoundMessages, \\ & by = lambda(dst, val) : (dst, \\ & ALPHA * \frac{1}{N} + BETA * val + \\ & GAMMA * \frac{1}{len(SEEDS)} \text{if dst in SEEDS else 0}) \end{aligned}$$

10. (12 points) TA: Janani **TF-IDF / Rocchio Algorithm**

Consider the following documents:
**Document 1:** *"Albus Severus," Harry said, "you are named for two headmasters of Hogwarts. One of them was a Slytherin and probably the bravest man I ever knew."*
**Document 2:** *"Why are the people staring?" demanded Albus as he and Rose craned around to look at the other students.*
**Document 3:** *It is our choices, Harry, that show what we truly are, far more than our abilities.*

(a) (2 points) Given the above set of documents, write the word that has the highest TF score in the following:

- Document 2: the
- Document 3: our

(b) (2 points) Given the above set of documents, arrange the following terms in the decreasing order of IDF scores (for example, write your answer as: $a > b = c$): Slytherin, Harry, our, are
Slytherin = out > Harry > are

(c) (2 points) Explain in words what the vector space representation of a class $v(y)$ represents, in Rocchio Classification?
It is the Centroid of the documents present in the class. [Any reasonable explanation has been considered and given points accordingly]

(d) (3 points) We require at least 3 passes through the entire data set to create vector space representation (**v(d)**) of the documents during the streaming version of Rocchio Algorithm. Assume that we have enough memory to store **k - hashtables** that map each vocabulary term to some number, one document and a $|V|$ vector where $V$ is the size of the vocabulary.
**True** **False**

(e) (3 points) We require 2 passes through the dataset to compute IDF for tokens, assuming we have enough memory to store one document at a time.
**True** **False**

11. (2 points) **Free points**

You can use this page for scratch work. No matter what you put here (even if you leave it blank) you get two free points.

Yay! :)