SI 650 / EECS 549: Homework 1 — Probabilities, Text, and Ranking

Due: Tuesday, September 21, 11:59pm (see syllabus for late policy)

1 Probabilistic Reasoning and Bayes Rule (25 points)

Sam lets a parent borrow their phone only to discover that they have opened up TikTok and managed to subscribe to thousands of accounts. Their feed is now a useless mess of spam accounts, weird dances, and outrage videos. Sam doesn't have time to deal with all of this manually so they decide to find a programmatic way of unfollowing accounts, starting with getting rid of spam accounts.

Luckily, Sam attended SI 650 / EECS 549 and decided to filter the accounts in a Bayesian way. They went through 12 accounts manually and noted down 4 observations for each account in Table 1 to figure out which features are more likely used in spam accounts.

1. S: whether the account posts mostly spam (1 for yes, 0 for no);
2. B: whether the account's bio has the word "based" (1 for yes, 0 for no);
3. U: whether the account's bio has an URL link (1 for yes, 0 for no);
4. E: whether the account's bio has an emoji (1 for yes, 0 otherwise).

Sam now wants to build a filter with these observations. Now in terms of probabilistic reasoning, we can formulate the question as evaluating the conditional probability $P(S|B,U,E)$, we say that the account is a spam if $P(S=1|B,U,E) > P(S=0|B,U,E)$. We make a further conditional independence assumption that $P(B,U,E|S) = P(B|S)P(U|S)P(E|S)$. In other words, we assume that if the status whether a account is a spam is known (i.e., value of S is known), the values of B, U, and E would be independent to each other.

a) (5 points) Fill in the Table 2 with conditional probabilities using only the information present in the 12 samples.

b) (5 points) With the independence assumption, use the Bayes formula and the calculated conditional probabilities to compute the probabilities that account a with $B=0$, $U=1$, $E=0$ is a spam. That is, compute $P(S=1|B=0,U=1,E=0)$ and $P(S=0|B=0,U=1,E=0)$. Would you conclude that account a is a spam? Show your computation.

c) (5 points) Now, compute $P(S=1|B=0,U=1,E=0)$ and $P(S=0|B=0,U=1,E=0)$ directly from the 12 examples in Table 1, just like what you did in problem A. Do you get the same value as in problem B? Why?

S	B	U	E
0	1	1	1
1	0	1	0
1	1	1	1
0	1	1	0
1	1	0	1
0	0	0	0
1	1	1	0
0	0	1	1
0	0	0	0
1	0	1	1
1	1	1	0
1	1	0	0

Table 1: Sample observations of the accounts

d) (5 points) Now, ignore Table 1, and consider any possibilities you can fill in Table 2. Are there any constraints on these values that we must respect when assigning these values? In other words, can we fill in Table 2 with 8 arbitrary values between 0 and 1? If not, are there any constraints on some values that we must follow? Describe your answer.

e) (5 points) Can you change your conclusion of problem a (i.e., whether account a is a spam) by only changing the value E (i.e., if the account bio has an emoji) in one example of Table 1? Describe your answer.

f) (5 points) Explain why the independence assumption $P(B, U, E|S) = P(B|S)P(U|S)P(E|S)$ does not necessarily hold in reality.

2 Text Data Analyses [30 points]

In this exercise, we are going to get our hands dirty and play with some data in the wild. Download two collections from Canvas, reddit-questions.10k.txt and wiki-bios.10k.txt. The first collection are 100,000 questions randomly sampled from r/AskReddit. The second collection is 100,000 biographies of people taken from Wikipedia. You can also find a stopword list in stoplist.txt.

For text processing, we'll use the SpaCy library, which is a modern NLP library that is well documented and highly performant. You want to use `nlp` function from SpaCy for tokenizing and part-of-speech (POS) tagging.

S	$P(B = 1\|S)$	$P(U = 1\|S)$	$P(E = 1\|S)$	prior $P(S)$
1	0.71428	?	?	?
0	?	?	?	0.41666

Table 2: Conditional Probabilities and Prior

1. (5 points) Tokenize the text using SpaCy and compute the frequency of words. Then, plot the frequency distribution of words in each collection after the removal of the stopwords: x-axis – each point is a word, sorted overall by frequency (number of times a word appears in the collection)[1]; y-axis – how many times the word occurred. Plot this using a log scale on each axis. Does each plot look like a power-law distribution? Are the two distributions similar or different?

2. (10 points) Now compare the two collections more rigorously. Report the following properties of each collection, using SpaCy to POS tag. Can you explain these differences based on the nature of the two collections?

   a) frequency of stopwords (percentage of the word occurrences that are stopwords.);

   b) percentage of capital letters;

   c) average number of characters per word;

   d) percentage of nouns, adjectives, verbs, adverbs, and pronouns;

   e) the top 10 nouns, top 10 verbs, and top 10 adjectives.

3. (10 points) We would like to summarize each document with a few words. However, picking the most frequently used words in each document would be a bad idea, since they are more likely to appear in other document as well. Instead, we pick the words with the highest TF-IDF weights in each document.

   In this problem, term frequency (TF) and inverse document frequency (IDF) are defined as:

   $$TF(t,d) = log(c(t,d)+1)$$

   $$IDF(t) = 1 + log(N/k).$$

   $c(t,d)$ is the frequency count of term t in doc d, N is the total number of documents in the collection, and k is the document frequency of term t in the collection.

   For each of the first 10 documents in the Wikipedia biographies collection, print out the 5 words that have the highest TF-IDF weights. Write whether you think these could be a good summary of the documents.

4. (5 points) As discussed in the class, TF-IDF is a common way to weight the terms in each document. It can also be easily calculated from the inverted index (covered in Week 3), since TF can be obtained from the postings and IDF can be summarized as a dictionary. Could you think of another weighting that cannot be calculated directly from inverted index? What is the advantage of such a weighting?

- **Hint 1:** You can find a tutorial for SpaCy at https://spacy.io/usage/spacy-101 which covers all of the functionality you'll need here as well as many more advanced preprocessing steps.)

[1] This also gets called the "rank" of the word in the collection

- **Hint 2:** You may find a lot of decision to make: Should I lower-case the words? Should I use stemmer or a lemmatizer? What to do with the punctuation? How should I handle html, markdown, or emoji? There is not always right and wrong, different answers are accepted. But you should write down clearly how you process the data in each parts and explain your decisions.

3 Document Ranking and Evaluation (20 points)

Suppose we have a query with a total of 20 relevant documents in a collection of 100 documents. A system has retrieved 20 documents whose relevance status is [++, -, +, ++, -, +, -, ++, +, -, -, +, ++, -, -, +, +, ++, +, -] in the order of ranking. A + or ++ indicates that the corresponding document is relevant, while a - indicates that the corresponding document is non-relevant.

- (10 points) Compute the precision, recall, F1 score, and the mean average precision (MAP).
- (10 points) Consider ++ as the corresponding document being highly relevant ($r_i = 2$), while + indicates somewhat relevant ($r_i = 1$), - being non-relevant ($r_i = 0$). For the nine rest relevant documents, treat them as somewhat relevant ($r_i = 1$) Calculate the Cumulative Gain (CG) at rank 10, Discounted Cumulative Gain (DCG) at rank 10, and Normalized Cumulative Gain (NDCG), at rank 10. Use $\log_2$ for the discounting function.

  Note You may find the definition of DCG in Wikipedia is different from the definition in our lecture. Please use the one in our lecture to calculate DCG and NDCG. (i.e. $DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2 i}$

4 Simple Search (25 points)

Let's build a simple search engine using some of the techniques we learned in class and evaluate it in practice both for its ability to retrieve relevant documents and its speed of retrieval. Here, we'll use our Reddit questions and play the role of an auto-suggest: If someone is about to ask a question on Reddit, they can quickly see if someone else has already asked that question by searching for a few keywords. Specfically, you'll implement a *very* simple search that (1) measures the cosine similarity of a bag-of-words representation of the query and a bag-of-words representation for a document and (2) returns the k most similar documents to the query.

To support the basic search functionality, we'll use the `scikit-learn` package to convert our Reddit questions and queries to bag of words vectors.

- (5 points) Write a function that uses `CountVectorizer` to convert the Reddit questions corpus to vectors. Write a function that given a new query, will convert its text to a vector (using the same vectorizer), estimate the cosine similarity between the query and each document (i.e., each Reddit question) and return the 10 most similar.

- (10 points) Using your method,
  - run the following queries and show the questions they return: (1) programming, (2) pets, (3) college, (4) love, and (5) food.
  - Score each retrieved question for relevance using a three point scale as in Problem 3 (very relevant, somewhat relevant, not relevant)
  - compute NDCG for each query and report it.
- (5 points) In a 2-3 sentences, describe how well you think your IR system is doing. What kids of queries do you think it would work well on? What kinds of queries do you think it will perform poorly on? (Feel free to describe example queries if you want to test things!)

Finally, let's get a sense of how scalable our system is. Because we're not imposing a minimum frequency for including a word in our `CountVectorizer`, we're effectively indexing every wold. For this exercise, we have 100K questions, but what if we had 1M or 1B—how many terms would we be indexing? Let's estimate this by looking at how our index size (i.e., the number of words recognized by the vectorizer) grows relative to our corpus size.

- (5 points) Re-run your `CountVectorizer` code with 1K, 5K, 10K, 50K, and 100K questions and for each plot how many terms appear in the `vocabulary_` field for the vectorizer. In general, we recommend using Seaborn for all plotting.[2] Write 2-3 on why you think this approach will or won't scale as we get more documents and justify you answer.

Academic Honesty Policy

Unless otherwise specified in an assignment all submitted work must be your own, original work. Any excerpts, statements, or phrases from the work of others must be clearly identified as a quotation, and a proper citation provided. Re-using worked examples of significant portions of your own code from another class or code from other people from places like online tutorials, Kaggle examples, or Stack Overflow will be treated as plagiarism. If you are in doubt, please contact one of the instructors to check.

Any violation of the University's policies on Academic and Professional Integrity may result in serious penalties, which might range from failing an assignment, to failing a course, to being expelled from the program. Violations of academic and professional integrity will be reported to Student Affairs. Consequences impacting assignment or course grades are determined by the faculty instructor; additional sanctions may be imposed.

[2] If you're feeling particularly curious, try repeating this process with random samples of each number of questions (e.g., a random 1K questions) and show the mean and standard error.