Lisa Yan and Jerry Cain
CS109

Problem Set #6
November 6, 2020

Problem Set #6
Due: 1:00pm on Monday, November 16th

With problems by Mehran Sahami, Chris Piech, Tim Gianitsos, Alex Tsun and Anand Shankar

- Submit on Gradescope by 1:00pm Pacific on Monday, November 16th, for a small, "on-time" bonus.
- All students have a pre-approved extension, or "grace period" that extends until Wednesday 1:00pm Pacific, when they can submit with no penalty. **The grace period expires on 1:00 Pacific on Wednesday, November 18th**, after which we cannot accept further late submissions.
- **Collaboration policy:** You are encouraged to discuss problem-solving strategies with each other as well as the course staff, but you must write up your own solutions and submit individual work. Please cite any collaboration at the top of your submission.
- **Tagging written problems**: When you submit your written PDF on Gradescope you must tag your PDF, meaning that you must assign pages of your PDF as answers to particular questions so that we can properly grade your submission. For problem sets, we are deducting **2 points** for any submissions that do not have all questions tagged.

Written Problems

1. Consider the Exponential distribution. It is your friend . . . really. Specifically, consider a sample of I.I.D. exponential random variables $X_1, X_2, \ldots, X_n$, where each $X_i \sim \text{Exp}(\lambda)$. Derive the maximum likelihood estimate for the parameter λ in the Exponential distribution.

2. You are designing a randomized algorithm that delivers one of two new drugs to patients who come to your clinic—each patient can only receive one of the drugs. Initially you know nothing about the effectiveness of the two drugs. You are simultaneously trying to learn which drug is the best and, at the same time, cure the maximum number of people. To do so we will use the Thompson Sampling Algorithm.

   Thompson Sampling Algorithm: For *each* drug we maintain a Beta distribution to represent the drug's probability of being successful. Initially we assume that drug i has a probability of success: $\theta_i \sim \text{Beta}(1, 1)$.

   When choosing which drug to give to the next patient we **sample** a value from each Beta and select the drug with the largest **sampled** value. We administer the drug, observe if the patient was cured, and update the Beta that represents our belief about the probability of the drug being successful. Repeat for the next patient.

a. Say you try the first drug on 7 patients. It cures 5 patients and has no effect on 2. What is your belief about the drug's probability of success, θ_1? Your answer should be a Beta.

Method	Description
`V = sampleBeta(a, b)`	Returns a real number value in the range [0, 1] with probability defined by a PDF of a Beta with parameters a and b.
`R = giveDrug(i)`	Gives drug i to the next patient. Returns `True` if the drug was successful in curing the patient or `False` if it was not. Throws an error if $i \notin \{1, 2\}$.
`I = argmax(list)`	Returns the index of the largest value in the list.

b. Write pseudocode to administer either of the two drugs to 100 patients using Thompson's Sampling Algorithm. Use functions from the table above. Your code should execute `giveDrug` 100 times.

c. After running Thompson Sampling Algorithm 100 times, you end up with the following Beta distributions:
$\theta_1 \sim \text{Beta}(11, 11)$,
$\theta_2 \sim \text{Beta}(76, 6)$.
For each drug, what is your maximum a posteriori (MAP) estimate of the probability of the drug's success?

3. Say you have a set of binary input features/variables $X_1, X_2, \ldots, X_m$ that can be used to make a prediction about a discrete binary output variable Y (i.e., each of the X_i as well as Y can only take on the values 0 or 1). Say that the first k input variables $X_1, X_2, \ldots, X_k$ are actually all identical copies of each other, so that when one has the value 0 or 1, they all do. Explain informally, but precisely, why this may be problematic for the model learned by the Naïve Bayes classifier.

Coding Problems

Programming Problems

For the following problems, you will be implementing two learning algorithms: the Naïve Bayes classifier and Logistic Regression. You must use the starter code provided in `naive_bayes.py`, `logistic_regression.py`, and `questions.py`.

Submission

Submit only the files `naive_bayes.py`, `logistic_regression.py`, and `questions.py` to Gradescope under "PSet 6 - Coding". Do not submit any other files. We will only grade your work in the three aforementioned files.

Implementation details

- You will be given starter code with clearly marked indicators on where you should write your code. Do NOT modify any code outside these markers.
- You do not need to handle any of the data loading; the data will be loaded for you into numpy arrays and passed to the functions (see the starter code for more details).
- You can write your algorithms to only deal with binary train/test features/labels. Your code should, however, be general enough to work for any positive number of input features or data instances i.e. the matrix dimensions can change, but the contents of every matrix cell can only be 0 or 1.

Datasets

You will be running your learning algorithms on four datasets (each of which has a respective training data file and testing data file). **See the respective README files for more details.**

Simple (`simple-train.txt`, `simple-test.txt`)

This is a simple dataset provided primarily to help you determine that your code is working correctly. There are two input features, and the output class value is determined by the value of the first feature (i.e., $y = x_1$). The training dataset and testing dataset are identical, each containing four data vectors. Both your Naïve Bayes classifier and Logistic Regression implementations should be able to classify all instances in the simple testing dataset with 100% accuracy after training on the simple training set.

Heart tomography diagnosis (`heart-train.txt`, `heart-test.txt`)

This dataset contains data related to diagnosing heart abnormalities based on tomography (X-ray) information. Each input vector represents data extracted from the X-ray of one patient's heart. There are 22 binary input features. The output class value represents the diagnosis of the patient's heart (normal or abnormal, encoded in binary). The training dataset contains 80 data vectors, and the testing dataset contains 187 data vectors.

(Thanks to Lukasz Kurgan and Krzysztof Cios for providing this data to the UC Irvine Machine Learning Repository.)

Genetic ancestry (`ancestry-train.txt`, `ancestry-test.txt`)

This dataset contains DNA nucleotide readings from 467 individuals. Each input vector represents locations in the human genome and whether the individual's nucleotide at given locations matches the human reference genome. The output class value represents the super population of the user.

(Thanks to Jim Notwell and Gill Bejerano for providing this dataset.)

Netflix dataset (`netflix-train.txt`, `netflix-test.txt`)

This dataset contains real user ratings from Netflix. Each input vector represents ratings by a single user for the 30 most commonly rated movies (1 = rating of 5). The output class value represents whether the user rated the target movie (*Love Actually*) as a 5.

(Thanks to Reed Hastings for providing this dataset, with processing by Chris Piech.)

Training and testing your machine learning models

You will implement two machine learning models. To evaluate their correctness, we have exposed a few correct answers as a sanity check, which you can run on your own computer by following the instructions in the README file. Ensure that you have the correct version of Python as listed in the README file.

We do not provide the correct answers for the remainder of the tests, but you can verify if your answer is correct by submitting to Gradescope and running the autograder. You may submit on Gradescope as many times as you like, and we'll only grade your last submission. That being said, we recommend running and debugging your code locally most of the time and only submitting to Gradescope once you're ready to see if your answers are correct. We advise against solely running your code in Gradescope.

You do not need to provide any arguments to any of the functions, as this is done for you in the starter code. Lastly, note that you ***must*** read the instructions in the comments in the starter code and abide by them to receive full credit.

4. **[Coding]** Implement a Naïve Bayes classifier. Detailed instructions are provided in the comments of the starter code file, `naive_bayes.py` and the README file.

   a. Implement the function `fit` in `naive_bayes.py`.

   b. Implement the function `predict` in `naive_bayes.py`.

5. **[Coding]** Once you have implemented your Naïve Bayes classifier, it's time to analyze! Implement the functions prefixed with `question_nb` in `questions.py`. Refer to the README file for how to run this file, and carefully read the instructions in the comments in the starter code.

6. **[Coding]** Implement a Logistic Regression classifier. Specifically, you should implement the gradient ascent algorithm described in class. Detailed instructions are provided in the comments of the starter code and in the README file.

   a. Implement the function `fit` in `logistic_regression.py`.

   b. Implement the function `predict` in `logistic_regression.py`.

7. **[Coding]** Once you have implemented your Logistic Regression classifier, it's time to analyze! In particular, *precision* and *recall* are two measures of performance that are often more informative than simple classification accuracy, which you will explore below.

   Implement the functions prefixed with `question_lr` in `questions.py`. Refer to the README file for how to run this file, and carefully read the instructions in the comments in the starter code.