David Varodayan
CS109

Problem Set #6
February 28, 2020

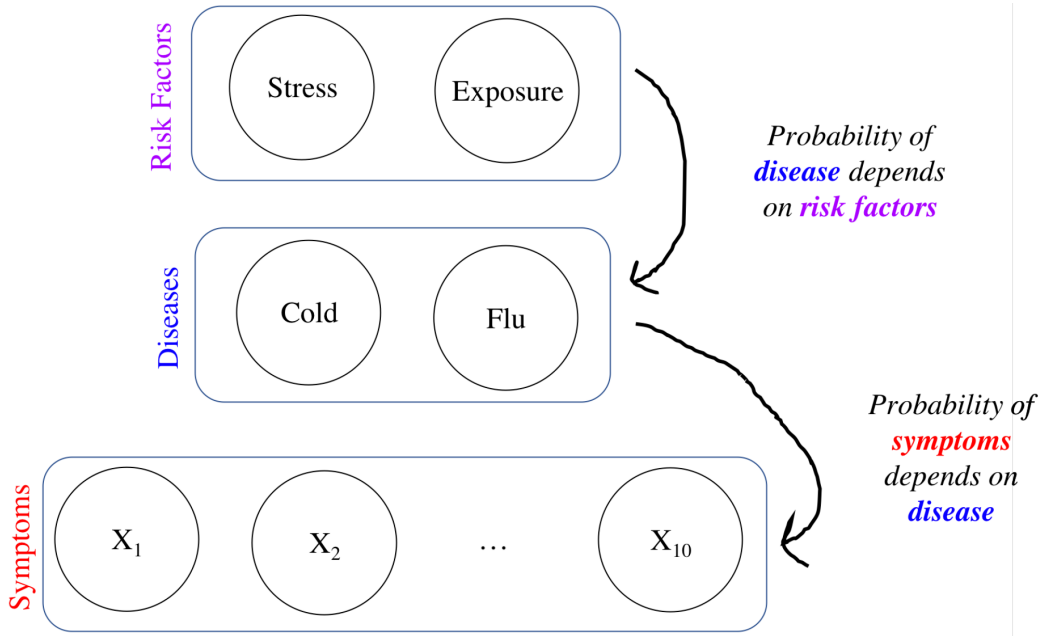
Problem Set #6
Due: 1:00pm on Wednesday, March 11

Note: The last day this assignment will be accepted (late) is Friday, March 13.

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

Written Problems

1. We are writing a WebMD program that is slightly larger than the one we worked through in class. In this program we predict whether a user has a flu ($F = 1$) or cold ($C = 1$) based on knowing any subset of 10 potential binary symptoms (e.g., headache, sniffles, fatigue, cough, etc) and a subset of binary risk factors (exposure, stress).



- We know the prior probability for Stress is 0.5 and Exposure is 0.1.
- The functions `probCold(s, e)` and `probFlu(s, e)` return the probability that a patient has a cold or flu, given the state of the risk factors stress (`s`) and exposure (`e`).
- The function `probSymptom(i, f, c)` which returns the probability that the `i`th symptom (X_i) takes on value 1, given the state of cold (`c`) and flu (`f`): $P(X_i = 1 | F = f, C = c)$.

We would like to write pseudocode to calculate the probability of flu *conditioned on observing* that the patient has had exposure to a sick friend and that they are experiencing Symptom 2 (sore throat). In terms of random variables $P(\text{Flu} = 1 \mid \text{Exposure} = 1, X_2 = 1)$:

```
def inferProbFlu() # P(Flu = 1 | Exposure = 1 and X_2 = 1)
```

Write pseudocode that calculates `inferProbFlu()` using **Rejection Sampling**.

2. 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.

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.

Continued on the next page. . .

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` and `logistic_regression.py`.

Submission

Submit only the TWO files `naive_bayes.py` and `logistic_regression.py` to Gradescope under "PSet 6 - Coding". Do not submit any other files. We will only grade your work in the two 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 driver 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. Implement a Naïve Bayes classifier. Detailed instructions are provided in the comments of the starter code.

    a. **[Coding]** Implement the function `fit` in `naive_bayes.py`.

    b. **[Coding]** Implement the function `predict` in `naive_bayes.py`.

5. 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.

    a. **[Coding]** Implement the function `fit` in `logistic_regression.py`.

    b. **[Coding]** Implement the function `predict` in `logistic_regression.py`.