IBM Watson

1. Choose preferred working location
2. video: why Watson
3. video: you skills/experiences might be additions to Watson team?
4. video:

Please read through the problem statement and describe your approach and thou ght process for how you would address creating this solution.

Problem Space: Recognizing “product names” within a very large Twitter text fe ed.
Problem Definition: Build a system that takes a twitter feed as input, and output each of the instances of “product names” mentioned within it.

1. What general approach would you use? How many approaches can you think of?

2. How do you plan to collect the data and evaluate the system before you actuall y create it? What metrics would you utilize?

3. What machine learning algorithms would you use for training?
If you used logistic regression how would you extract features? How would you do feature selection?

---

5. Programming Challenge Description:
A group of NLP researchers have developed a universal language translation sys tem that can translate English to any foreign languages.
Unfortunately, an error was introduced by a junior developer of the team, which resulted the system read input English sentences from right to left.
Because this is the opposite of how English is written, the team asks you to write a program that fixes this error.
In addition to the direction, you also have to modify the case of some letters such that only the leftmost character of a space-
delimited word is in uppercase. The other letters in the word should be in lowerc

ase.
Input:
Your program should read lines of text from standard input. Each line contains a single sentence.
Output:
Print to standard output each modified sentence, one sentence per line.

---

6. Programming Challenge Description:

In NLP and machine learning, calculating probabilities are pervasive. When probabilities are large values, like on the order of 1E-1, calculations will be pretty accurate. However, when probabilities are small, e.g. on the order of 1E-1000, it is almost impossible to manipulate those values directly without transformation.

Please consider how we would solve a problem of variable marginalization: summing over all possible probabilities for a variable, and those probabilities are too small to be handled directly. Log values have to be used instead of original probabilities. Formally, we have a list of log probabilities of small numbers [logP1, logP2, ...], the algorithm is to calculate the log (P) = log ([P1+ P2 + ...]). One of the technique is to find a max among inputs and use that to improve the calculation accuracy. The following is an example code segment for how to do this. With the the time that remains, please update the code segment to provide the desired outputs. Candidates will further differentiate themselves for updates that will check all values in logP are <=0.

Input:

Input is a list of n valid log probabilities delimited by white space on single line
logP1 logP2 .... logPn

Output:

The log of sum of those probabilitites logP = log(P1 + P2 + ... + Pn)

```
public class Main {
  public static void main(String[] args) throws IOException {

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String s=in.readLine();
    double numbers[] = convert(s.split(" "));

    System.out.printf("%.13f", getLogSum(numbers));

}

  public static double[] convert(String[] strings){
      double[] numbers = new double[strings.length];
      for (int i = 0; i < strings.length; i++)
        numbers[i] = Double.parseDouble(strings[i]);
        return numbers;
  }

  public static double getLogSum(double [] logP){

                if(logP.length == 0){
                        return Double.NaN;
                }

                double max = Double.NEGATIVE_INFINITY;
                for(double d : logP){
                        if(d > max){
                                max = d;
                        }
                }

                double sum = 0.;
                for(double d : logP){
                        sum += Math.exp(d - max);
                }
        return max * Math.log(sum);
    }
}
```

---

7. Programming Challenge Description:

An *n*-gram is a contiguous sequence of *n* words appearing in a given text. For a text consisting of multiple sentences, *n*-grams are computed sentence by sentence. For instance, suppose an example text consisting of 2 sentences: *"IBM created Watson. It won Jeopardy."* For *n*=2, valid *n*-grams include *"IBM created", "created Watson", "It won",* and *"won Jeopardy"*, but not *"Watson It"* because *"Watson"* and *"It"* are not in the same sentence. If the length of a sentence is less than the value of *n*, no *n*-grams can be computed from the sentence.

When used as a language model, *n*-grams estimate conditional probability of each word given prior context of *n*-1 words. *N*-gram language models can be used, among others, to compute the likelihood of unseen sentences.

For instance, the probability of a sentence *"I want Chinese food"* can be computed as *P(want | I) x P(Chinese | want) x P(food | Chinese)* using bigrams (*n* = 2) or *P(Chinese | I want) x P(food | want Chinese)* using trigrams (*n* = 3).

In this task, you need to write a code that computes the probability of an unseen sentence based on *n*-gram probability distribution. The probability distribution of *n*-grams should be derived from a training corpus using Add-one smoothing technique. Given a string of *N* words wnN1n1wnN1...wn1wn*wn*−*N*+1*n*−1=*wn*−*N*+1,...,*wn*−1,*wn*. Add-one smoothing defines the probability of a word wn*wn* given prior *n*-1 words as follows:

Pwn|wnN1n1racCwnN1n1wn1CwnN1n1|V|*P*(*wn*|*wn*−*N*+1*n*−1)=*racC*(*wn*−*N*+1*n*−1*wn*)+1*C*(*wn*−*N*+1*n*−1)+|*V*|

where

(1) CwnN1n1wn*C*(*wn*−*N*+1*n*−1*wn*) is the count of the *n*-gram wnN1...wn1wn*wn* −*N*+1,...,*wn*−1,*wn* in the training corpus,

(2) CwnN1n1*C*(*wn*−*N*+1*n*−1) is the count of all n-grams in training corpus that has a substring wnN1...wn1*wn*−*N*+1,...,*wn*−1 as prefix, and

(3) V = the Union of {individual words appearing in training corpus} and {individual words appearing in unseen sentence}.

For simplicity, following assumptions are made:

- All characters are in lower case.
- There is no punctuation except for period (.) that marks end of sentence.
- Given a sentence, *s*, *P(s)* = 0 when *length(s)* < *n*.

Input:

Input is a single string consisting of three parts delimted by a comma(,). The first part is an integer that specifies the value of N. The second part contains a training corpus and the third part is an unseen sentence.

Output:

For each test case, output is the likelihood of the unseen sentence, which is a single floating-point number. Ouput the number only up to third digit after the decimal point.

```
import java.io.*;

public class Main {
  public static void main(String[] args) throws IOException {

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String s;
    while ((s = in.readLine()) != null) {
      System.out.println(s);
    }
  }
}
```

8.video

Please take approximately 5-
10 minutes to explain your approach to exercise "Likelihood of Unseen Sentence
". Include and support your rationale for how you approached the exercise.

9.video
Summarize how the Naive Bayes algorithm is derived; for an example in a binar
y classification problem of: “Classifying emails as Spam and non-
Spam”. Be sure to explain the independence assumptions in the derivation.