Chris Piech
CS 109

Section #9
Mar 9, 2022

# Section Solution 9: Final Section

Problem 2 by David Varodayan

1. **Bayesian Carbon Dating (34 points)**

   We are able to know the age of ancient artefacts using a process called carbon dating. This process involves a lot of uncertainty! Living things have a constant proportion of a molecule called C14 in them. When living things die those molecules start to decay. The time to decay in years, $T$, of a C14 molecule is distributed as an exponential. $T \sim \text{Exp}(\lambda = 1/8267)$.

   a. (5 points) Consider a single C14 molecule. What is the probability that it decays within 500 years?

   $$P(T \leq 500) = 1 - e^{-\frac{1}{8267} * 500} = 0.05868875306$$

   b. (8 points) C14 molecules decay independently. A particular sample started with 100 molecules. What is the probability that exactly 95 are left after 500 years? Let $p$ be your answer to part a.

   $X \sim \text{Bin}(n = 100, p = 0.0586)$
   P(X = 5) = scipy.stats.binom.pmf(5, 100, 0.05868)

   c. (6 points) Write pseudocode for a function `pr_measure_given_age(m, age)` which returns $P(M = \texttt{m} | A = \texttt{age})$, the probability that exactly `m` molecules are left out of the original 100 after exactly `age` number of years.

   Consider each of the 100 original molecules as an independent trials of a C14 molecule. We can model `pr_measure_given_age(m, age)` as a binomial with parameters 100 and probability of a single element left after some age.

```
def pr_measure_given_age(m, age):
    # Find the exponential probability
    p = 1 - np.exp(-(1/8267) * age)

    # Find the amount of successes after 100 trials.
    pr_m_given_age = scipy.stats.binom.pmf(100 - m, 100, p)

    return pr_m_given_age
```

d. (15 points) You observe a measurement of 95 C14 molecules in a sample. You assume that the sample originally had 100 C14 molecules when it died. Write pseudocode for a function `age_belief()` that returns a list of length 1000 where the value at index $i$ in the list stores $P(A = i|M = 95)$. Age is a discrete random variable which takes on whole numbers of years. $A = i$ is the event that the sample organism died $i$ years ago. You may use the function `pr_measure_given_age(m, age)` from part c. For your prior belief: you know that the sample ***must*** be between $A = 500$ and $A = 600$ inclusive and you assume that every year in that range is equally likely.

First apply Bayes rule

$$P(A = i|M = 95) = \frac{P(M = 95|A = i)P(A = i)}{P(M = 95)}$$

We will write psudo-code using this equation. A few notes,

i. $P(M = 95|A = i)$ = `pr_measure_given_age(95, i)`

ii. $P(A = i)$ = 1 / (100 + 1), with 101 because 500 to 600 **inclusive**.

iii. $P(M = 95)$ Can be disregarded in the code since it is only a normalization term (e.g, we normalize the returned array).

With that, here's the code:

```
def age_belief(m, age):
    # Initial array
    prob_list = np.zeros(1000)

    # Range over all possible ages, zero elsewhere.
    for age in range(500, 600 + 1):

            # (i) -- form c
            m_given_a = pr_m_given_age(95, age)

            # (ii) -- from assump
            prior_age = 1/101

            prob_list[idx] = m_given_a * prior_age

    # (iii) -- normalize out
    prob_list /= sum(prob_list)

    return prob_list
```

2. **Timing Attack**:

In this problem we are going to show you how to crack a password in linear time, by measuring how long the password check takes to execute (see code below). Assume that our server takes $T$ ms to execute any line in the code where $T \sim N(\mu = 5, \sigma^2 = 0.5)$ seconds. The amount of time taken to execute a line is always independent of other values of $T$.

```
# An insecure string comparison
def string_equals(guess, password):
    n_guess = len(guess)
    n_password = len(password)
    if n_guess != n_password:
        return False                    # 4 lines executed to get here
    for i in range(n_guess):
        if guess[i] != password[i]:
            return False                # 6 + 2i lines executed to get here
    return True                         # 5 + 2n lines executed to get here
```

On our site all passwords are length 5 through 10 (inclusive) and are composed of lower case letters only. A hacker is trying to crack the root password which is "gobayes" by carefully measuring how long we take to tell them that her guesses are incorrect.

a. What is the distribution of time that it takes our server to execute $k$ lines of code? Recall that each line independently takes $T \sim N(\mu = 5, \sigma^2 = 0.5)$ ms.

Let $Y$ be the amount of time to execute $k$ lines. $Y = \sum_{i=1}^{k} X_i$ where $X_i$ is the amount of time to execute line $i$. $X_i \sim N(\mu = 5, \sigma^2 = 0.5)$.

Since $Y$ is the sum of independent normals:

$$\begin{aligned} Y &\sim N(\mu = \sum_{i=1}^{k} 5, \sigma^2 = \sum_{i=1}^{k} 0.5) \\ &\sim N(\mu = 5k, \sigma^2 = 0.5k) \end{aligned}$$

b. First the hacker needs to find out the length of the password. What is the probability that the time taken to test a guess of correct length (server executes 6 lines) is longer than the time taken to test a guess of an incorrect length (server executes 4 lines)? Assume that the first letter of the guess does not match the first letter of the password. Hint: $P(A > B)$ is the same as $P(A - B > 0)$.

From last problem:
Time to run 6 lines of code $A \sim N(\mu = 30, \sigma^2 = 3)$
Time to run 4 lines of code $B \sim N(\mu = 20, \sigma^2 = 2)$

$$\begin{aligned} -B &\sim N(\mu = -20, \sigma^2 = 2) \\ A - B &\sim N(\mu = 10, \sigma^2 = 5) \\ P(A > B) &= P(A - B > 0) \\ &= 1 - F_{A-B}(0) \\ &= 1 - \Phi\left(\frac{0 - 10}{\sqrt{5}}\right) \\ &\approx 1.0 \end{aligned}$$

c. Now that our hacker knows the length of the password, to get the actual string she is going to try and figure out each letter one at a time, starting with the first letter. The hacker tries the string "aaaaaaa" and it takes 27s. Based on this timing, how much more probable is it that first character did not match (server executes 6 lines) than the first character did match (server executes 8 lines)? Assume that all letters in the alphabet are equally likely to be the first letter.

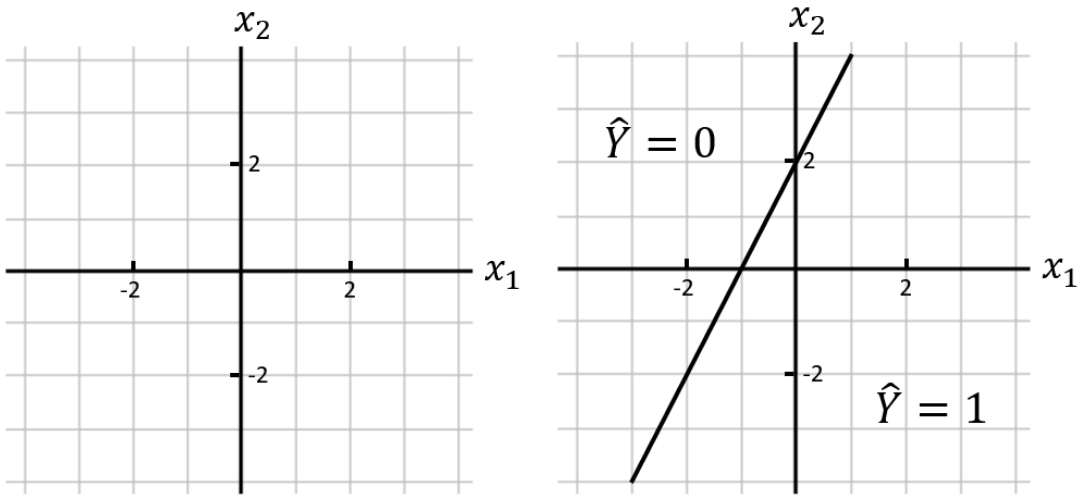
Let $M$ be the event that the first letter matched.

$$\begin{aligned}
\frac{P(M^C|T=27)}{P(M|T=27)} &= \frac{f(T=27|M^C)P(M^C)}{f(T=27|M)P(M)} \\
&= \frac{f(T=27|M^C)\frac{25}{26}}{f(T=27|M)\frac{1}{26}} \\
&= 25 \cdot \frac{f(T=27|M^C)}{f(T=27|M)} \\
&= 25 \cdot \frac{\frac{1}{\sqrt{6\pi}}e^{-\frac{(27-30)^2}{6}}}{\frac{1}{\sqrt{8\pi}}e^{-\frac{(27-40)^2}{8}}} \\
&= 25 \cdot \frac{\sqrt{8}}{\sqrt{6}} \cdot \frac{e^{-\frac{9}{6}}}{e^{-\frac{169}{8}}} \\
&\approx 9.6 \text{ million}
\end{aligned}$$

d. If it takes the hacker 6 guesses to find the length of the password, and 26 guesses per letter to crack the password string, how many attempts does she need to crack our password, "gobayes"? Yikes!

$7 \cdot 26 + 6 = 188$

3. **Logistic regression**




Suppose you have trained a logistic regression classifier that accepts as input a data point $(x_1, x_2)$ and predicts a class label $\hat{Y}$. The parameters of the model are $(\theta_0, \theta_1, \theta_2) = (2, 2, -1)$. On the axes, draw the decision boundary $\theta^T \mathbf{x} = 0$ and clearly mark which side of the boundary predicts $\hat{Y} = 0$ and which side predicts $\hat{Y} = 1$.

$\theta^T \mathbf{x}$ can be expanded as $2 + 2x_1 - x_2 = 0$ because $x_0 = 1$ by definition. The prediction is 1 when $\theta^T \mathbf{x} > 0$. For example, the origin $(x_1, x_2) = (0, 0)$ yields $\theta^T \mathbf{x} = 2$, which gives us the prediction $\hat{Y} = 1$.

See the graph above, to the right of the original.

4. **The Most Important Features**

Let's explore saliency, a measure of how important a feature is for classification. We define the saliency of the $i$th input feature for a given example $(\mathbf{x}, y)$ to be the absolute value of the partial derivative of the log likelihood of the sample prediction, with respect to that input feature $\left|\frac{\partial LL}{\partial x_i}\right|$. In the images below, we show both input images and the corresponding saliency of the input features (in this case, input features are pixels):

First consider a trained logistic regression classifier with weights $\theta$. Like the logistic regression classifier that you wrote in your homework it predicts binary class labels. In this question we allow the values of $\mathbf{x}$ to be real numbers, which doesn't change the algorithm (neither training nor testing).

a. What is the Log Likelihood of a single training example $(\mathbf{x}, y)$ for a logistic regression classifier?

$$LL(\theta) = y \cdot \log \sigma\Big(\theta^T \cdot \mathbf{x}\Big) + \Big(1 - y\Big)\log \Big[1 - \sigma\Big(\theta^T \cdot \mathbf{x}\Big)\Big]$$

b. Calculate is the saliency of a single feature $(x_i)$ in a training example $(\mathbf{x}, y)$.

We can calculate the saliency for a single feature as follows.

$$\begin{aligned}
LL(\theta) &= y \log z + \Big(1 - y\Big) \log \Big(1 - z\Big) && \text{where } z = \sigma\Big(\theta^T \cdot \mathbf{x}\Big) \\
\frac{\partial LL}{\partial x_i} &= \frac{\partial LL}{\partial z} \cdot \frac{\partial z}{\partial x_i} && \text{chain rule} \\
&= \Big(\frac{y}{z} - \frac{1-y}{1-z}\Big) \cdot \Big(z(1-z)\theta_i\Big) && \text{partial derivatives} \\
\text{saliency} &= \Big| \Big(\frac{y}{z} - \frac{1-y}{1-z}\Big) z(1-z)\theta_i \Big|
\end{aligned}$$

Show that the ratio of saliency for features $i$ and $j$ is the ratio of the absolute value of their weights $\frac{|\theta_i|}{|\theta_j|}$.

We can take the ratio as follows using our expression above.

$$\text{saliency for feature } i, S_i = \Big| \Big(\frac{y}{z} - \frac{1-y}{1-z}\Big) z(1-z)\theta_i \Big| \text{, and same for } S_j$$

$$\frac{S_i}{S_j} = \frac{\Big| \Big(\frac{y}{z} - \frac{1-y}{1-z}\Big) z(1-z)\theta_i \Big|}{\Big| \Big(\frac{y}{z} - \frac{1-y}{1-z}\Big) z(1-z)\theta_i \Big|} = \frac{S_i}{S_j} = \frac{\Big| \theta_i \Big|}{\Big| \theta_j \Big|} \text{ by elimination}$$

[1]: Modeling menstrual cycle length using a mixture distribution.
https://academic.oup.com/biostatistics/article/7/1/100/243078

[2]: Weibull Distribution.
https://en.wikipedia.org/wiki/Weibull_distribution