Homework #2

March 6, 2022

[38]:
```
# Initialize Otter
import otter
grader = otter.Notebook("Homework #2.ipynb")
```

[39]:
```
import numpy as np
from sklearn.linear_model import SGDClassifier
```

1 CS 639 - Foundations of Data Science

2 Homework #2

The goal of this homework is to provide practice examples for applications of concentration inequalities in determining confidence intervals and for performing simple hypothesis tests.

2.1 Problem 1: Empirical Estimate of Variance

In the discussion session (and Eric's notes) you saw how to estimate variance from bounded data samples when the true mean is known. In this question, you are asked to estimate the varaince in the same setting, but *without knowing the true mean.* The precise problem statement is:

Suppose you are given access to i.i.d. samples $X_1, X_2, \ldots, X_n$ that are guaranteed to come from a probability distribution such that $X_i \in [a, b]$ surely, for all $i \in \{1, \ldots, n\}$, where $a < b$ are real numbers.

Let $\bar{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i$ be the empirical estimate of the mean and $\bar{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{\mu})^2$ be the empirical estimate of the variance. Given a confidence parameter $\delta \in (0, 1)$ and n (the number of samples), provide a symmetric confidence interval for your variance estimate (that is, find $\epsilon > 0$ such that your variance estimate is within $\pm\epsilon$ of the true variance with probability at least $1 - \delta$). Justify your answer.

Code a function in Python that takes a, b, n, δ as input and returns ϵ from your answer above.

Hint: Triangle inequality tells us that for any $p, q, r \in \mathbb{R} : |p - q| \leq |p - r| + |r - q|$. Use triangle inequality and the union bound.

Let, empirical variance $= \hat{\sigma}^2$ and empirical estimate of variance $= \bar{\sigma}^2$ and population variance $= \sigma^2$

$\mathbb{P}(|\bar{\sigma}^2 - \sigma^2| \geq k) \leq \mathbb{P}(|\bar{\sigma}^2 - \hat{\sigma}^2| + |\hat{\sigma}^2 - \sigma^2| \geq k) = \mathbb{P}(|\bar{\sigma}^2 - \hat{\sigma}^2| \geq k/2) + \mathbb{P}(|\hat{\sigma}^2 - \sigma^2| \geq k/2)$

$\mathbb{P}(|\hat{\sigma}^2 - \sigma^2| \geq k/2) \leq 2e^{\frac{-nk^2}{2(b-a)^4}}$ (From lecture, applying hoeffding bound)

$\bar{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{\mu})^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \mu + \mu - \bar{\mu})^2 = \hat{\sigma}^2 - (\mu - \bar{\mu})^2$ or, $\hat{\sigma}^2 - \bar{\sigma}^2 = (\mu - \bar{\mu})^2$

$==>$ $\mathbb{P}(|\bar{\sigma}^2 - \hat{\sigma}^2| \geq k/2) = \mathbb{P}(|(\bar{\mu} - \mu)^2| \geq k/2) = \mathbb{P}(|\bar{\mu} - \mu| \geq \sqrt{k/2}) \leq 2e^{\frac{-2nk}{2}} = 2e^{-nk}$ (Hoeffding bound)

$\mathbb{P}(|\bar{\sigma}^2 - \sigma^2| \geq k) \leq \mathbb{P}(|\bar{\sigma}^2 - \hat{\sigma}^2| \geq k/2) + \mathbb{P}(|\hat{\sigma}^2 - \sigma^2| \geq k/2) \leq 2e^{-nk} + 2e^{\frac{-nk^2}{2(b-a)^4}}$

2.2 Problem 2: DNA and P-Values

A strand of a DNA molecule consists of a sequence of four chemical bases: adenine (A), guanine (G), cytosine (C), and thymine (T). In a usual strand of human DNA, adenine and thymine comprise 30% of the chemical bases (each), while guanine and cytosine comprise the remaining 20% (each). Suppose you are presented with a sequence of N letters A, G, C, and T that are claimed to come from a human DNA. Suppose that there are n occurences of C in this sequence. You are asked to determine whether the sequence you see is surprising.

1. Write down what are your null hypothesis and the alternative hypothesis.
2. Write down what test statistic you would use.
3. Write down the formula for the p-value for this specific problem.
4. Assuming $N = 30$ and $n = 14$, compute the p-value (you can do this in Python; you are allowed to use scipy.stats). Now use Chebyshev inequality and Hoeffding bound to bound the p-value.
5. Using the computed bounds on the p-value (including the true p-value), discuss when you would reject the null hypothesis.

1. H_0 : The sequence is not surprizing, H_1 : The sequence is surprizing
2. Test statisctic, S: number of occurrence of C in the sequence (=n)
3. For two-sided test, p-value = $\mathbb{P}[S \geq n|H_0] + \mathbb{P}[S \leq n_1|H_0]$, Here $n_1 = \mathbb{E}[X] - (n - \mathbb{E}[X])$ which is the left point that is equally distant from the expected value as the observed value, n (n_1 in this case is negative. so $\mathbb{P}[S \leq n_1|H_0]$ adds nothings in this case for N=30, n = 14)
4. The p value computed is 0.0009018. N = 30, n = 14, p = 0.20

Let, S_n be the number of occurrence of C. $S_n = X_1 + X_2 + X_3 .... + X_{30}$, where $X_i = 1$ if ith sequence is C ans 0 otherwise. $\mathbb{E}[S_n]$ = Np = 30 * 0.20 = 6 and $var[S_n]$ = Npq = 30 * 0.20 * 0.80 = 4.8

Chebyshev Inequality: $\mathbb{P}[S_n >= 14] = \mathbb{P}[|S_n - 6| \geq 8] \leq 4.8/64 = 0.075$

Hoeffding bound: $\mathbb{P}[S_n >= 14] = \mathbb{P}[|\frac{S_n}{30} - \frac{6}{30}| \geq \frac{8}{30}] \leq 2e^{\frac{-2nt^2}{(b-a)^2}} = 2e^{\frac{-64}{15}} = 0.028$, Here $[a, b] = [0, 1]$

5. Actual P value, P_ac = 0.00090, P_chebyshev $\leq$ 0.075, P_hoeffding $\leq$ 0.028.

if $\alpha = 0.05$, we will reject H_0 for actual p value and P_hoeffding. But we will fail to reject H_0 for P_chebyshev.

if $\alpha = 0.01$, we will reject H_0 for actual p value. But we will fail to reject H_0 for P_chebyshev and P_hoeffding.

[40]:
```
import numpy as np
from scipy.stats import binom
from scipy import stats

#stats.binom_test(14, n=30, p=0.2, alternative='greater')
stats.binom_test(14, n=30, p=0.2, alternative='two-sided')
#stats.binom_test(14, n=30, p=0.2, alternative='less')
#binom.pmf(0, 30, .2)
```

[40]:
```
0.0009018693287275339
```

2.3 Problem 3: Detecting Cancer from DNA

Exact Sciences is a molecular diagnostics company specializing in the detection of early stage cancers based in Madison. Suppose you work for Exact Sciences and are tasked to develop a new diagnostic test for identifying whether a certain type of cancer is present in a patient based on the patient's DNA features. In particular, given a patient's DNA data encoded in a vector x in $\{-1, +1\}^d$ where $d = 100$, you want to develop a test to classify whether the patient has cancer or not, outputting -1 when the patient does not have cancer and $+1$ when the patient has cancer.

2.3.1 Cancer Test Development Phase (Training Phase)

You have n patients' DNA features in a CSV file named `training-features.csv`. After conducting a 5-year long experiement, you also know whether the corresponding patients have developed cancer or not, and the labels are saved in another CSV file named `training-labels.csv` which will be what your diagnostic test aims to output.

We provide an example code for training linear classifers (link to documentation).

```
np.random.seed(0)  # Fixing a random seed

from sklearn.linear_model import SGDClassifier
X = [[-1, +1], [+1, -1], [+1, +1], [-1, -1]]  # Training features
Y = [-1, 1, 1, -1]  # Training labels

# Define the classifer to train
clf = SGDClassifier(max_iter=1000, tol=1e-3)

# Train the classifer
clf.fit(X, Y)
```

1. Write a function that returns a trained linear classifer object using the training data provided. You should use `SGDClassifier` from `sklearn.linear_model` and train it with max number of iterations `max_ter = 1000` and tolerance `tol = 1e-3`. You should also initiate random seed to 0 by calling `np.random.seed(0)` before training your classifer.

[41]:
```
# Run this cell to read training data into notebook.
# Please put the csv files in the same directory as this notebook.
```

```
training_features = np.genfromtxt("./training-features.csv", delimiter=",")
training_labels = np.genfromtxt("./training-labels.csv", delimiter=",")
```

[42]:
```
def train_classifier(training_features, training_labels):
    """
    Parameters
    ----------
    training_features: numpy.array
        Features of the training dataset. Shape = (n, d)
    training_labels: numpy.array
        Labels of the training dataset. Shape = (n,)

    Returns
    -------
    SGDClassifier
        A trained classifier.
    """
    np.random.seed(0)
    clf = SGDClassifier(max_iter=1000, tol=1e-4)
    clf.fit(training_features, training_labels)
    return clf
    ...
```

[43]:
```
grader.check("p3-1")
```

[43]:
```
p3-1 results: All test cases passed!
```

2. How large does n need to be so that with confidence 95% you can guarantee that

$$\mathbb{P}_{(x,y)\sim\mathcal{D}}[h(x) \neq y] \leq \epsilon$$

for $\epsilon = 1e-3$, where h is a linear classifer (a linear treshold function as in Lecture 4)?

$\mathbb{P}[\exists f \epsilon Cst | \hat{L}(f) - L(f)| \geq \epsilon/2] \leq 2^{cd\ln(d)} 2e^{\frac{-n\epsilon^2}{2}}$

For 95% confidence, $\delta = 0.05, \epsilon = 1e-3$, $n \geq \frac{2(cd\ln d\ln 2 + \ln(2/\delta))}{\epsilon^2} = \frac{2(100\ln 100\ln 2 + \ln(2/0.05))}{(1e-3)^2} =$ 645789905, ($1e-3 = 10\hat{}(-3)$)

3. Now suppose that you cannot recruit as many patients as in part 2, but you have n patients, where n is fixed. How small can you make ϵ? Make this value specific for $n = 8000$ (the number of examples in the training data set).

$\epsilon^2 \geq \frac{2(cd\ln d\ln 2 + \ln 2/\delta)}{n} \approx \frac{2(d\ln d\ln 2 + \ln(2/\delta))}{n}$

the range of delta for $\ln(2/\delta)$ can be between 0(not exactly, but very close to) and 1.

if n = 8000, length of vector d = 100, we can minimize ϵ by making $\delta = 1$ or $\ln(2/\delta) = 0.693$.

So, we get, $\epsilon = 0.2827$

4. Given that $d = 100$, discuss whether it would be possible to train a linear classifier with zero error (in terms of the population loss) in your lifetime.

If we want zero error, we need a linear function that minimizes the loss, ϵ to 0. $|\hat{L}(f) - L(f)| = \epsilon = 0$. In that case, $n \geq \frac{2(cd \ln d \ln 2 + \ln(2/\delta))}{\epsilon^2}$ indicates that if $\epsilon = 0$, n needs to be as large as infinity, which should not be possible I guess.

2.3.2 Cancer Test Trial Phase (Testing Phase)

After we have developed the diagnostic test for detecting cancer in patients, we typically evaluate how well the test performs in practice by going through a trial phase which involves using the test on real patients. A patient comes in and after conducting examinations to retrieve the patient's DNA features, you run the cancer diagnostic test to make a prediction of whether cancer has been detected.

In this trial phase, we have N patients and their DNA features are recorded in a CSV file named `testing-features.csv`. Assume we also get the true positive/negative cancer results via a more expensive and invasive diagnostic test, and the results are located in a CSV named `testing-labels.scv`.

5. Write a function that makes predictions for the patients in the trial phase using the linear classifier you trained, then compute the following quantities for our cancer diagnostic test: 1) Specificity, 2) Sensitivity and 3) Empirical Misclassification Error. Your code should return a tuple of the three quantities in the same ordering.

Hint: You can call `clf.predict([x])` to make a prediction for input feature `[x]` using your linear classifier `clf`.

[44]:
```
# Run this cell to load the classifier into variable diagnostic and read␣
↪testing data into notebook.
# Please put the csv files in the same directory as this notebook.

diagnostic = train_classifier(training_features, training_labels)  # Your␣
↪linear classifier

testing_features = np.genfromtxt("./testing-features.csv", delimiter=",")
testing_labels = np.genfromtxt("./testing-labels.csv", delimiter=",")
```

[45]:
```
def compute_performance_quantities(diagnostic, testing_features,␣
↪testing_labels):
    """
    Parameters
    ----------
    diagnostic: SGDClassifier
        A trained SGDClassifier linear classifier.
    testing_features: numpy.array
        Features of the testing dataset. Shape = (N, d)
    testing_labels: numpy.array
        Labels of the testing dataset. Shape = (N,)

    Returns
    -------
```

```
    (float, float, float)
        A tuple of specificity, sensitivity, miss.
    """
    X = testing_features
    y_actual = testing_labels
    y_hat = diagnostic.predict(X)
    #print(y_hat)
    TP = 0
    FP = 0
    TN = 0
    FN = 0
    miss = 0
    for i in range(len(y_hat)):
        if y_actual[i]==y_hat[i] and y_hat[i]==1:
           TP += 1
        if y_hat[i]==1 and y_actual[i]!=y_hat[i]:
           FP += 1
        if y_actual[i]==y_hat[i] and y_hat[i]==-1:
           TN += 1
        if y_hat[i]==-1 and y_actual[i]!=y_hat[i]:
           FN += 1
        if y_hat[i] != y_actual[i]:
            miss += 1
    #print(TP,FP,FN,TN,miss)
    spec = TN / (TN + FP)
    sens = TP/(TP+FN)

    #print(spec,sens,miss/ X.shape[0])
    return (spec,sens,miss/X.shape[0])
    ...
```

[46]:
```
grader.check("p3-5")
```

[46]:
```
p3-5 results: All test cases passed!
```

6. Compute the specificity, sensitivity, and misclassification error using the above function you wrote for our diagnostic test. How does this compare to ϵ you computed in Part 3? Output $\epsilon_{\text{miss}}/\epsilon$, where ϵ_{miss} is the empirical misclassification error computed by your code, and discuss the result.

specificity = (TN/(TN+FP)) = 253/253 = 1 sensitivity = TP/(TP+FN) = 245/(245+2) = 0.991902834 misclassification error = miss/length = 2 / 500 = 0.004 $\epsilon_{miss} = 0.004$ and $\epsilon = 0.2827$. $\epsilon_{miss}/\epsilon = 0.01414$ It indicates that the calculated ϵ is many times higher than the empirical ϵ_{miss}. We used Hoeffding bound to bound the n. And the Hoeffding bound is comparatively loose than actual. Therefore, we are getting smaller empirical ϵ_{miss} than the ϵ from our calculation.

To double-check your work, the cell below will rerun all of the autograder tests.

[49]:
```
grader.check_all()
```

[49]:
```
p3-1 results: All test cases passed!

p3-5 results: All test cases passed!
```

2.4 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

Please download the zip file after running the cell below, then upload the zip file to GradeScope for submission.

[50]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export(pdf=False)
```

```
<IPython.core.display.HTML object>
```