CS 412: Fall'23
Introduction To Data Mining

Assignment 3

(Due Wednesday, November 08, 11:59 pm)

- The homework is due at 11:59 pm on the due date. We will be using Gradescope for the homework assignments. You should join Gradescope using the entry code shared on Aug 3. Please do NOT email a copy of your solution. Contact the TAs if you are having technical difficulties in submitting the assignment. We will NOT accept late submissions!

- Please use Slack or Canvas first if you have questions about the homework. You can also come to our (zoom) office hours and/or send us e-mails. If you are sending us emails with questions on the homework, please start subject with "CS 412 Fall'23: " and send the email to *all of us* (Arindam, Ruby, Hyunisk, Rohan, Kowshika, Sayar) for faster response.

- Please write your code entirely by yourself.

  Programming Assignment Instructions

  – All programming needs to be in Python 3.

  – The homework will be graded using Gradescope. You will be able to submit your code as many times as you want. There are no post deadline tests. The grade appearing on Gradescope right after your submission will be your final grade for this assignment.

  – Two python files named `homework3_q1.py`, `homework3_q2.py` containing starter code are available on Canvas.

  – Do NOT add any additional `import` lines to your code. Required libraries have already been included in the starter code.

  – For submitting on Gradescope, you would need to upload both the files `homework3_q1.py` and `homework3_q2.py`.

1. (50 points) The problem will focus on developing code for Bayes Theorem applied to estimating the type of candy bag we have based on candies drawn from the bag, following the example discussed in class.

   Suppose there are five types of bags of candies:

   - π_1 fraction are h_1: p_1 fraction "cherry" candies, $(1 - p_1)$ fraction "lime" candies
   - π_2 fraction are h_2: p_2 fraction "cherry" candies, $(1 - p_2)$ fraction "lime" candies
   - π_3 fraction are h_3: p_3 fraction "cherry" candies, $(1 - p_3)$ fraction "lime" candies
   - π_4 fraction are h_4: p_4 fraction "cherry" candies, $(1 - p_4)$ fraction "lime" candies
   - π_5 fraction are h_5: p_5 fraction "cherry" candies, $(1 - p_5)$ fraction "lime" candies

   For the specific example discussed in class:

$$
\begin{array}{rrrrr}
\pi_1 = 0.1 \ , & \pi_2 = 0.2 \ , & \pi_3 = 0.4 \ , & \pi_4 = 0.2 \ , & \pi_5 = 0.1 \ , \\
p_1 = 1 \ , & p_2 = 0.75 \ , & p_3 = 0.5 \ , & p_4 = 0.25 \ , & p_5 = 0 \ .
\end{array}
$$

   A bag is given to us, but we do not know which type of bag $h \in \{h_1, h_2, h_3, h_4, h_5\}$ it is. We will be drawing a sequence of candies $c_1, c_2, \ldots, c_i \in \{\text{"cherry"}, \text{"lime"}\}$ from the given bag and, using Bayes rule, maintain posterior probabilities of the type of bag conditioned on the candies which have been drawn, i.e.,

$$
\begin{array}{lrl}
\text{After drawing } c_1 : & p(\pi_h | c_1) \ , & h = 1, \ldots, 5 \\
\text{After drawing } c_1, c_2 : & p(\pi_h | c_1, c_2) \ , & h = 1, \ldots, 5 \\
\ldots & \ldots &
\end{array}
$$

   We will make the following assumptions regarding the setup:

   - The probabilities $p_h, h = 1, \ldots, k$ of drawing "cherry" candies from the bag do not change as candies are being drawn the bag. As a result, the probabilities $(1 - p_h), h = 1, \ldots, k$ of drawing "lime" candies from the bag also do not change as candies are being drawn the bag.
   - The joint probability of drawing different candies from a bag are conditionally independent given the bag, i.e.,

$$
\begin{aligned}
p(c_1, c_2 | \pi_h) &= p(c_1 | \pi_h) p(c_2 | \pi_h) \ , \\
p(c_1, c_2, c_3 | \pi_h) &= p(c_1 | \pi_h) p(c_2 | \pi_h) p(c_3 | \pi_h) \ ,
\end{aligned}
$$

     and so on.

   You will have to develop code for the following function:

   `my_Bayes_candy`(π, p, $c_{1:10}$), which uses prior probabilities π, conditional probabilities p, and sequence of 10 candy draws $c_{1:10}$ to compute posterior probabilities of each type of bag.

   The function will have the following **input**:

   - Prior probability of each type of bag: $\pi = [\pi_1, \pi_2, \pi_3, \pi_4, \pi_5]$

- Conditional probability of cherry candies in each type of bag: $p = [p_1, p_2, p_3, p_4, p_5]$
- Sequence of 10 candies drawn from the bag under consideration: $c_{1:10} = [c_1, c_2, \ldots, c_{10}]$

where $c_i \in \{0, 1\}$ with "0" denoting cherry and "1" denoting lime candy. For the specific example discussed in class:

$$c = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1] \ .$$

The function will have the following **output**:

(a) A list of posterior probabilities of each type of bag $\pi_h, h = 1, \ldots, 5$ after every subsequence of candy draws, i.e., $c_{1:t} = [c_1, \ldots, c_t], t \in \{1, \ldots, 10\}$.

In particular, the output will be the following posterior probabilities:

$$\begin{array}{lrl}
\text{After drawing } c_1 : & p(\pi_h|c_1) \ , & h = 1, \ldots, 5 \\
\text{After drawing } c_1, c_2 : & p(\pi_h|c_1, c_2) \ , & h = 1, \ldots, 5 \\
\ldots & \ldots & \\
\text{After drawing } c_1, \ldots, c_{10} : & p(\pi_h|c_1, \ldots, c_{10}) \ , & h = 1, \ldots, 5
\end{array}$$

The function `my_Bayes_candy` will return a two-dimensional list of size 10x5 containing the aforementioned posterior probabilities. Specifically, the returned list must contain probabilities in the format:

$$\begin{gathered}
[[p(\pi_1|c_1),\ p(\pi_2|c_1),\ \ldots,\ p(\pi_5|c_1)], \\
[p(\pi_1|c_1, c_2),\ \ldots,\ p(\pi_5|c_1, c_2)], \\
\ldots, \\
[p(\pi_1|c_1, .., c_{10}),\ \ldots,\ p(\pi_5|c_1, .., c_{10})]]
\end{gathered}$$

Note that your code should work for any given $\pi, p, c_{1:10}$. We will consider 5 types of bags and 2 types of candies in each bag. `my_Bayes_candy` would have a time limit of 10 seconds. You could use the specific example discussed in class to test out your implementation. However, your code must work for any valid input.

2. (50 points) The problem will focus on developing your own code for: K`-fold cross validation`.

   Your code will be evaluated using five standard classification models applied to a multi-class classification dataset.

   Dataset: We will be using the following dataset for the assignment.

   `Digits`: The `Digits` dataset comes prepackaged with `scikit-learn` (sklearn.datasets.load_digits). The dataset has 1797 points, 64 features, and 10 classes corresponding to ten numbers $0, 1, \ldots, 9$. The dataset was (likely) created from the following dataset:
   `https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits`

   Classification Methods. We will consider five classification methods from `scikit-learn` and `xgboost`:

   – Linear support vector classifier: `LinearSVC`,
   – Support vector classifier: `SVC`,
   – Logistic Regression: `LogisticRegression`,
   – Random Forest Classifier: `RandomForestClassifier`, and
   – Gradient Boosting Classifier: `XGBClassifier`.

   Use the following parameters for these methods (do not specify any additional parameters):

   – `LinearSVC: max_iter=2000, random_state=412`
   – `SVC: gamma='scale', C=10, random_state=412`
   – `LogisticRegression: penalty='l2', solver='lbfgs', random_state=412, multi_class='multinomial'`
   – `RandomForestClassifier: max_depth=20, n_estimators=500, random_state=412`
   – `XGBClassifier: max_depth=5, random_state=412`

   You will have to develop **code** for the following two functions:

   (a) `get_splits`(n, k, `seed`), which returns: randomized k 'almost equal' sized lists of unique disjoint indices from the set of all indices $\{0, \ldots, n-1\}$, where the randomization depends on the integer `seed`. By 'almost equal', we mean the cardinality of the lists can differ by at most 1. These k list of indices correspond to the k folds over which cross-validation will be performed. The function will have the following **output**:

      i. a python list containing exactly k lists. Each of these sublists should be disjoint, each of size roughly $\frac{n}{k}$, contain elements from the set $\{0, 1, ..., n-1\}$ and must not contain repeated elements. The union of all the k sublists should include all elements in $\{0, \ldots, n-1\}$.

      Input `seed` determines the randomization and the output should be the same every time we use the same `seed` for a given n, k, and should be (randomly) different for different values of the `seed`.

      For example, `get_splits`(4, 2, 1) may return [[0,2], [1,3]], and the output must be same every time with the same input; `get_splits`(4, 2, 73) may return [[0,3], [1,2]]; `get_splits`(7, 2, 2) may return [[0,2,4,6], [1,3,5]]; `get_splits`(11, 3, 7) may return [[0,3,6,9], [1,4,7,10], [2,5,8]].

For our tests, n would be less than 1200. `get_splits` would have a time limit of 10 seconds.

(b) `my_cross_val`(`method`, X, $\mathbf{y}$, `splits`), which runs k-fold cross-validation for `method` on the dataset $(X, \mathbf{y})$. The **input parameters** are:

   i. `method`, which specifies the (class) name of one of the five classification methods under consideration,

   ii. X, $\mathbf{y}$ which is the data for the classification problem

   iii. `splits`: the output of the `get_splits method (len(splits) =` k`)`

   The function will have the following **output**:

   i. the test set error rates for each of the k folds.

   The error should be measured as $\frac{\texttt{\# of wrong predictions}}{\texttt{\# of total predictions}}$. The (auto)grader will judge your solution as correct if the difference between the reported and the expected mean error rates is within 10^{-3}. `my_cross_val` would have a time limit of 2 minutes.

   Within `my_cross_val`, strictly use the *splits* encoded in the input parameter `splits`. Do not define your splits for K-fold cross validation within this method.

   Also, make sure that you are NOT inadvertently shuffling your training data during K-fold cross validation. The training examples (for any particular split) should be in the same order as the input X.

Use `my_cross_val` to return the error rates in each fold for k fold cross-validation for the specific `method` (one of `LinearSVC`, `SVC`, `LogisticRegression`, `RandomForestClassifier`, and `XGBClassifier`) with the parameters outlined above.

To verify your code locally or on Google Colab, you could use:

```
from sklearn.datasets import load_digits
digits = load_digits()
X, y = digits.data, digits.target
```