ELG 5255 Applied Machine Learning Fall 2020

Quiz 1 (Bayesian Decision Theory)

As mentioned earlier, quizzes can be in the format of short quizzes or programming assignments. This quiz is in the format of a programming assignment.

Start Date: $Oct.\ 1^{st}\ 2020$

Due Date: $Oct.\ 7^{th}\ 2020\ 23:59$ **Eastern Time (US and Canada)**

Submission

You must submit your quiz on-line with Bright Space. This is the only method by which we accept quiz submissions. Do not send any quiz by email. We will not accept them. We are not able to enter a mark if the quiz is not submitted on Bright Space! The deadline date is firm since you cannot submit a quiz passed the deadline. It is a student's responsibility to ensure that the quiz has been submitted properly. A mark of 0 will be assigned to any missing quiz.

Quizzes must be done individually. Any team work, and any work copied from a source external to the student will be considered as an academic fraud and will be reported to the Faculty of Engineering as a breach of integrity. The consequence of academic fraud is, at the very least, to obtain an F for this course. Note that we use sophisticated software to compare assignments (with other student's and with other sources...). Therefore, you must take all the appropriate measures to make sure that others cannot copy your assignment (hence, do not leave your workstation unattended).

Goal

In this quiz, by using Bayesian Decision Theory, we will train our model to predict a Pokémon's type (e.g. grass, water, fire, etc.) given the Pokémon's attributes (e.g. attack, defense, height, weight, etc.). By doing so, we can better understand enemy's Pokémon and choose ours accordingly.

During this quiz, we will learn how to implement calculating posterior probity and utility. Meanwhile we will also be more familiar with Bayesian Decision Theory.

Dataset

This time we use Pokémon dataset, which contains information on all 802 Pokémon from all Seven Generations of Pokémon. The information contained in this dataset include Base Stats, Performance against Other Types, Height, Weight, Classification, Egg Steps, Experience Points, Abilities, etc. To reduce the training and testing time and to simplify the problem, only 600 samples are used and we have excluded all non-numeric values like Pokémon names. Also, we only need to predict the major type of Pokémon, namely 'type1'. For more information about the dataset such as distribution and classes, please access this link.

We have implemented the dataset loading function, which fills Nan with 0, scales attributes from 0 to 1, and encodes the labels. By calling loadDataset(), we can simply load the whole dataset.

Besides the dataset, we also generate the utility matrix (according to utility theory). The row of the matrix is the action and the columns denote the true classes. The values of the matrix are arbitrary, so no need to worry about the meaning of these values. By invoking loadUtilityMat(), utility matrix will be loaded.

Here is an example of loading the dataset and the utility matrix.

```python
# load the dataset
X, y, _ = loadDataset()
# split dataset to Training set and Testing set
XTrain, XTest, yTrain, yTest = train_test_split(X, y, test_size=0.2,
random_state=randomState)
# load utility matrix
utilityMat = loadUtilityMat()
```

Bayesian Decision Theory

Prior: $P(C_j) = \frac{N_{C_j}}{N_X}$, where N_{C_j} is the number of samples belonging C_j, and N_X is the number of samples in the dataset

Likelihood: $P(x|C_j)$

Evidence: $P(x) = \sum_{j=0}^{J} P(C_j|x)P(C_j)$

Posterior: $P(C_j|x) = \frac{P(x|C_j)P(C_j)}{P(x)}$, where C_j is the j class, e.g. 0, 1, 2, ... 18

Utility: $E(U(\alpha_i|x)) = \sum_{j=0}^{N_C} U_{ij}P(C_j|x)$, where N_C is the number of classes

Decision: $\alpha_i^* = \arg\max_{\alpha_i} E(U(\alpha_i|x))$

Since modeling the likelihood function has not been taught yet, we provide these necessary functions to help you overcome these issues. Please follow the instructions in the Instruction section, and it will be quite smooth to complete the quiz.

For more details about Bayesian Decision Theory, please check professor's lecture video and slides, as well as the problem solving video..

Instructions

File Structure

1. Data

   This folder stores our dataset and results. When you start the quiz, it should contain *Pokemon.csv* and *UtilityMatrix.csv*. You must not change the content within those files, otherwise, it will influence your final grade. After you implement all functions and methods, *results.csv* and *results.txt* should occur which store the numerical results.

2. main.py

   It is where we should implement our quiz. Please note that any function marked as **!!! Must Not Change the Content !!!** must not be changed, otherwise, it will influence your final grade. All functions marked by **ToDo: Implement This Function** should be well implemented.

3. Readme

   Quiz Instructions.

4. requirements.txt

   Required Python packets to finish the quiz. Please run the following code to install these packets.

   ```bash
   pip install -r requirements.txt
   ```

5. TicToc.py

   Used for timing functions. No need to change or check.

Main Function (main() in main.py)

This function will perform:

1. Load the Pokémon Dataset and Utility Matrix
2. Split the dataset to training set and testing set
3. Train the classifier
4. Test on the testing set
5. Store the results into *results.csv* and *results.txt*

A few details about our classifier in this assignment

Here are some explanations about the attributes and methods of this classifier.

Attributes

As the likelihood estimation function is already given; you don't need to know all of these attributes but they are just given as an explanation.

1. *X*, *y* are the training sets X and y
2. *classes* stores the unique classes of y
3. *meanDict* stores the mean vector of each class
4. *priorDict* stores the prior possibility of each class
5. *cov* is the covariance of the assumed distribution
6. *covInv* is the inverse of the covariance
7. *mvndDenominator* is $\sqrt{(2\pi)^k * (\det \Sigma_j)}$ which is used for speed up the calculation
8. ***utilityMat*** stores the utility matrix which is input from init function

Methods

1. __init__: Initialize the classifier
2. inv: Using SVD to calculate the inverse of the given matrix
3. fit: Train the classifier
4. likelihood: Calculate the likelihood $P(C_j|x)$
5. evidence: Calculate Evidence $P(x)$
6. prior: Calculate Prior $P(C_j)$
7. posterior: Calculate posterior probability $P(C_j|x)$
8. utility: Calculate the utility expectation of the action given x, $E(U(\alpha_i|x)) = \sum_{j=0}^{N_C} U_{ij}P(C_j|x)$
9. predictSample: Predict a sample to maximize utility expectation. $\alpha_i^* = \arg\max_{\alpha_i} E(U(\alpha_i|x))$
10. predict: Predict all given samples
11. actualUtility: Calculate the actual utility of yPredicted

What We Need to Do and Implement

Must not change the shape (format of parameters and return values) of the functions that we need to implement.

Must not change the imports. No additional python packets should be used.

1. Install Required Packets

   ```bash
   pip install -r requirements.txt
   ```

2. Open main.py

3. Implement Classifier.posterior (**20 marks**)

   ```python
   def posterior(self, x, c) -> float:
   ```

   This function takes a sample and a class as parameters and returns the posterior probability of given x and class.

   $P(C_j|x) = \frac{P(x|C_j)P(C_j)}{P(x)}$, where the likelihood, prior and evidence function have been implemented.

   Please note that the return value should be $P(C_j|x) = \frac{P(x|C_j)P(C_j)}{P(x)}$, instead of $P(C_j|x) = P(x|C_j)P(C_j)$, although evidence will not influence the decision.

4. Implement Classifier.utility (**40 marks**)

   ```python
   def utility(self, x, action) -> float:
   ```

   This function takes a sample and an actions as input and returns the utility expectation of given x and class.

   $E(U(\alpha_i|x)) = \sum_{j=0}^{N_C} U_{ij}P(C_j|x)$

5. Implement Classifier.predictSample (**40 marks**)

```python
def predictSample(self, x) -> (int, float):
```

This function takes an sample as input and return a tuple of the predicted class and the expectation of choosing this class

The action of choosing the class should maximize the utility expectation.

$\alpha_i^* = \arg\max_{\alpha_i} E(U(\alpha_i|x))$

6. Run main.py

7. Open Results.csv and Results.txt to check the results

Expected Results

After we implement and run the code, we should be able to see the following results.

1. Results.csv

```
yPredicted,yTrue,uExpected,uActual
12.0,12,9.691992936800657,10
0.0,0,9.999999999999998,10
10.0,10,9.996865279756054,10
13.0,13,9.99999995442563,10
11.0,11,10.0,10
1.0,1,9.999999803224163,10
13.0,13,9.999999999852577,10
13.0,13,9.999999999946706,10
...
```

2. Results.txt

```
              precision    recall  f1-score   support

           0    1.00000   0.82353   0.90323        17
           1    1.00000   1.00000   1.00000         3
           2    1.00000   1.00000   1.00000         2
           3    0.50000   1.00000   0.66667         3
           4    1.00000   1.00000   1.00000         1
           5    1.00000   1.00000   1.00000         3
           6    1.00000   1.00000   1.00000         4
           7    0.85714   1.00000   0.92308         6
           8    0.80000   1.00000   0.88889         8
           9    0.62500   0.71429   0.66667         7
          10    1.00000   0.66667   0.80000         3
          11    1.00000   1.00000   1.00000        18
          12    1.00000   1.00000   1.00000         3
          13    0.82353   0.93333   0.87500        15
          14    0.83333   0.62500   0.71429         8
          15    1.00000   1.00000   1.00000         3
          16    1.00000   0.81250   0.89655        16

    accuracy                        0.89167       120
```

```
   macro avg    0.90818   0.91620   0.90202       120
weighted avg    0.91198   0.89167   0.89379       120

Average of Expected Utility: 9.922089637514866
Average of Actual Utility: 9.275
```

Marking Criterion

Since the random state is fixed we should see the exact same results as the expected results.

Only if we implement the whole function of a method, we can get marks; otherwise, we will get 0 on the method.

Implementing methods can receive marks, while only submitting results files will not receive any marks.

1. Implement Classifier.posterior (**20 marks**)
2. Implement Classifier.utility (**40 marks**)
3. Implement Classifier.predictSample (**40 marks**)