Pattern Recognition: Homework 6

Due date: 2023.4.11

Problem 1 (30 pt)

A casino owner recently discovered that a player was consistently winning in the dice game. He suspected that the player had switched fair dice with unfair ones during the game, so he used a camera to record the number of points for each roll in every game.

(1) The point information is stored in `sequences.npy` file, where each row represents a game and each column represents a roll result. There are 200 games in total, and each game rolls 30 times. Based on this data, establish and train an HMM model, and provide initial, transition, and emission probabilities obtained from fitting in your report.

(2) According to your established model, **manually** calculate the probability of observing sequence 6 6 6 6 using forward algorithm and backward algorithm respectively.

(3) If this player is currently playing games with observed point numbers before this round being: 3, 2, 1, 3, 4, 5, 6, 3, 1, 4, 1, 6, 6, 2, 6 , please infer whether this player is cheating based on your established model. If yes, when did he switch his dice?

Note:

(1) You can use `hmmlearn` module to build HMM models directly by installing it through `pip install hmmlearn` command in Python; please search for specific usage methods yourself.

(2) Friendly reminder: gambling harms health; background information provided here is purely fictional:)

Problem 2 (70 pt)

We have finished learning almost all the (classical) supervised learning methods except for deep learning neural networks. Now it's time to wrap them up and have a comprehensive understanding and taste of them. In this problem, **you need to implement all the 10 different methods we have learned in class on three different datasets and compare their performance.**

Don't worry! It is not horrible, because we have already provided you with the basic framework for loading the dataset, fitting the training set, validating the validation set, and final result report and plot. What you need to do is just using `sklearn` package to construct each method by using a few lines, and tune the hyperparameters to maximize each method's capacity on each dataset. Here we specify the hyperparameters that you need to tune for each method and the list of options for each hyperparmaeter.

- **Decision Tree**.
  - `criterion: {'gini', 'entropy', 'log_loss'}`
  - `max depth: {3,5,10,20}`
- **Random Forest**.
  - `criterion: {'gini', 'entropy', 'log_loss'}`

  - `n_estimators: 20, 50, 100, 200`

- **Bagging**.
  - `n_estimators: {2, 5, 10, 20}`
- **Gradient Boosting**.
  - `n_estimators:{ 10, 20, 50, 100, 200 }`
  - `learning_rate: {1e-3, 1e-2, 1e-1}`
- **Naive Bayes (GaussianNB)**.
  - `var_smoothing: {1e-9, 1e-7, 1e-3, 1e-1}`
- **Perceptron**.
  - `penalty: {None, 'l1', 'l2'}`
  - `alpha: {1e-5, 1e-4, 1e-3, 1e-2, 1e-1}`
- **Logistic Regression**.
  - `penalty: {None, 'l1', 'l2' }`
  - `C: {0.1, 1, 10}`
- **LDA (Linear Discriminative Analysis)**
- **SVM**
  - `kernel: {'linear', 'poly', 'rbf', 'sigmoid'}`
  - `C: {1e-3, 1e-2, 1e-1, 1, 10, 100, 1000}`

In `sk_all_algorithms.py`, we present an example of how to use a decision tree for classification, along with detailed instructions. We have highlighted the sections of code that require completion in TODO comments. Your tasks are:

(1) For all the remaining algorithms, find the corresponding algorithms in `sklearn` library and finish the declarations in the `models` variable.

(2) Set different hyperparameters for each algorithm in `model_hparams` variable using the list of hyperparameter options provided above.

(3) For each algorithm, summarize the results on the **validation set** for all datasets in tables.

(4) For each algorithm, report the optimal hyperparameter configuration (selected based on the validation set results) and the accuracy on the **test set** for all datasets.

Important Note: Tuning hyperparameters on the test set is a **serious mistake** in machine learning. Therefore, we divided the data into training, validation, and test sets (7/1/2). We use the validation set to experiment with different hyperparameters in the `run_method` function. It is only in the `report_result` function that we test the actual performance of the model.