HW2

Submission date: 23.11

Ex1

In this exercise, you will implement one of the behavioral models that exhibit context effect choice behavior which we learned in class – the compromise effect model. So far, we saw how these effects work on at most three items and for two dimensions; now, we consider more general settings with choice sets of arbitrary size and dimensions.
A dataset in this exercise will consist of:

(i) m choice tasks (i.e., examples – here we call them "slates"), each composed of

(ii) n items, with each item described by

(iii) q features.

You will implement the compromise effect component of the Context RUM model.

Specifically, write a class with the following interface:

CompromiseUserModel(UserModel)

- utility(self, X): calculates utility
- choice(self, X): returns the selection

Description:

1. choice: a function that get as input a numpy array X of size mXnXq (i.e., m choice sets, each of n items, each described by q features), and outputs a numpy array of choices y of size m with each entry in {1,…,n} (i.e., item position indicators).
2. utility: a function that get as input a numpy array x of size mXnXq, and outputs an array of utility values of size mXn (i.e., one value per choice set, per item).

A user's utility in this model is defined to be a combination of (i) an item-wise utility, and (ii) a set-dependent "modification" (i.e., a behavioral context effect). Specifically, utility is a parameterized function $v(x|s;\beta,\alpha)$ where $\beta \in \mathbb{R}^q, \alpha \in \mathbb{R}$ are parameters, and is defined by:

$$v(x|s;\beta,\alpha) = \beta^\top x + \alpha \cdot \mathrm{com}(x|s)$$

Be sure to initialize the class with *beta* and *alpha* as arguments to the constructor.

The $\mathrm{com}(x|s)$ term as defined as follows:

- Define a compromise, set-dependent "reference point" $r(s) \in \mathbb{R}^q$ by:

$$(r(s))_i = \frac{\min_{x \in s}(x)_i + \max_{x \in s}(x)_i}{2}$$

- Let $d_{com}(x, x') = \|x - x'\|_2$ be *the Euclidean distance* between x and x'
- The compromise term is simply $\text{com}(x|s) = -d_{com}(x, r)$.

Hint: To validate your code, try it on the simple 3-item, 2D example we saw in class that demonstrates the effect. Vary α from 0 to whatever number necessary to observe the effect.

Ex2

In this exercise you will implement a simple learning-to-rank algorithm that can be applied to discrete choice data. The approach consists of two main steps:

1. define the discrete choice prediction problem as a learning-to-rank problem with *partial ranking inputs*.
2. Solve the (partial) learning-to-rank problem using a reduction to a binary classification problem – this is done by appropriately transforming the dataset, and applying an off-shelf classification algorithm (specifically, we will use scikit-learn's SVM).

The basic idea is to take each example (s, y) (where $s = (x_1, \dots x_n)$ and $y \in [n]$) in the original data set S, and create k=n-1 new examples $(\bar{x}, \bar{y})$ in a new data set T. Each new example corresponds to one pair of items $x_y, x_i \in s$ where x_y is the chose item and $y \neq i \in [n]$.

Steps:

a. **Transform data into pairs with balanced labels**

   Implement a function *transform_pairwise(X,y)* with:

   Input: (both NumPy arrays)

   - X (nXq) – a single choice set of n items having q features
   - y (scalar in {1,…,n}) – a chosen item (indicator to position in set)

   Return: (both NumPy arrays)

   - X_{new} (kXq)
   - y_{new} (k) with entries in {-1,+1}

Specifically, k=n-1 – this in the number of <u>pairs</u> of items in X. Elements in X_{new} and y_{new} are defined in the following way: first, sample a scalar $a = 1$ w.p. ½ and $a = -1$ w.p. ½; then, set:

$$(\bar{x}, \bar{y}) = \left(a\left(x_y - x_i\right), a\right)$$

Be sure to set the numpy.random seed to 0!

b. **Create Partial Ranker**

Create a class "MyRanker" that inherits from *svm.LinearSVC* and contains the following class functions:

- *shape_transform (self, X, y)*
  <u>Input:</u> X ($m \times n \times q$) and y (m) – NumPy arrays
  <u>Return:</u> X' ($m \cdot k \times q$), Y' ($m \cdot k$)
  Create a new dataset of binary-labeled examples by applying transform_pairwise to all examples in the input dataset (X,y), and concatenating the results

- *Fit (self, X, y)*:
  <u>Input</u>: X (mXnXq) and y (m) – NumPy arrays
  <u>Return:</u> self
  Fits *svm.LinearSVC* to the transformed binary-labeled data set.

- *Predict (self, X):*
  <u>Input:</u> X (mXnXq) – NumPy array
  <u>Return:</u> y (m) – NumPy array, idx of top choice per slate
  Predicts using the argmax rule, and according to the coefficients of the learned SVM model.

Submission guidelines:

1. Submit a .ipynb file with your solution.
2. Add a text block in the beginning of your notebook with your IDs.
3. Indicate clearly with a text block the sections of your solutions .
4. For any questions regarding this homework, contact Bar.