BanditNet with Partial Support

Anika Talwar,Henry Samuelson, Phillip N O'Reggio

Spring 2019

1 Introduction

We aim to develop and analyze an algorithm that generates a policy for a given dataset, which has been appropriately normalized and split into training, test, and validation sets, outputting a lambda value that will later be used to test batch learning from logged bandit feedback.

2 Basic Outline

1. Combine both training and test data sets to maximize row count.

$$D = D_{test} \cup D_{Train} \tag{1}$$

2. Normalize all the features (output and input) by mean and standard deviation. Normalized equation below on each column:

$$\frac{X - \mu}{\sigma} \tag{2}$$

3. Softmax Equation as a function of λ.

$$softmax_{\lambda}(y|s) = \frac{exp(\lambda s_y)}{\sum_y exp(\lambda s_y)} \tag{3}$$

4. generate the $\delta(y|x)$ by normalizing the output features to percentile rank. This is done by sorting the list least to greatest. Where S is sorted, and i is an index

$$\frac{S_i}{i} \tag{4}$$

5. Make train/test split S_{train} and S_{test}

6. further split train S_{train} into train for logger $S_{Log-Train}$ and train for banditnet $S_{banditnet-train}$

7. Train linear regression on logger training set $S_{Log-Train}$. From predictions s of the linear regression, generate Logging policy π_0

$$s = \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix} \rightarrow \pi_0(y|s) = softmax_\lambda(y|s) \tag{5}$$

   Takes the last 30% and uses it as a Validation dataset. Uses the first 30% ** At the moment not doing this ** Drops rows that contain banned values (EX. -999 on the higgs dataset since since that represents "variables [that] are meaningless or cannot be computed" Doesn't do this since when normalized, it's hard to distinguish these types of number from other rows Preprocesses data: Right now does nothing since this is handled by the R file before running Sets up the Linear Regressor with learning rate, gradient clip, inputs, outputs, etc. Trains on the first 70% (train/test) using batch sizes of size LINEAR-BATCH-SIZE for LINEAR-STEPS number of steps. If REPORTING-PERIOD is not 0, will record the current loss based on a sample of size REPORTING-SAMPLE-SIZE of the train/test data.

8. Generate Bandit-data on train data for banditnet $S_{banditnet-train}$

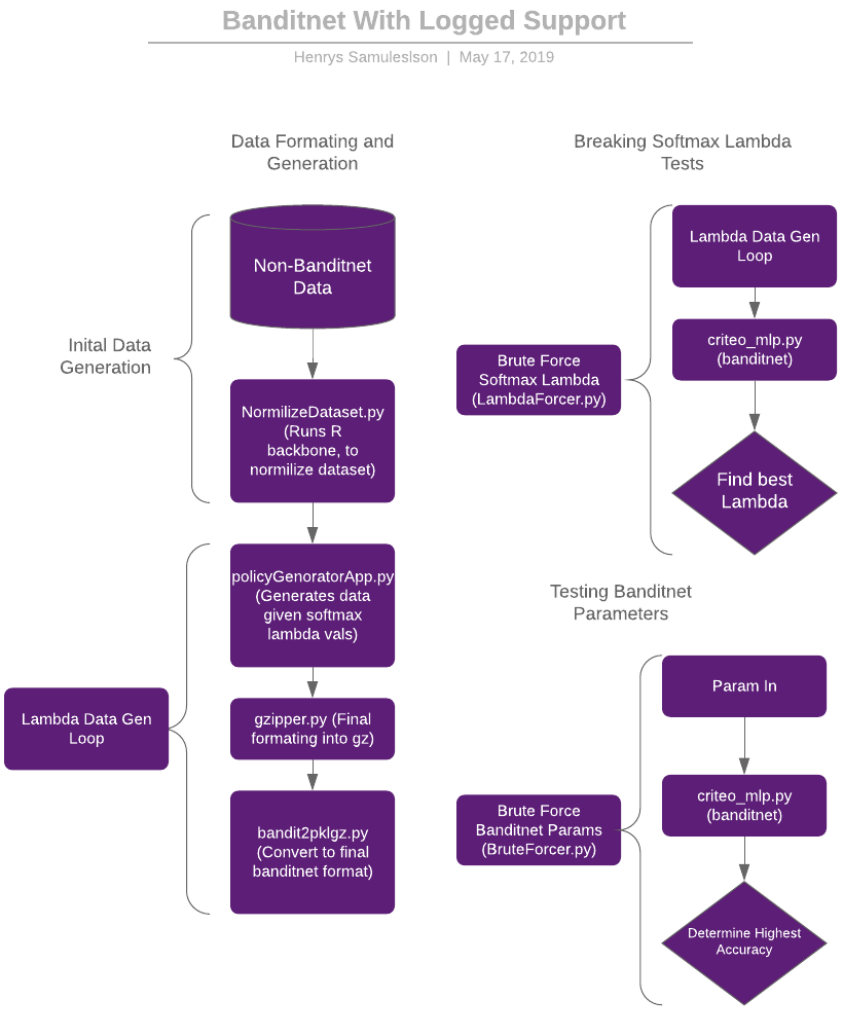
$$S_\lambda = ((x_i, y \sim \pi_0(y|x_i), pi_0(y|x_i), \delta(y|x)))^* \tag{6}$$

9. Train Banditnet on $S_\lambda \rightarrow \pi$

10. Test π on Test 10 output of S_{test}

$$\sum_{10-Test} \sum_y \pi(y|x_i)\delta(y|x_i) \tag{7}$$

3 Code Description



3.1 Initial Lambda Values

First, the lambda value arguments inputted by the user are read into a list of floats. If no lambda value is specified, then default of [2.0] is used.

3.2 Data Initialization and Preprocessing

After getting the lambda values to be used for generation, the data is read into a pandas dataframe from a csv file called "NormilizedTraining5perc" in the same directory as policyGeneratorApp.py. Then, rows containing values in VALUES_TO_DROP constant are removed; specify_dropped _values() specifies what values should be dropped. Columns used as inputs and outputs are then specified according to the values of the InputFeatures and OutputFeatures enum classes. The data is then initialized and preprocessed using policyGen.py methods set_up_data and preprocess_data. The former uses the 2 lists of string feature names to create separate pandas dataframes; one containing all the input feature columns and one with the output feature columns. In the latter method, the data is preprocessed by changing the values of both the input and output dataframe to their percentile ranking. Percentiles are calculated by replacing the value of a row by its percentile based on all the values in its column.

3.3 Set Up Regression

The linear regression is set up by initializing a tensorflow Linear Regressor with the dataset, a learning rate of 0.05 and a gradient clip of 5.0.

3.4 Train Data

The tensorflow Linear Regression model is then trained using only the first 5 percent of the rows of the percentile normalized dataset. After training, validation statistics for each output feature and the average results is calculated based off of the entire dataset. If period in train_linear is not equal to 0, will periodically print out the current RMSE for all the output features and the average RMSE, which was used in debugging.

3.5 Softmax

After the data has been passed to the linear regressor and predictions have been generated, the softmax function is applied using the softmax equation. First, the array of predictions for each output feature is obtained and the softmax sum (given in the denominator of Equation 3) is calculated for each feature's set of predictions. Next, a dictionary is kept as an accumulator with the keys as the output features and the values being the softmax values applied to each prediction. Finally, the dictionary containing all the relevant softmax values is converted to a pandas dataframe which will later be used while generating bandit data.

3.6 Banditnet Data Generation

3.6.1 Files and Split

Lastly, the data is generated in the bandit net data format, for each lambda used as an argument. The data is generated in 3 splits: a train, test, and validation set (Which means 3 files are generated for each lambda value specified. The files are called banditnet_lambdaN_train.txt, banditnet_lambdaN_train.txt, and banditnet_lambdaN_train.txt respectively (where "N" would be replaced by the value of the lambda value it was generated with. Whole valued numbers would be shown without the decimal point. For example: 0 would be lambda0, while 1.8 would be lambda1.8). The size of these splits are determined by the values of the constants TRAIN_SPLIT and TEST_SPLIT. If either of these values are equal to 0, then it will instead generate a singular file containing all the splits together (called banditnet_lamnbaN.txt). Theses txt files are located inside of a directory called banditnet_data.

3.6.2 Banditnet Format and Choices

Banditnet data is formatted with a "header", containing general information about the body and a "body", which has information about chosen products. The header contains information for a exID, hashID, wasAdClicked, propensity, nbSlots (number of slots), nbCandidates (number of candidates), and displayFeatures. To convert the dataset to fit these categories, the row ID was used as a substitute for exID (replaced by the number) and hashID (replaced by the hash value of the string representation of the row ID). wasAdClicked is always replaced by the number 1. propensity is determined by the result of the propensity of the first item picked from the softmax set using random sampling without replacement. nbSlots is always one, while nbCandidates is equal to the number of output features. displayFeatures corresponds to each of the input features; the first number before the colon is the index of the feature with respect to the row and the number after the colon is the value of said feature. The body contains one element for each output feature, ordered by random sampling without replacement of the softmax results of the row. In the body, information such as wasProductClicked, exID, and productFeatures are included. wasProductClicked is replaced with 1 just like the body, and exID is the same as the exID in the header. For productFeatures, the number before the colon corresponds to the selected features position in the dataset, while the number after the colon corresponds to the index of the feature (which is always 1 here).

3.7 Debugging Methods + Misc

3.7.1 parse_feature_labels()

Used to get columns from the dataset based on their index instead of by name. Is currently not used to specify what features are inputs and outputs.

3.7.2 print_inputs_targets_names()

Prints the values of the names of columns used as input values and output values.

3.7.3 print_data_stats()

Prints statistics about the input and output columns of the dataset. Information includes the count, number of unique values, mean, median, mode, percentiles, maxes, and mins. Also prints the first 10 values of input and output columns.

3.7.4 print_predictions()

Prints a number of predicted values using the same dataset the linear regressor was trained on, or a different dataset if provided. Used to debug the results of linear regressor and to make sure training was yielding fairly comparable values to the true values.

3.7.5 make_predictions_dataframe() and make_percentile_dataframe()

The predictions and percentile dataframes were constructed to initially be used in the banditnet data generation phase but were later useful in debugging. The predictions dataframe was a dataframe of the output feature names as the column headers and the values being the predictions returned by the linear regressor for each example corresponding to the output feature. make_predictions_dataframe() constructs this dataframe and scales the values to be in the range 0 to 1. The percentile dataframe was a dataframe of the input feature values, their predictions, a randomly chosen softmax value, and the reward based on the scaled percentile values. make_percentile_dataframe() uses percentile_scale() to scale the predictions appropriately.

3.7.6 show_loss_graph()

Graphs the RMSE against number of steps in training. Lines for each individual output feature are partially transparent and colored randomly to distinguish between different features. The average RMSE of all output features is plotted using an opaque line.

3.7.7 progress_bar()

This is called in generate_bandit_data to show how the progress of creating banditnet data for each lambda.