TBEEF Documentation

Christopher V. Rackauckas

August 14, 2013

This program is TBEEF, a doubly ensemble framework for prediction problems. This documentation is written for those who wish to use this program in order to solve their specific prediction problems.

1 System Requirements

The program can run on Mac OSX and Linux. The program can be easily modified to run on Windows by disabling the transformation of the data into binary format for libFM (libFM does not support using binary files in Windows), though this is not recommended for performance issues. The required tools are as follows:

- The tools for compilation of libFM and SVDFeature (standard GNU tools).
- Python 3.2.2: Other versions of python may work but are not recommended.
- R 3.0.1: Other version of R may work but are not recommended.
- Perl : Required for the data conversion in libFM.

In addition, the default ensemble models require the use of the following R packages (these are easily installed using the scripts/installPackages.R script):

- Metrics : Used for calculating the RMSE for all models.
- ridge : Used for the ridge regression model.
- glmnet : Used for the lasso regression model.
- ipred : Used for the bagged regression trees model.
- BMA : Used for the Bayesian model averaging model.
- randomForest : Used for the random forest model.
- party : Used for the conditional inference random forest model.
- mboost : Used for the gradient boosted regression tree model.

2 Getting Started

The steps for installing TBEEF are as follows:

1. Specify a folder to place the source code.
2. Download and compile the libFM and SVDFeature binaries (including the extra tools if needed).
3. In the Models folder, create a directory called 'libFM' and place the libFM binaries (libFM, convert, and transpose) in this folder (make sure these are given executable permissions). Also include the conversion file triple_format_to_libfm.pl.
4. In the Models folder, create a directory called 'SVDFeature' and place the SVDFeature binaries (svd_feature and svd_feature_infer) inside this folder. Also place the compiled tools binaries in this folder (to make Models/SVDFeature/tools). Make sure all of these binaries are given executable permissions.
5. Go to Data/Original. Untar BaiduCompetitionData.tar.gz into this folder. This should give 5 files:
   - data_set.txt, the training/CV dataset. Contains three columns, the first being the independent variable (movie rating), the second a user id, and the third a movie id.
   - predict.txt, the test dataset. This contains two columns, the first a user id and the second a movie id.
   - movie_tag.txt, user_social.txt, and user_history.txt. These are special datasets used for feature engineering in the Baidu dataset. The movie_tag.txt file contains a movie ids in the first column and then ids for associated movie tags in the following columns. The file user_social.txt contains a user id and then the next columns are the user ids of the users that the specific user follows. The file user_history.txt is a list of (userid,movieid) pairs where the user has seen (but maybe not rated) the given movie.
6. Go to the scripts folder and run installPackages.R. This will install the R packages required by the default ensemble models.
7. Go to the main directory and run the program using python3 driver.py. This by default will run the basic factorization machine model and the basic SVD Feature model, and then ensemble them using OLS regression and then synthesize them using OLS regression. It should be successful.
8. Investigate the outcomes in the Data folder. Check the ModelPredictions and the HybridPredictions folders to see the predictions given at these steps of the algorithms (by the respective models) and check Output to see the output files. Also look at the log files to see the logs of the model runs.

9. Clean out the data folders before your next run using python3 clean.py.
10. Turn on more models! Do this by opening config.py. Uncomment some of the libFM, SVDFeature, and ensemble models. Maybe even change the synthesize model.
11. Re-run the program using python3 driver.py.

3 Program Structure

The program runs as follows. The driver is the brains of the program, directing the entire scheme. It first calls a pre-processing step which takes the data from Original and saves out pre-processed forms to PreProcessed. Then the model setup phase begins. First, the model objects are constructed. Then, they call their setup methods which generates datafiles in ModelSetup until they produce the files for runtime which are saved in ModelData. Then the model's run methods are called. They read in the data from ModelData and saves out predictions to ModelPredictions. These predictions are then aggregated into the HybridSetup folder. The ensemble methods are then called and they take the data from HybridSetup to produce the files in HybridPredictions. These are then compiled into training and test matrices for the synthesize phase and saved in the SynthSetup folder. The chosen synthesis ensemble method is ran and places its predictions in the SynthPredictions folder. The post processor looks at the datasets in the SynthPredictions folder and saves the predictions per trial in the output folder. It then takes the one with the lowest RMSE on a CV and copies it to output.txt.

4 Data Structures

The main data structures of the program are as follows:

- Models are defined as an object. Models start by definition by the user in the configuration file, config.py. The user specifies the models as follows:

  [tag,program,featureSet,[misc]]

  - Tag is a unique identifier to the model. It must be unique in order to ensure the models do no overwrite eachother's data files.
  - Program is a string, either 'FM' or 'SVD', which states whether the model should be evaluated using either libFM or SVDFeature respectively.
  - Feature set is a parameter that lets the user choose which feature set the models should use. To know which features are implemented, look inside the PreProcess directory to find the libFM and SVDFeature setup directories. Inside each of these directories should be a file titled -FeatureSetup.py. These are the folders where the feature choice

    process occurs. Look for conditional statements based on model[2], these are the statements that check the feature set parameter.
  - Misc is a list of miscellaneous parameters for controlling the models. Currently, the choices are as follows:
    * libFM: ['dimensions']
    * SVD: []

At the model setup phase, these are converted into an object defined in utils for the purpose of calculation. These are then saved into modelList, an array stored in driver. The methods and fields are defined as follows:

- tag : a unique identifier for the model .
- mode : defines which program to use, choices are 'FM' and 'SVD'
- featureSet : defines which feature set to use. See the model's setupFeatures method for options.
- misc : the array passed in as misc.
- trial : the trial the model is assigned to. Saved as a string.
- Setup Data Paths:
  - bootTrain : Training dataset given by bootsplit
  - bootCV : CV dataset given by bootsplit
  - bootTest : Test dataset with dummy variables (required for computation)
  - featTrain : The training dataset just after features are added
  - featCV : The CV dataset just after features are added
  - featTest : The Test dataset just after features are added
  - tmpTrain : Temporary dataset in the setup process
  - tmpCV : Temporary dataset in the setup process
  - tmpTest : Temporary dataset in the setup process
  - runTrain : Training dataset for the model to run
  - runCV : CV dataset for the model to run
  - runTest : Test dataset for the model to run
  - predCV : Where the predictions from the CV dataset are saved
  - predTest : Where the predictions from the Test dataset are saved
- Feature Paths: These are the external datasets used for feature creation for the Baidu dataset.
  - movieTagPath : Path to the movie tag dataset

  - userSocialPath : Path to the user social dataset
  - userHistoryPath : Path to the user history dataset

The Model is class is not used directly. Rather, it is used through four subclasses, FMModel, SVDModel, HybridModel, SynthModel which are determined by the mode. The HybridModel and SynthModel are equivalent except in their construction due to the differences in paths. These have extra fields as follows:

- FMModel
  - dims : Dimension of the factorization machine.
  - logCV : Path for the printout of the log file for the CV run
  - logTest : Path for the printout of the log file for the Test run.
  - libFMBinary: Path to the libFMBinary.
  - strItr : number of iterations for the program to run. Stored as a string.
  - globalBias : 1 means use global bias in factorization machine. 1 by default.
  - oneWay : 1 means use one way interactions in factorization machine. 1 by default.
  - initStd : The initial standard deviation for the MCMC optimization technique as specified by the user in config.py
- SVDModel
  - numItr : Number of iterations for the training run
  - SVDBufferPath : Path to the svd_feature_buffer program
  - learningRate : Learning rate (λ) for the SGD optimization technique used in SVDFeature
  - regularizationItem : Regularization term for the item parameters
  - regularizationUser : Regularization term for the user parameters
  - regularizationGlobal : Regularization term for the global parameters
  - numFactor : Number of factors used in the model
  - activeType : Sets the SVDFeature active type parameter
  - modelOutPath : Folder where all of the .model files are kept
  - SVDFeatureBinary : Path to the SVDFeature binary
  - SVDFeatureInferBinary : Path to the SVDFeature Infer binary

These subclasses also share the following methods:

- setup : Sets up the model dataset to be ran

- setupFeatures : Builds the features
- run : Runs the model
- fixRun : Fixes the printout of the run (or runs the prediction part)

Additional helper methods are used for carrying out these procedures:

- logCV : Where the logfile for the CV run is saved (libFM)
- logTest : Where the logfile for the Test run is saved (libFM)
- configPath : Where the config file for the run is saved (SVDFeature)
- HybridModel and SynthModel
  - tag : a unique identifier for the model .
  - mode : defines which program to use, choices are 'FM' and 'SVD'
  - featureSet : defines which feature set to use. See the model's setupFeatures method for options.
  - misc : the array passed in as misc.
  - trial : the trial the model is assigned to. Saved as a string.
  - Data Paths:
    - masterTest : The path to the ids for the test set. This is for appending to the prediction file after the predictions have been made.
    - runTrain : The path to the training set for running the model.
    - runCV : The path to the cross-validation set for running the model.
    - runTest : The path to the test set for running the model.
    - bootTrain : The path to the original training dataset for the model. Used for fixing the predictions after the run.
    - bootCV : The path to the original CV dataset for the model. Used for fixing the predictions after the run.
    - bootTest : The path to the original test dataset for the model. Used for fixing the predictions after the run.
    - predCV : The path that the predictions from the CV dataset are saved to after processing.
    - predTest : The path that the predictions from the test dataset are saved to after processing.
    - predCVTmp : The path that the predictions from the CV dataset are saved to before processing.
    - predTestTmp : The path that the predictions from the test dataset are saved to before processing.
    - log : Path where the log file (R printout) is saved.

* RMSEPath : Path where the RMSE of the CV dataset is saved.
* miscStr : A string from the misc files used for passing into R.
* ensembleScriptPath : The path to the ensembles script.
* RCatch : A string used for calling R.

These both call run in order to run the R script.

5 Data Flow Note

The flows for the datasets for the setup phase are different between libFM and SVDFeature and should be noted. Both start with the boot datasets. libFM then immediately adds features, taking boot->feat. Then it converts it into the libFM sparse matrix as feat->tmp. Then it is converted into the libFM binaries as tmp->run for runtime.

SVDFeature on the otherhand is different. It starts with boot but it is first reindexed going boot->tmp. Then features are added tmp->feat. Then the datasets are converted to the sparse form and the buffers feat-run.

6 Modifying TBEEF

TBEEF has a plugin interface for easy modification of its models and ensemble methods. Three main points are discussed here:

1. Setup the basic models for the new datasets.
2. Feature-Engineering models for new datasets.
3. Implementing new ensemble methods.

Additionally, a fourth but less recommended option of implementing new model programs (instead of using libFM and SVDFeature, say to implement Boltzman machines or neural networks) is discussed.

6.1 Setting Up TBEEF Basic For New Datasets

To setup the basic models for new datasets, simply replace the data files data_set.txt and predict.txt in Data/Originals with the appropriate files for the dataset ids and the prediction ids. Running the basic factorization and SVDFeature models should work at this point (if they are formatting like the Baidu dataset, note that you should check that the line endings should be Linux style). You can modify the paths for any of the files in the utils/utils.py file.

6.2 Feature-Engineering

To feature-engineer for this program, you need to incorporate the extra feature datasets and build these features into libFM and SVDFeature respectively.

To incorporate the feature datasets into the programs, follow the examples of the Baidu dataset in the utils/utils.py, PreProcess/preProcess.py, and utils/Model.py files. The steps are as follows:

- Define the paths at which the extra feature datasets are stored. It is recommended that this would be a path to the Data/Originals folder. If preprocessing is necessary, also define paths for the preprocessed data files which is recommended to be in the Data/PreProcessed folder.
- If necessary define the appropriate preprocessing steps in PreProcess/preProcess.py.
- Modify the constructor in utils/Model.py (or utils/FMModel.py and utils/SVDModel.py if the preprocessed files are different) to add the appropriate paths to the preprocessed files to the object.

At this point, the object has the appropriate paths to the extra feature files and one simply needs to add new options to the setupFeatures method of the respective model type. New featureSets can be used by putting in a conditional checking for a new string passed in through the models list in config. As an instance function, it has access to all instance variables, which includes all relevant dataset paths. If any other materials are needed, it is wise to pass them into the model by adding them as class attributes through the constructor.

6.3 Implementing New Ensemble Models

New ensemble models are easily implemented by following the structure in Hybrid/ensemble.R. The following variables are defined in this file:

1. model.type : The type variable from the config.py file used for choosing the model.
2. dataTrain : The training matrix.
3. dataCV : The matrix for the cross-validation.
4. dataTest : The matrix of predictors for prediction of the test set.
5. input1 : The first input in the misc array. Extra inputs from the misc array are added in this same manner that input1 is defined.

To define a new ensemble model, build a conditional asking for the appropriate model.type string. In here, write a prediction algorithm that outputs predictions to CVPredictions and testPredictions arrays respectively. It is recommended that details about the run are printed out (these will automatically be saved to the appropriate log file). The program will use these two arrays to calculate the RMSE and build the prediction files.

6.4 Implementing New Model Forms

New model forms can be implemented, though this is not recommended due to the extra complexity, though it can easily be done. To do this, define a new object for the model form. Create a constructor which passes in the necessary information and paths. Define a function called setup for setting up the model for running. This should take files from bootTrain, bootCV, and bootTest to runTrain, runCV, and runTest. Add this model in the same manner as libFM and SVD to the PreProcess/setupModels.py file. This will make and setup the models be constructed given the definition in the config.py file. In your model, define a function called run for taking files from runTrain, runCV, and runTest to predCV and predTest (the predictions). Note that these fields are required by the script for finding the predictions in the hybrid step (the script Models/runModels.py adds model.predCV and model.predTest to CVPrediction Paths and testPredictionPaths respectively, from which the hybrid model takes the prediction array to build its training and test matrices).