Research in Industrial Projects for Students

Institute for Pure & Applied Mathematics
IPAM
University of California, Los Angeles

Sponsor

Baidu, Inc.

Technical Report

Doubly Ensemble Movie Prediction With Social Media Data Using TBEEF

Student Members

Christopher V. Rackauckas, University of California, Irvine (Project Manager)

Weijie Cai, Hong Kong University of Science and Technology

Colin Jarvis, Macalester College

Chenxiao Xu, Hong Kong Baptist University

Academic Mentor

Avery Ching, Hong Kong University of Science and Technology

Sponsoring Mentors

Shi Hongzhu, Baidu, Inc.

Wang Guanchun, Baidu, Inc.

Hu Yichuan, Baidu, Inc.

Date: August 10, 2013

Abstract

In the spring of 2013, Baidu, Inc. hosted a competition for teams to develop new algorithms for movie recommendation systems. The purpose of the competition was to develop better models for rating prediction and suggest methods for incorporating social media data into the prediction models. The goal of our project was to use the information disseminated by the top competitors of the contest to develop new algorithms for recommendation systems. Noting the prevalence of ensemble methods employed on factorization models, our team developed a doubly ensemble framework named TBEEF, a software framework with a plugin interface through which factorization and ensemble models could be easily developed and ran. Our results showed that TBEEF performed adequately on the Baidu dataset, though it did not perform as well as the models from the top competitors.

Contents

Chapter 1

Problem Statement

Recommender systems have become commonplace in recent years. Many internet companies have been actively funding the development of recommender systems for the purposes of increasing revenue. For example, Amazon uses a recommender system to recommend products to users as a way to tempt them into buying more goods [16]. Pandora Radio uses recommender systems to find new music for users in order to keep users interested and increase ad revenue. Netflix utilizes recommender systems in order to find new movies for subscribers in order to keep their attention. Google implements recommendation systems to give users the most useful search options in hopes that good recommendations will keep users in its ecosystem [12].

Baidu, Inc. wishes to use recommender systems to more actively keep the attention of users of its video service. By doing so, Baidu hopes to increase ad revenue and pull users into its ecosystem of products. This project will help to improve the accuracy of the recommender systems and thus help Baidu, Inc. maximize the attention it can receive out of its users.

Thus in the spring of 2013, Baidu, Inc. hosted a competition for teams to develop new algorithms for movie recommendation systems [2]. Baidu provided the teams with an example dataset of user ratings and information about the users regarding their interactions on social networks. The competition was divided into two problems:

1. Develop a model which most accurately predicts user ratings.
2. Incorporate social media data to improve user rating predictions.

For the first project, Baidu provided the competitors with data on approximately 10,000 users for around 8,000 movies. The contestants were to train a prediction model and test its predictions against a test dataset by measuring the RMSE (root mean squared error) of the model's predictions on a test dataset. The goal was to develop a model which most accurately predicts the user's ratings for movies not included in the training dataset, and thus minimize the RMSE of the model's predictions on the test dataset.

For the second project, Baidu provided the teams with a dataset of about 15 million users with around 15,000 external records, such as film tag information and social media information for the users (such as tweets, friends, etc.). Each team wrote a report on the different ideas on how to utilize this information to improve movie recommendations and deal with associated problems, including:

1. How tags could be used to overcome the rating sparseness problem.
2. Feature-engineering techniques for including social information in current recommendation models.

At the end of the competition, the top competitors we invited to present their implementation techniques. Our project, sponsored by Baidu Inc., was to utilize the knowledge gained from the competitors of the competition to develop a new algorithm for movie recommendation. To do so, our team developed a program, TBEEF (Triple Bagged Ensemble Ensemble Framework), which is a robust framework for utilizing an ensemble of prediction models and ensemble techniques for prediction. A plugin interface was made to let users easily add extra prediction models and ensemble techniques to used in a doubly ensemble fashion, an extension of the methods employed by the top teams of the Baidu competition. As this program is open-source, this gives the general public an easy manner to implement the ensemble frameworks employed by the winners of the competition.

Chapter 2

The Statistical Models

Though a wide variety of statistical methods were employed in this project, the techniques can be summed up in two distinct categories:

1. Factorization Models
2. Ensemble Models

Factorization models are statistical models specifically designed to solve problems of low rank approximation with sparse data. This is important since the data matrix that is the input to movie prediction problems is sparse since most users will rate a very small subset of the total movies. These models take the sparse matrices and utilize latent factors to form a predictive model. The output of these models is a vector of predictions. When using an ensemble of statistical models as in TBEEF, these vectors of predictions are then combined to create a matrix of predictions of which to be reduced to a single prediction for each user movie pair (each row). To do so, another set of statistical models, ensemble models, are employed. These differ from the factorization models since this stage of the program does not have sparse matrices but rather complete information, and thus statistical models fit to use the complete information are employed at this stage.

2.1 Factorization Models

Although many factorization models are employed (and can be implemented) in TBEEF, all of these models fall into one of two frameworks:

1. Feature-Based Matrix Factorization
2. Factorization Machines

These were chosen because most of the common statistical models for prediction, such as Nearest Neighbor and Implicit Feedback models, could be specified within the framework of Feature-Based Matrix Factorization and Factorization Machines. Since these two frameworks already have programs for training and prediction (SVDFeature and libFM respectively), the work of the user is to develop an algorithm to write the proper input into these programs in order to train and predict from the model. First we will describe the intuition for using a factor matrix to solve sparse prediction problems. Then we will describe the two model frameworks, some methods for developing statistical models within the frameworks through feature-engineering, and the optimization techniques used to train models in these frameworks.

2.1.1 Factor Matrix Intuition

The purpose for using a factor matrix is best understood through an example [19]. Say you wish to estimate the interaction between the movies *Alice* and *Star Trek* to better predict the ratings. Assume that no user has seen both movies. Thus the standard regression models with interactions would estimate the effect of the interaction as 0. But with the factorized interaction parameters, we can estimate the interaction using the movie interactions via the dot products of their factors, $\langle v_{Alice}, v_{StarTrek} \rangle$. Many user's ratings for a movie like *Star Wars* would be correlated with user ratings for *Star Trek*, and thus *Star Wars* and *Star Trek* will have similar interaction effects for most movies. Since for most movies j, $\langle v_j, v_{StarTrek} \rangle \approx \langle v_j, v_{StarWars} \rangle$, it follows that $v_{StarWars} \approx$

$v_{StarTrek}$. Therefore $\langle v_{Alice}, v_{StarTrek}\rangle \approx \langle v_{Alice}, v_{StarWars}\rangle$. Thus an FM is able to obtain a good estimate of interaction effects for which we have little or no data by using the factor matrix.

2.1.2 Feature-Based Matrix Factorization

By using a pre-existing solver (SVDFeature) for Feature-Based Matrix Factorization, one can test new models simply by defining new features without having to write new code. Features allow us to incorporate additional information about the users, items, or global relationships to improve our predictive success. The basic matrix factorization model is

$$\hat{y}_{ui} = \mu + b_u + b_i + p_u^T q_i.$$

where μ is the global mean movie rating, b_u is the mean effect of user u, b_i is the mean effect of item i, and p_u, q_i are the latent factors of user u and item i. Generalizing this, we get Feature-Based Matrix Factorization:

$$\hat{y}_{ui} = \mu + \sum_{g\in G} b_g\gamma_g + \sum_{m\in M} b_m^u\alpha_m + \sum_{n\in N} b_n^i\beta_n + p_u^T q_i.$$

Here G, M, N are the matrices of global, user and item features γ, α, β respectively. SVDFeature finds the weights b_i and the latent factors p, q through stochastic gradient descent. Notice that if we let $G = \emptyset, \alpha = 1$ when $k = u$ or 0 when $k \neq u, \beta_k = 1$ when $k = i$ or 0 when $k \neq i$ in our input vector x_k, then the model simplifies to the basic matrix factorization problem.

Included SVD Feature Models for the Baidu Dataset

The following models are included by default into TBEEF to show the flexibility of the plugin interface in setupFeatures(). These models utilize both basic features and extra datasets (whose paths are specified in utils and processed in preProcess.py) in order to generate statistical models with the relevant social media data.

Basic

The basic model does not incorporate any feature engineering. It only uses the user id, the movie id, and the interaction effect between them. Mathematically, the basic model can be written as

$$\hat{y}_{u,i} = \mu + b_u + b_i + p_u^T q_i.$$

Movie Tags Neighborhoods

Pairs of movies which share a number of tags above a threshold are considered to be similar. Thus we can define the neighborhood of a movie as the set of all movies in which one of the movies share at least the threshold number of movie tags with another. The formula for this model is

$$\hat{y}_{u,i} = \mu + b_u + b_i + \sum_{j\in N(u,i)} (y_{u,j} - b'_u)w_{i,j} + p_u^T q_i,$$

where b'_u is the average of all ratings a particular user has given.

Social Information Neighborhoods

We assume that users who follow another user do so because they share tastes. This would result in these two users giving similar ratings to a given movie. A neighborhood model incorporating this idea can be written as

$$\hat{y}_{u,i} = \mu + b_u + b_i + \sum_{j\in N(u,i)} (y_{v,i} - b'_i)w_{v,u} + p_u^T q_i.$$

User History Implicit Feedback

Users watching history can be assumed to be able to contribute to the performance of recommendation system, regardless of whether the user like or dislike the movie, using an implicit feedback model. The formula for such a model is

$$\hat{y}_{u,i} = \mu + b_u + b_i + \sum_{j \in I(u)} \alpha_{u,x} b_x^{'} + \left(p_u + \sum_{j \in I(u)} \alpha_{u,x} v_x \right)^T q_i,$$

where I(u) is the set of movies that users have already watched and $\alpha_{u,x}$ is the feature value of implicit feedback features.

User History Neighborhoods

Pairs of users who view (though may not watch) similar movies are more likely to have similar interest. Thus we can apply a neighborhood model of the form

$$\hat{y}_{u,i} = \mu + b_u + b_i + \sum_{j \in N(u,i)} (y_{v,i} - b_i^{\prime}) w_{v,u} + p_u^T q_i.$$

2.1.3 Factorization Machines

One of the methods most widely used by top competitors in the Baidu competition was Factorization Machines (FMs). FMs are a generalization of Matrix Factorization techniques in the sense that the commonly used Matrix Factorization models are able to be expressed in the FM framework using feature engineering [18, 19].

A FM can be intuitively understood as a linear regression model with interaction effects where the coefficients for the interaction effects are "smarter" and take into account the correlations between variables. Formally, the factorization machine of degree 2 defined by the model:

$$\hat{y} = w_0 + \sum_{j=1}^{m} w_j x_j + \sum_{j=1}^{m} \sum_{k=j+1}^{m} \langle v_i, v_j \rangle x_i x_j,$$

where

$$w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}, \mathbf{V} \in \mathbb{R}^{n \times k},$$

are the parameters to be estimated. $\langle v_i, v_j \rangle$ is defined as the dot product between the i-th and the j-th rows of the factor matrix $\mathbf{V}$. This model can be summed up as:

1. w_0, the global bias.
2. w_j, the base effect of the i-th parameter.
3. $\hat{w_{i,j}} := \langle v_i, v_j \rangle$, the effect of the interaction.

The FM model can easily be expanded to include the higher order interaction terms, leading to a degree d FM in which $\hat{y}$ is written as a polynomial of deg d in the variables x_i with i running from 1 to p.

There are three optimization techniques implemented by the commonly used package, libFM, to train FM models [19]. These are:

1. Stochastic Gradient Descent.
2. Alternating Least Squares.
3. Markov Chain Monté Carlo.

The details of these techniques are discussed in a later section.

Included libFM Models for the Baidu Dataset

The following models are included by default into TBEEF to show the flexibility of the plugin interface in setupFeatures(). These models utilize both basic features and extra datasets (whose paths are specified in utils and processed in preProcess.py) in order to generate statistical models with the relevant social media data.

Basic

The basic model does not incorporate any feature engineering. The input vectors from our sparse matrix are given using only user IDs u and movie IDs i.

$$(u, i) \rightarrow \mathbf{x} = (\underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|\mathbf{U}|}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|\mathbf{I}|}).$$

The formula for this model is

$$\hat{y}(x) = w_0 + w_u + w_i + \sum_{f=1}^{k} v_{u,f} v_{i,f}.$$

Nearest Neighbor

In this model we incorporate all the rating data from each user into each row vector. The user u is implicit; rather we use the movie ID i, and a set of tuples that include all other movies rated by the same user. Thus r_j is the rating for some l_j movie rated by the user. This info is included in the input vector with a value of the rating divided by the total number of movies rated by the user, m.

$$(i, \{(r_1, l_1), \ldots, (r_m, l_m)\}) \rightarrow \mathbf{x} = (\underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|\mathbf{I}|}, \underbrace{0, \ldots, r_1/m, 0, \ldots, r_m/m, \ldots, 0}_{|\mathbf{I}|}).$$

The formula for this model is

$$\hat{y}(x) = w_0 + w_i + \frac{1}{m} \sum_{j=1}^{m} r_j w_{l_j} + \frac{1}{m} \sum_{j=1}^{m} \langle v_i, v_j \rangle + \frac{1}{m^2} \sum_{j=1}^{m} \sum_{j'>j}^{m} r_j r_{j'} \langle v_{l_j}, v_{l_{j'}} \rangle.$$

Movie Tags

Our first model to includes movie tag data t by using it as a categorical indicator: 1 if the movie is associated with the tag t and 0 otherwise.

$$(u, i, t) \rightarrow \mathbf{x} = (\underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|\mathbf{U}|}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|\mathbf{I}|}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|\mathbf{T}|}).$$

This yields the model

$$\hat{y}(x) = w_0 + w_u + w_i + w_t + \sum_{f=1}^{k} v_{u,f} v_{i,f} + \sum_{f=1}^{k} v_{u,f} v_{t,f} + \sum_{f=1}^{k} v_{i,f} v_{t,f}.$$

Related Movie Tag Threshold

Here we use the movie tag data to determine how many tags each movie pair shares. We set the threshold h, and for each user u, movie i, we include data for all movies that share at least h tags with i. The value we assign here is n_j/m, where m is the maximum number of tags shared between any two movies in the set and n_j is the number of tags shared between movie i and l_j.

$$(u, i, \{(n_1, l_1), \ldots, (n_m, l_m)\}) \rightarrow \mathbf{x} = (\underbrace{0, \ldots, 1, 0, \ldots}_{|\mathbf{U}|}, \underbrace{0, \ldots, 1, 0, \ldots}_{|\mathbf{I}|}, \underbrace{0, \ldots, n_1/m, 0, \ldots, n_m/m, \ldots, 0}_{|\mathbf{I}|}).$$

We then have a neighborhood model given as

$$\hat{y}(x) = w_0 + w_u + w_i + \frac{1}{m}\sum_{j=1}^{m} n_j w_{l_j} + \frac{1}{m}\sum_{j=1}^{m}\langle v_i, v_j\rangle + \frac{1}{m^2}\sum_{j=1}^{m}\sum_{j'>j}^{m} n_j n_{j'}\langle v_{l_j}, v_{l_{j'}}\rangle.$$

Related Movie Tag Threshold 2

Similar to the model above, this model uses the movie tag data. However, while this model uses the shared tag threshold, it only includes movies rated by the user u. So for each movie l_j that shares at least h tags with movie i and has been rated by user u, we now assign a value of r_j/m where m is the total number of movies rated by user u and r_j is the rating.

$$(u, i, \{(r_1, l_1), \ldots, (r_m, l_m)\}) \to \mathbf{x} = (\underbrace{0, \ldots, 1, 0, \ldots}_{|\mathbf{U}|}, \underbrace{0, \ldots, 1, 0, \ldots}_{|\mathbf{I}|}, \underbrace{0, \ldots, r_1/m, 0, \ldots, r_m/m, \ldots, 0}_{|\mathbf{I}|}).$$

User History

Using data collected about history for each implicitly defined user, we give each movie viewed in the history a r_j/m value where m is the total number of movies in the history and r_j is either the rating of the movie l_j or the user never rated it, then simply a 1.

$$(i, \{(r_1, l_1), \ldots, (r_m, l_m)\}) \to \mathbf{x} = (\underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|\mathbf{I}|}, \underbrace{0, \ldots, v_1/m, 0, \ldots, v_m/m, \ldots, 0}_{|\mathbf{I}|}).$$

User Social

Using social data collected about our users, we construct our input vectors using user ID u, movie ID i, and a set of friends S. Each friend in the vector is given a $1/m$ value where m is the total number of friends for the given user.

$$(u, i, \{s_1, \ldots, s_m\}) \to \mathbf{x} = (\underbrace{0, \ldots, 1, 0, \ldots}_{|\mathbf{U}|}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|\mathbf{I}|}, \underbrace{0, \ldots, 1/m, 0, \ldots, 1/m, \ldots, 0}_{|\mathbf{U}|}).$$

Mathematically, this is represented by the model

$$\hat{y}(x) = w_0 + w_u + w_i + \langle v_u, v_i\rangle + \frac{1}{m}\sum_{j=1}^{m}\langle v_i, v_{s_j}\rangle + \frac{1}{m}\sum_{j=1}^{m} w_{s_j} + \frac{1}{m}\sum_{j=1}^{m}\langle v_u, v_{s_j}\rangle + \frac{1}{m^2}\sum_{j=1}^{m}\sum_{j'>j}^{m}\langle v_{s_j}, v_{s_{j'}}\rangle.$$

2.1.4 Optimization Techniques Employed

Machine learning processes are defined by a model which defines the best prediction $\hat{y}$ given a set of data. Machine learning models are evaluated by considering the loss function [20, 11]

$$l(y, \hat{y}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

The goal of a machine learning algorithm is to find the parameters which minimize the prediction error as measured through the loss function. There are three commonly used computational methods for minimizing a general loss function for large data problems: Stochastic Gradient Decent, Alternating Least Squares, and Markov Chain Monté Carlo (MCMC).

Stochastic Gradient Descent

During the process of machine learning, consider the problem of minimizing the objective function

$$Q(w) = \sum_{i=1}^{n} Q_i(z_i, w),$$

where w is to be estimated and each summand function $Q_i(w)$ is associated with the i-th observation z_i in the data set [4].

For this problem, the standard iteration method is to use the scheme

$$w_{k+1} = w_k - \lambda \sum_{i=1}^{n} \nabla Q_i(z_i, w_k),$$

where λ is the step size and $\nabla Q_i(z_i, w_k)$ is the direction. Under sufficient regularity constraints (sufficiently small λ and a sufficiently close initial estimate) this algorithm achieves linear convergence, that is $-\log \hat{r} \sim k$ where $\hat{r}$ stands for the residual error and k is the number of iterations.

We can improve the accuracy of this calculation by specifying λ as a positive definite matrix M_t that approaches the inverse of the Hessian matrix of Q. This is a variant of the well-known Newton's method whose convergence $-\log \log \hat{r} \sim k$.

However, when the data set is enormous, evaluating the large sums of gradients is computationally expensive. To increase the computational efficiency, at each iteration we can estimate the gradient on the basis of a randomly picked observation z_i. This defines the iterative process for stochastic gradient descent:

$$w_{k+1} = w_k - \lambda \nabla Q(z_k, w_k).$$

Note that $\{w_k\}$ is a stochastic process which depends on the random order that the observations are chosen.

The expectation of the residual error decreases as $E(\hat{r}) \sim k^{-1}$. If regularity assumptions are relaxed, the theory suggest slower asymptotic convergences rates, typically around $E(\hat{r}) \sim k^{\frac{-1}{2}}$.

Even though stochastic gradient descent methods have a slower convergence rate, in large scale-learning problems where the limiting factor is the computing time rather than the number of observations, stochastic gradient decent can be preferred above other methods due to it being less computationally taxing.

Alternating Least Squares

The intuition behind the alternating least squares method is that the optimal parameter set can be found by changing only one parameter at a time [23]. To perform alternating least squares, the procedure is as follows:

1. Choose a parameter γ from the set of parameters.
2. Set all parameters constant except for γ.
3. Minimize the cost function with respect to γ.
4. Repeat.

The advantage of such method is that it does not replace objective function by an approximation. This method achieves a speedup by applying mathematical simplification. Since alternating least squares is not sensitive to the order of training examples, it can be more robust than stochastic gradient descent in circumstances where the data is not symmetric.

Markov Chain Monté Carlo

Markov Chain Monté Carlo is utilized to estimate the posterior distributions of the parameters in statistical models estimated in a Bayesian framework [13]. To estimate the models using a Bayesian framework, it is common for the model's parameters θ to be defined with a Gaussian hyperprior on each parameter mean and a Gamma hyperprior on each parameter precision. Since y is given by the statistical model $y(\mathbf{X})$, the likelihood is defined by the function $y(\mathbf{X})$ of θ and $\mathbf{X}$. Thus by defining the priors on θ, we have defined the prior distribution of y. Using Bayes' formula,

$$Posterior(y) \propto Likelihood(y) \times Prior(y),$$

we can receive the posterior distribution for y. However, in many cases this posterior distribution must be found numerically. Markov Chain Monté Carlo (MCMC) is the method for estimating this distribution. There are many different variants of MCMC algorithms such as the Metropolis Algorithm and Gibbs Sampling. The program which implements an MCMC optimization, libFM, utilizes single parameter Gibbs Sampling. A thorough discussion of this technique is outside the scope of this paper, though the exact implementation for libFM can be found in Freudenthaler et. al. (2011) [8]. It is shown elsewere that for factorization machines, this algorithm is as efficient as SGD and ALS but has the benefit of estimating the regularization hyperparameters, making it the preferred estimation method for libFM [19].

2.2 Ensemble Models

The ensemble models are statistical models without having to worry about problems due to sparsity. The following models fall into this category and are implemented into TBEEF:

1. Ordinary Least Squares Regression (OLS)
2. Ordinary Least Squares Regression with Interaction Effects
3. Ridge Regression
4. Lasso Regression
5. Bagged Regression Trees
6. Random Forest Regression
7. Conditional Inference Random Forest Regression
8. Gradient Boosted Regression Trees
9. Bayesian Model Averaging Regression

2.2.1 Ordinary Least Squares, Ridge, and Lasso Regression

Linear Regression is defined by the model

$$\mathbf{Y} = \mathbf{X}\beta,$$

where $\mathbf{Y}$ is an $n \times 1$ vector of observed results, β is an $m \times 1$ vector of parameters β_i, and $\mathbf{X}$ is an $n \times m$ matrix of the observed data constructed by $1 \times m$ row vectors x_i of observations [11]. This can also be described using the summation formulation

$$y_i = \beta_0 + \sum_{j=0}^{m} x_{i,j}\beta_j.$$

The model with k interaction effects is described by the complete polynomial of order k. The goal of the model is to find an estimator vector $\hat{\beta}$ whose predictions

$$\hat{y_i} = x_i\hat{\beta},$$

accurately predict a result y_i from an observation x_i.

The most common estimator for a linear regression model is known as Ordinary Least Squares which estimates the model by finding the parameter vector $\hat{\beta}^{OLS}$ which minimizes the loss function

$$l := ||\mathbf{Y} - \mathbf{X}\beta||^2.$$

In the case of Ordinary Least Squares regression, there is a known closed form solution for the estimators:

$$\hat{\beta}^{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}.$$

For Ordinary Least Squares Regression with interaction effects, the interaction effect variables can be computed as new variables and added to $\mathbf{X}$ and then the Ordinary Least Squares estimator on this new input matrix will result in the estimator for the model with interaction effects. Ridge Regression estimates are defined as the parameter vector $\hat{\beta}^{ridge}$ which minimizes the loss function:

$$l = ||\mathbf{Y} - \mathbf{X}^T\beta||^2 - \lambda||\beta||^2,$$

where λ is a constant. Luckily, as in the case of Ordinary Least Squares regression, there is a known closed form solution to the problem:

$$\hat{\beta}^{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T y.$$

Lasso Regression estimates are defined as the parameter vector $\hat{\beta}^{lasso}$ which minimizes the loss function:

$$l = ||\mathbf{Y} - \mathbf{X}^T\beta||^2 - \lambda||\beta||,$$

where λ is a constant. However, in the case of Lasso Regression, there is no closed form for the estimator, though standard optimization techniques such quadratic programming can be used to solve for the estimator [11]

2.2.2 Advantages of the Regression Techniques

Ordinary Least Squares Regression with interactions can lead to better predictions than standard Ordinary Least Squares Regression when there is reason to believe that the variables may be related. For example, if two of the prediction models utilize the user history data, then their predictions (inputs to the ensemble techniques) may be correlated which would in turn lead the interaction effect model to better predictions over the non-interaction model. Ridge Regression estimators are also known as shrinkage estimators because by definition it places an additional cost on the parameters over Ordinary Least Squares regression, which in turn causes the parameter estimations to "shrink" to zero. Ridge Regression is very useful if the correlation between different estimators is strong. Under this situation, the determinant of $\mathbf{X}^T\mathbf{X}$ is close to zero and thus the estimator for Ordinary Least Squares regression does not have a unique solution since $\mathbf{X}^T\mathbf{X}$ does not have an inverse. On the other hand, Ridge Regression always provides a unique solution because the matrix we need to invert cannot have a determinant near zero [14].

Ridge Regression is preferred in the case of prediction problems because there is a mathematical theorem which states that there is a λ such that the corresponding Ridge Regression estimator will give better predictions than the Ordinary Least Square's estimator [5]. This can be formalized as follows. Define the Mean Squared Error (MSE) as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2.$$

Note that the MSE is a measurement of the average prediction error. The following theorem holds:

Theorem 2.1 ***Existence Theorem.*** *There exists a λ such that the MSE of $\hat{\beta}^{ridge}$ is less than the MSE of $\hat{\beta}^{OLS}$*

Thus for prediction problems, Ridge Regression is preferred because there always exists a Ridge Regression estimator that gives more accurate predictions. The problem is finding the correct λ. For the purposes of TBEEF, our implementation utilized the method for choosing λ developed by Cule et. al [9].

Lasso Regression has the advantage that for large enough λ some coefficients are set to exactly zero, which can be an indicator of bad predictability in the model.

2.2.3 Regression Trees

Linear regression is a global model where a single prediction formula is used over the entire dataset [21, 11]. When the data has many features and the interactions are non-linear, a global model may not be the correct description of the data. Regression trees iteratively partition the dataset into similar portions on which prediction models are run. Prediction trees can be used to represent this recursive partitioning. An example can be seen in Figure 2.1. A given row in the data matrix is then ran through the partitioning algorithm to arrive at the proper terminal node (referred to as the data belonging to the terminal node) from which the terminal node's prediction algorithm is ran.

The tree is iteratively developed using an entropy minimization technique. For a given tree T, let K be the set of terminal nodes of T and let C be the set of data points which belong to the terminal node c. The entropy S is defined as

$$S = \sum_{c \in K} \sum_{i \in C} (y_i - p_c)^2,$$

where p_c is the prediction for the leaf c. A common prediction function is to use the average within the leaf, that is

$$p_c = \frac{1}{|C|} \sum_{i \in C} y_i.$$

Two inputs are required for the algorithm, thresholds δ and q. The recursive algorithm for generating the tree is as follows:

1. Start with a single node for all points.
2. If all points have the same value of S for the independent variable, stop. Else, check all binary splits and choose the one which reduces S the most. If the largest decrease is less than the δ threshold or one of the nodes contains less than q points, stop. Otherwise, take the split.
3. Repeat step 2.

Regression Tree Model Distinctions

Each of the regression tree models implemented is distinct in its own manner. The Bagged Regression Trees utilizes the Bagging (Bootstrap Aggregation) technique described later on a standard regression tree model. In random forest models, the model is applied on bagged copies though an additional form of randomization is applied by utilizing a random subset of the predictors for the splitting step (Step 2) [15]. Conditional Inference models utilize a more advanced splitting technique than the one described earlier [22]. Gradient Boosted trees repeatedly apply the regression tree on the residuals to obtain a very complex fitting model (cross-validation techniques such as bagging and K-fold cross-validation are used to choose the number of boosting iterations to prevent overfitting).

Cross-Validation Techniques

There are two cross-validation techniques utilized by the random forest implementations. These are bagging (bootstrap aggregating) and K-fold cross-validation. The bagging algorithm is as follows [17]:

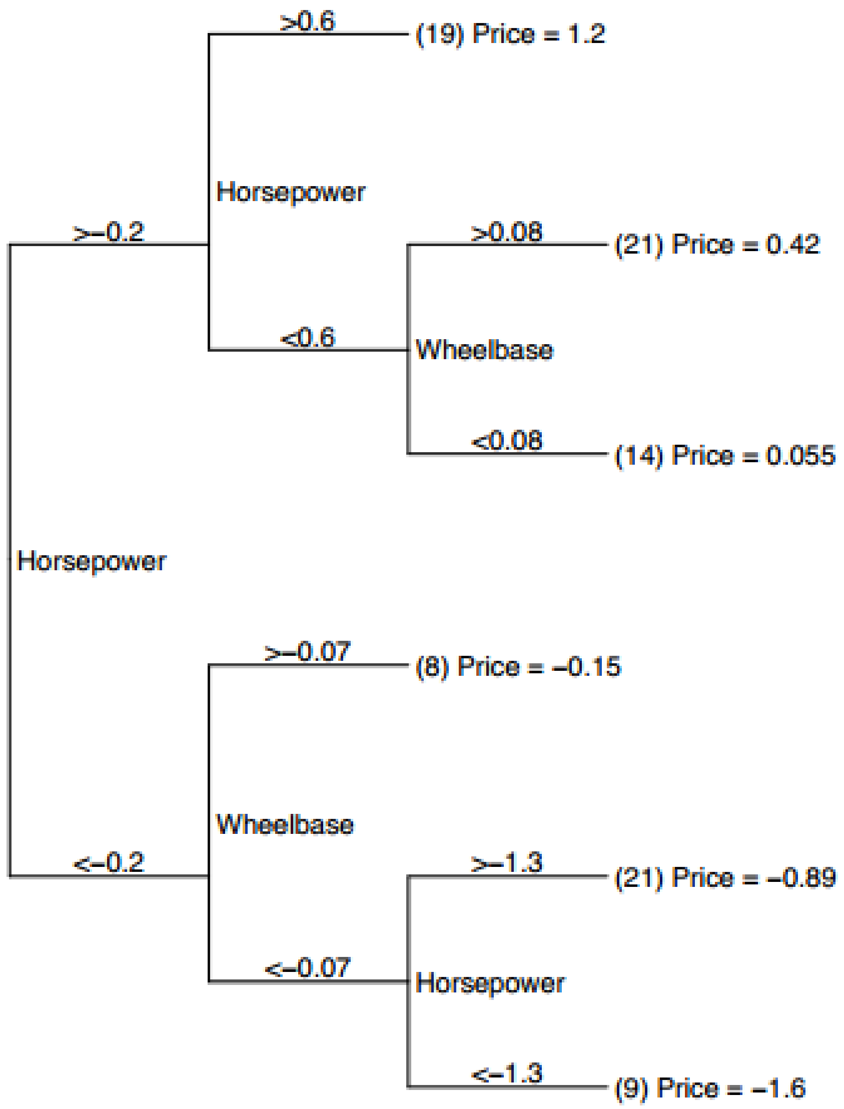

Figure 2.1: An example of a regression tree [21].

1. Start with a data source of size N.
2. Sample with repetition to create M datasets of size N' following the same distribution of the original dataset (Bootstrap).
3. Train the model on each of the M datasets and calculate the RMSE on the data not in the specific training set.
4. Choose the model with the lowest RMSE.

The K-fold cross-validation performs is performed similarly but does not utilize randomization:

1. Split the data into K subsets.
2. Train the model on $K-1$ sets, calculate the RMSE on the subset that is left out.
3. Repeat K times, leaving out each set exactly once.
4. Choose the model with the lowest cross-validation RMSE.

2.2.4 Bayesian Model Averaging

Consider the linear regression model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \epsilon.$$

For k regressors there exist 2^k different combinations of the variables. We index these combinations by M_j for $j = 1, 2, \ldots, 2^k$. Given this setup, the posterior distribution for the h^{th} coefficient, β_h is

$$\Pr(\beta_h|D) = \sum_{j:\beta_h \in M_j} \Pr(\beta_h|M_j)\Pr(M_j|D),$$

where the model's posterior probability, $\Pr(M_j|D)$ is used as a weight [1]. The posterior model probability of M_j is found by the ratio of its marginal likelihood and the sum of the marginal likelihoods over the entire model space (Bayes' Formula) as follows:

$$\Pr(M_j|D) = \frac{\Pr(D|M_j)\Pr(M_j)}{\sum_{i=1}^{2^k}\Pr(D|M_i)\Pr(M_i)}.$$

The estimators are defined as the expected values of posterior distributions, that is

$$E[\hat{\beta}|D] = \sum_{j=1}^{2^k} \hat{\beta} Pr(M_j|D),$$

and the standard errors are given by the variance of the posterior distributions

$$V[\hat{\beta}|D] = \sum_{j=1}^{2^k} (Var[\beta|D, M_j] + \hat{\beta}^2)\Pr(M_j|D) - E[\beta|D]^2.$$

Chapter 3

Introduction to TBEEF

When one looks at the results of the Baidu competition, one quickly realizes that all of the top competitors utilized ensemble methods on a large variety of models. However, each of the teams had different ensemble methodologies, leading to an ensemble of different ensembling techniques. Our goal was to implement a model which could use an ensemble of ensemble techniques to see if this would lead to better predictions than simply using one ensemble technique.

Rather than writing a single-use script to perform the analysis, our team decided to create a software package for developing prediction algorithms utilizing this doubly ensemble framework. This program is called TBEEF, Triple Bagged Ensemble Ensemble Framework. The program is pictured in Figure 3.1. It has three steps:

1. Model
2. Hybrid
3. Synthesize

To begin each step, the data is bagged without replacement to create a training and a cross-validation set. Previous research shows that this division of the data has equivalent properties to bagging with replacement (standard bagging), and thus bagging without replacement is used in order to build a complete matrix of predictions for the following steps [6].

The Model step utilizes the factorization models to generate predictions, of which are aggregated into a prediction matrix. The cross-validation set of the Model step is utilized to create a training matrix in the Hybrid step since the program has access to the real values for the independent variable (as this set is created from the leftovers of the bagged without replacement training set). The Hybrid step is the use of an ensemble of ensemble algorithms for prediction from the Model step predictions, and thus the training matrix is bagged without replacement and the Hybrid model is ran. The cross-validation set of the Hybrid step is used to then create the training matrix for the Synthesize step as before. For the Synthesize step, a single ensemble algorithm is used to ensemble the predictions from the Hybrid step. Using a cross-validation set, a cross-validation RMSE is calculated at the end of the Synthesize step. To ensure that the data is actually bagged without replacement instead of simply randomly split, this entire algorithm is ran K times and the trial with the lowest Synthesize cross-validation RMSE is chosen as the best trial.

A plugin interface has been implemented to create factorization models for the Model step and ensemble models for the Hybrid and Synthesize steps. For more information, please see the program documentation in the appendix.

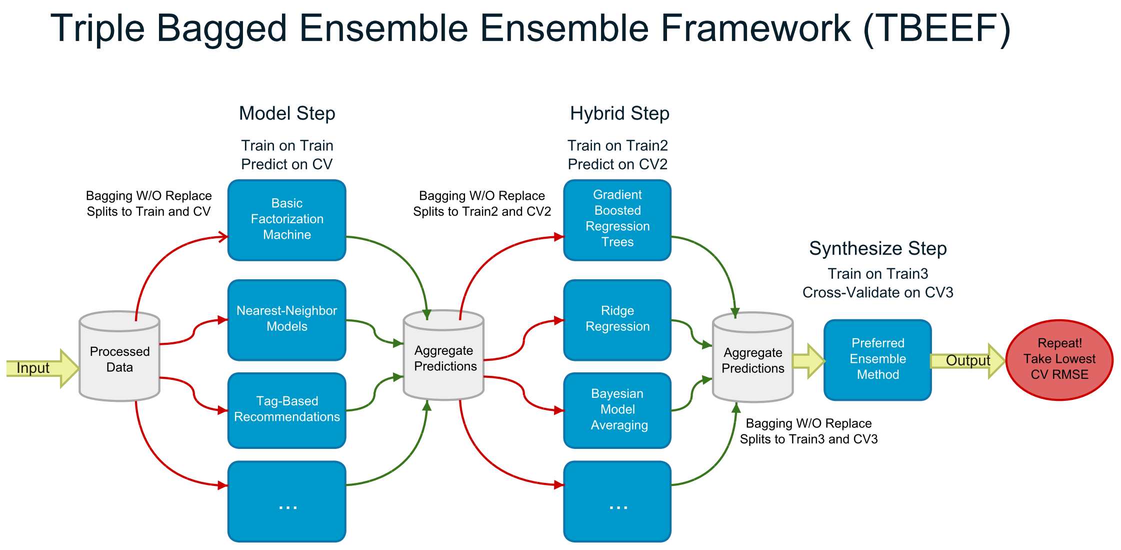

Figure 3.1: Diagram of the TBEEF program. Shown are the three steps, the Model step, the Hybrid step, and the Synthesize step. Example models are shown in the Model and Hybrid steps.

Chapter 4

TBEEF Performance on the Baidu Dataset

The results of running TBEEF on the Baidu dataset using the default configuration is shown in the following tables. Table 4.1 shows the results of the factorization models. The left column shows the models and the middle column shows the RMSE. The rightmost column shows the results of the second place team's comparable models [24]. Our factorization model results do not compare favorably to the results of the top competitors. For example, our implicit feedback model received an RMSE of .6439 while Pandas' implicit feedback model received an RMSE of .6046. Table 4.2 shows the results of the ensemble models. These do not compare well with the RMSEs of the competitors after ensembling (their final RMSEs, the top being .5913, .5932, .5955) due to the performance of the factorization models. Note that the ensemble model that fared best, the gradient boosted regression trees, was the favored ensembling technique of the contestants which gives some empirical evidence towards the idea that gradient boosted regression trees may be the best ensembling technique for prediction problems. The final RMSE was .6263. This was higher than the singly ensemble models; this shows that the use of the doubly ensemble method may have lead to overfitting. These results may suggest that for this technique to be useful a larger dataset may be required.

However, the large difference in our base model's prediction abilities is very likely the cause of the final RMSE difference from the top competitors since the ensembled RMSEs are close (within .1) to the RMSEs of the factorization models, indicating that the factorization modeling portion has the largest effect on the final outcome. However, the difference in the final RMSEs are still very small. This is because the models with social information did not have greatly increase the model's prediction accuracy, a finding that matches the results of the top competitors [3, 10, 24].

We wish to note the computability of this model. Many of the factorization models implemented in LibFM, for example Nearest Neighbor Models, were not shown in these results due to their runtime being on the order of a few days (5-7). This is on the Baidu dataset of 10,000 users, 8,000 movies, and 15,000 external records. This means that for practical purposes the LibFM models may not be viable. On the other hand, the SVDFeature models ran much faster, completing similar models within hours. The ensemble methods completed within the same time frame. This shows that to scale TBEEF to larger datasets (which would be required to stop the overfitting seen in the doubly ensemble results), one would probably need to stay with feature engineering for SVDFeature to due time constraints. Lastly, none of the random forest ensemble models were able to be ran on the full dataset given the large memory requirement of these models. This shows that the random forest methods may not be applicable to practical applications of movie prediction algorithms.

Factorization Model	TBEEF Default RMSE	PANDAS RMSE
LibFM Basic	.6355	.6449
LibFM Movie Tag Neighborhoods	.8129	.6069
LibFM Social Information Neighborhoods	.6379	NA
SVDFeature Basic	.6307	.6204
SVDFeature Implicit Feedback	.6439	.6046

Table 4.1: RMSEs for the Factorization Models

Ensemble Model	TBEEF Default RMSE	PANDAS RMSE
Bayesian Model Averaging	.6237	.5932
Bagged Regression Trees	.6377	.5932
Gradient Boosted Regression Trees	.6235	.5932
Lasso Regression	.6250	.5932
Ordinary Least Squares w/ Interactions	.6237	.5932
Ordinary Least Squares	.6236	.5932
Ridge Regression	.6236	.5932

Table 4.2: RMSEs for the Ensemble Models

Chapter 5

Conclusion

Prediction algorithms have become important in the online-era due to the money that could be gained through successfully targeting users with the information collected through user activities. The most common methods utilized ensemble techniques on factorization models. Our team developed a doubly ensemble framework to utilize not only an ensemble of factorization models, but also an ensemble of ensemble methods. What resulted was the program TBEEF, a flexible software package through which factorization models and ensemble methods could be implemented. TBEEF's adequate performance showed that larger datasets may be necessary for doubly ensemble techniques to be effective.

Appendix A

TBEEF Documentation

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.

A.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.

A.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.

A.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.

A.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 : $1 \implies$ Use global bias in factorization machine. 1 by default.
  - oneWay : $1 \implies$ 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.

A.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.

A.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.

A.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.

A.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.

A.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.

A.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).

Selected Bibliography Including Cited Works

[1] Shahram Amini and Christopher F. Parmeter. Bayesian model averaging in r. Technical report, 2011.

[2] Baidu, Inc. Movie recommendation algorithm contest website.

[3] Stars Wang (bluejazz). Baidu movie recommendation competition. In *Baidu, Inc. Movie Recommendation Algorithm Contest Showcase*. Baidu, Inc., 2013.

[4] Léon Bottou. Stochastic Gradient Descent Tricks. In Grégoire Montavon, GenevièveB Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 421–436. Springer Berlin Heidelberg, 2012.

[5] Patric Breheny. Ridge regression. In *BST 764: Applied Statistical Modeling for Medicine and Public Health*. University of Kentucky, 2011.

[6] Andreas Buja and Werner Stuetzle. Observations on bagging.

[7] Lu Zhang-Yu Chen, Zheng. Feature-based matrix factorization. Technical Report MSU-CSE-00-2, APEX, Shanghai, China, July 2011.

[8] Steffen Rendle Christoph Freudenthaler, Lars Schmidt-Thieme. Bayesian factorization machines. In *Workshop on Sparse Representation and Low-rank Approximation, Neural Information Processing Systems (NIPS-WS)*.

[9] Erika Cule and Maria De Iorio. A semi-automatic method to guide the choice of ridge parameter in ridge regression. May 2012.

[10] Yuyu Zhang (frank0523). Rating prediction with hybrid models. In *Baidu, Inc. Movie Recommendation Algorithm Contest Showcase*. Baidu, Inc., 2013.

[11] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[12] John Riedl Joseph A. Konstan. Deconstructing recommender systems, 2012.

[13] John K Kruschke. *Doing Bayesian data analysis : a tutorial with R and BUGS*. Academic Press, Burlington, MA, 2011.

[14] Jia Li. Linear, ridge regression, and principal component analysis. In *STAT557/IST557: Data Mining Course Material*. The Pennsylvania State University, 2010.

[15] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[16] Richard MacManus. A guide to recommender systems, 2009.

[17] Mehryar Mohri. Introduction to machine learning, lecture 12. In *Courant Institute and Google Research*.

[18] Steffen Rendle. Factorization machines. In *Proceedings of the 10th IEEE International Conference on Data Mining*. IEEE Computer Society, 2010.

[19] Steffen Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3(3):57:1–57:22, May 2012.

[20] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.

[21] Cosma Shalizi. Lecture 10: Regression trees. In *36-350: Data Mining*, 2006.

[22] Stephanie Shih. Random forests for classification trees and categorical dependent variables: an informal quick start r guide. 2011.

[23] Gábor Takács and Domonkos Tikk. Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 83–90, New York, NY, USA, 2012. ACM.

[24] Qiang Yan (Pandas Team). Informative boosted collaborative filtering for baidu movie recommendation challenge. In *Baidu, Inc. Movie Recommendation Algorithm Contest Showcase*. Baidu, Inc., 2013.

[25] Hastie & Tibshirani. K-fold cross-validation. In *SLDM III*.