The Hong Kong Polytechnic University

Department of Computing

Individual Project Final Report

Author:
Ruixiang JIANG
19079662D

Supervisor:
Prof. Song Guo

An Assignment submitted for the PolyU(H.K.):

COMP 4434 Big Data Analytics

https://www.bilibili.com/video/BV1254y157b8

May 5, 2021

Contents

1 Introduction

1.1 Background

PolyTube is an online media platform that provides various teleplay services. In company's database, a rating record is associated with each teleplay that is already in service. However, for new teleplays, such rating information is not available. For the platform's benefit, it is of high interest to predict their ratings. On the other hand, how to recommend proper content to potentially interested users is yet another challenge. In this project, we develop a solution based on Big Data techniques.

Presentation Video: www.bilibili.com/video/BV1254y157b8, Back-up

1.2 Problem Formulation

There are two main tasks in this scenario. The first is a regression task and second is to build a recommender system. For simplicity, we denote them as task 1 and task 2 in later sections. Formally, they are:

1. Given a set of rated teleplays R, and a set of unrated teleplays U, develop a model f to predict (regress) the rating $f(U)$, such that prediction error $error(f(U), G)$, *where G is unknown ground truth* is minimized.
2. Given a set of user' rating history H, develop a recommendation system f, such that for a particular user u, the rating $r(f(u; H))$ is maximized for future data.

2 Data Processing

In this section we present the general ideas of data processing in this project. The algorithmic description could be found in Appendix 6.3. We also implement preprocessings with numpy and pandas, to make the model end-to-end trainable.

2.1 Missing Value

In the dataset (Teleplay.csv and NewTeleplay.csv), there are a lot of missing data (NA, Unknown, etc.). Our strategy is use mean value to impute those missing values. Two set of MapReduce functions are used to fill missing values, first is mean value computation, and second is imputation.

2.2 Dealing With Outliers

Another problem is that datapoints in Teleplay.csv are long-tail distributed on many feature dimensions (e.g. ”episodes” and ”members”), with many exceptionally large outliers. The existence of such outliers and long-tail distribution may affect models' performance as the difference in magnification of different features may hamper gradient descent.
To deal with such issues, we first apply feature clipping, A manually defined boundary is applied to constraint the features magnification, and hence relief some of the outlier problem. Apart from it, we apply feature scaling by a transformation to all data points on a certain feature dimension. The transformation is based on quantile and statistically robust to outliers, the result of such transformation is illustrated in figure 1.

2.3 Feature Encoding

Some of the features in this dataset are not numerical. To run model on it, we need to encode them as numerical values[1]. In this section we briefly present our encoding strategy.
For ”genre” feature, it is multi-valued feature. We convert them into vectors according to a global corpus. More specifically, we use ”multi-hot encoding”, which means a vector with many '1's and '0's, and 1 means a teleplay belongs to specific genre. The reason behind is that the genre is usually not exclusive. We do not use ordinal encoding as the distance between any pair of genre is undefined. However, this encoding technique may cause ”the curse of dimensionality”, since the encoded feature will be a sparse and high-dimension matrix.

[1]this does not include teleplay ”name” feature, which is simply discarded in our design.

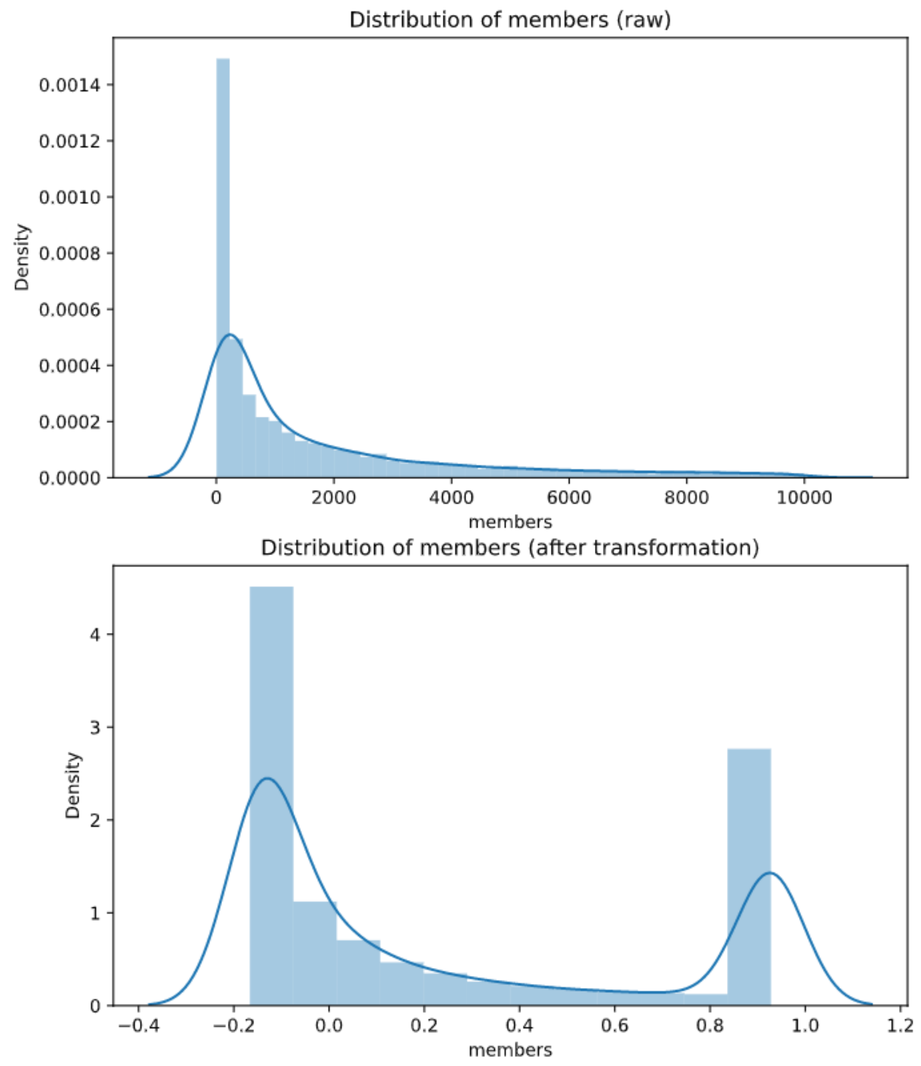

Figure 1: Up: Data distribution of "members", outliers larger than 10000 are trimmed for better illustration.
Bottom: distribution after transformation

For ”type” feature, we adopt ordinal encoding. The labels are encoded as scalars. Our intuition is that there is a loosely order relationship between types (”long” > ”medium” > ”short”)

2.4 Others Data Analytics

Besides aforementioned preprocessing, we also applied extensive exploratory data analysis (EDA) on this dataset, including correlation analysis, pair-wise feature visualization, etc. Due to limited pages, we cannot further illustrate them in this report, but the visualization details could be found in attached scripts.

3 Model

3.1 Model Architecture

3.1.1 Task 1 Model

For the task 1, we propose a end-to-end regression model, which is illustrated in fig. 2. Such model work flow has 2 major components, 1). MapReduce data preprocessing, 2). a regression model. The MapReduce-based preprocessing is already introduced in section 2.1.

The machine learning model adopts model stacking [1], a ensembling method that promote model precision and robustness by learning a hierarchical of models. Basically, the processed data will be duplicated and feed to a wide variety of base models, which outputs ”meta feature”. The meta feature is then feed as input for a higher-level model(meta model) and used to predict the ratings. Such model could be trained end-to-end.

3.1.2 Task 2 Model

In task 2 we build a recommender model. Such model can recommend contents to potential interested users. In this task, we also treat this as a regression task. Specifically, for each users, we train a personalize model to predict his/her rating on all teleplays. Such predicted ratings could be used for recommendation, by recommending top K unwatched

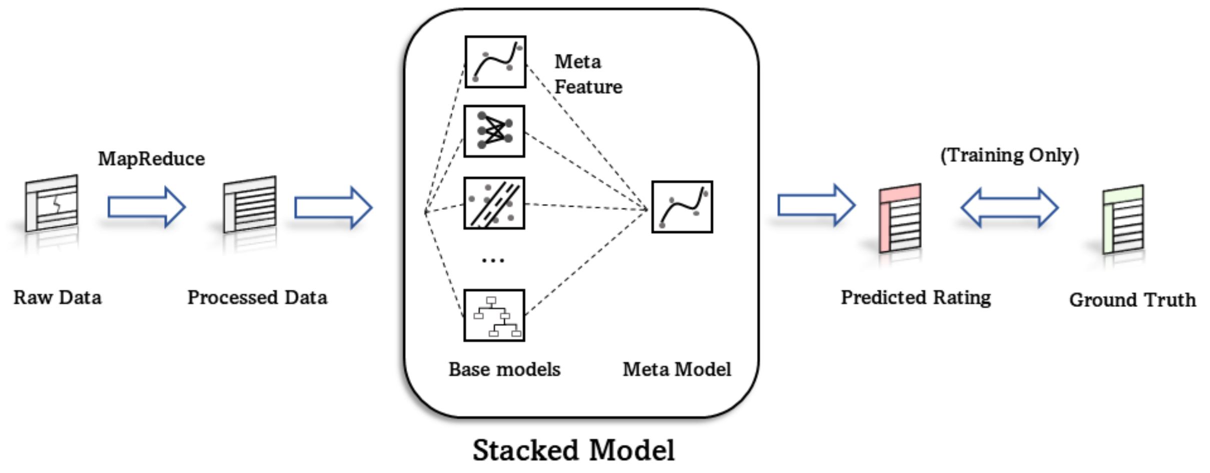

Figure 2: Machine learning model for task 1, the components in stacked model is for illustration only.

teleplay to that user.

3.2 Implementation Details

3.2.1 Task 1 Details

For task 1, We first apply recursive feature elimination(RFE) with 3-fold cross validation to find possible unexplanatory features. The results are illustrated in fig 3, As the cure are converging to maximal as feature increase and there are no dramatic decrease of performance, we decide not apply any elimination to the training data.

In the stacked model. we used a wide range of base models, which include weak learner such as Linear Regression (LR) up to strong estimators such as Gradient Boosting Decision Tress (GBDT).
in detail we used following base models: Linear Regression (LR), Lasso Regression, Bayesian Ridge Regression[2],Decision Tree (DT) Support Vector Regressor (SVR), k-nerest neighbor (kNN), Random Forest Regression (RF), Gradient Boosting(GBDT). Due to limited computational resources, we did not include MLPs as base model. Further implementation details of LR and MLP are in Appendix 6.1.
For the meta-model, we simply use linear regression. The intuition behind this is that our base models are already strong enough, a simpler hyper-model can potentially avoid overfitting.

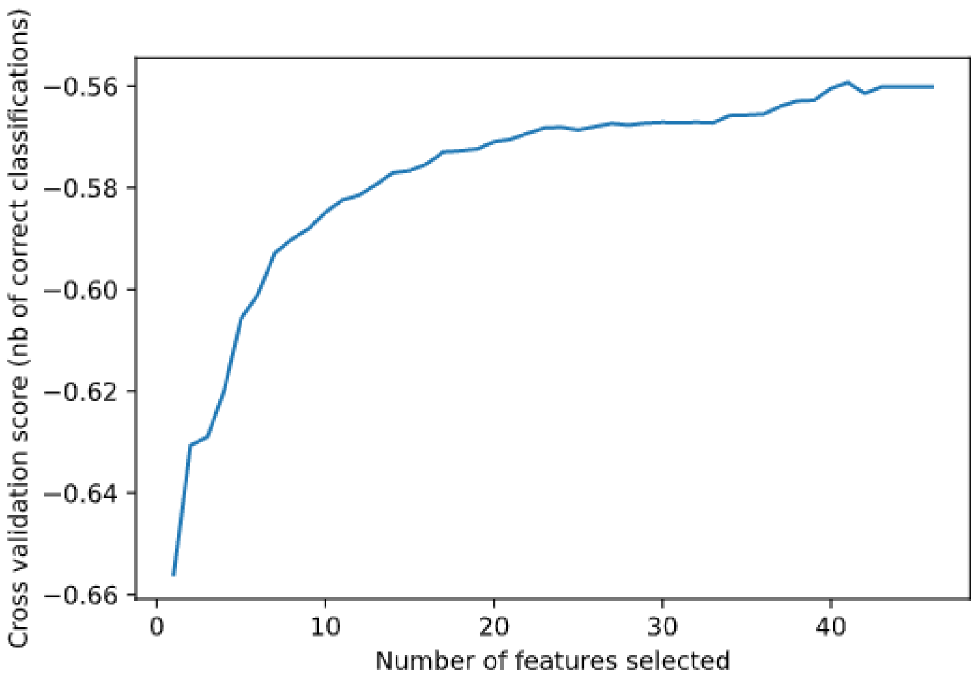

Figure 3: Recursive Feature Elimination Experiment result

3.2.2 Task 2 Details

We implement the task 2 model as content-based, such that for each user there will be a individual model, and therefore training stage could be parallelized. More specifically, for each user, we extract their rating history and aggregate as a new table (The distributed algorithm are in Appendix 6.3). Such rating history table for a specific user is then joined with the table all of teleplays (NewTeleplay.csv) on the ”teleplay_id” feature, generating a table with teleplay features and user ratings for model training. We then train a regression model for this user, where rating is the target, and other features are input.

3.3 Training

For task1, the stacked model has a wide range of base models. In order to search for best hyper-parameters, we used grid searching with cross-validation. For each of the base models, we define some set of parameters to be optimized, and the grid search take about 9-12 hours on a Intel Xeon 16 core CPU. In particular, each cross-validation round take about 10 minutes. The result of performance, however, only increased about 1%, which indicates that we are already approximately arrive at the asymptotic upper boundary of

Models	LR	PolyLR(n = 2)	Lasso	MLP	**Stacked**
5-fold CV RMSE	0.748	0.762	0.764	0.688	**0.649**

Table 1: Quantitative result of cross-validation task 1 model performance

Models	LR	PolyLR(n = 2)	Lasso	**MLP**
5-fold CV RMSE	1.273	1.611	1.266	**1.242**

Table 2: Quantitative result of cross-validation model performance for task 2

us processed dataset.

The training for task 2 is similar as task 1. Apart from linear regression, we build a fully connected network in Tensorflow for baseline experiments. The training of that network take less than 10 seconds on a Nvida RTX2080Ti.

4 Evaluation

4.1 Quantitative Results

In this section we compare our models with several baselines.

For task1, we use following baseline: plain linear regression, linear regression with polynomial feature expansion (degree = 2), Lasso regression ($\lambda = 0.01$), multilayer perceptron (architecture see 6.2). We use 5-fold cross validation root mean square error (RMSE) to evaluate model performance, and the quantitative results are summarised in table 1.

The baseline selection and evaluation for task 2 is similar as task 1, and the results are in table 2

4.2 Discussion

From above model performance table, we find that complex models seems to outperform others (Stacked, MLPs). For task 1, stacked model and MLP are significantly better than linear regression, one possible explanation to this is that the complex model has better ability to learn complex patterns due to over-parameterization. Moreover, ensembling methods (Stacked) will have better generalization ability. However, in task 2, the performance MLPs and LR are close, while MLPs are computationally-heavy. It may due to the very limited training set size, where deep learning methods seems not well fitted. Another interesting observation is that LR with n=2 polynomial expansion seems always get worse result compared with no expansion. We primarily accuse such performance gap to 2 possible factors: 1): although this task might be a non-linear problem, a power expansion could be improper to this task. 2): our encoding scheme of ”genre” has introduced a sparse matrix, and power expansion of it will produce an even sparser matrix, along with many meaningless feature. As a result, the polynomial regression result on such a high dimension matrix performs worse than plain LR.

Lastly, the magnitude of RMSE in task 2 seems greater than that of task 1. In fact this is mainly due to the limited validation sample size. As we use 5-fold cross validation, the validation set consists only 400 datapoints, which has intrinsically high variance (by central limit theorem). To validate our assumption, we compared our predicted rating for all teleplays versus all users rating and found that the RMSE for task 2 is in normal scale (≈ 0.7).

5 Conclusion

In this project, we followed a full workflow of data analytic tasks in a teleplay platform scenario. We first analyzed task objectives, which is to build a regression model and a recommender system. Next, extensive data analysis were applied, to get an overview of data formation. After that, we used MapReduce model to process the dataset. Then, we build several baselines and our own models to make predictions. Careful model optimization and selection were applied along with experiments on a remote server. Lastly, we use our models to make prediction.

6 Appendix

6.1 Linear Regression Implementation

We implement Linear Regression in Python language. For classical linear regression, we use closed-form formula by least-square method to arrive at the global minimum. Solving matrix inversion could consume a lot memory, so in intermediate steps, we use pseudo-inversion when the matrix **X** is in high dimension or singular[2].

Our linear regression implementation also support L1 and L2 regularization. In face of such non-convex regularized optimization problem, we adopt stochastic gradient descent (sgd) method with Nesterov momentum [3] to find the optimal value. Moreover, to avoid overflow and underflow in loss/gardient computation, we add a small value ϵ to avoid division by zero.

6.2 Neural Network Architecture

We implement three hidden-layer fully connected neural networks both in sklearn and tensorflow [3] as baseline model. There are 300,300,200[4] units in each hidden layer. All layers, except last output layer, are activated by ReLU nonlinearity. As this is a regression task, the output layer has only one node, with on activation function.

To regularize the model complexity, each layer are constrained by L2 penalizer, and we also adopts early stopping and adaptive learning rate scheduler to prevent overfitting. Moreover, for task 2, we also used batch normalization [4], drop out[5], and reduce learning rate on plateau to further regularize the model.

6.3 MapReduce Algorithms

```
##############mean value computation################
```

[2]In principal the matrix shall be non-singular, but our experiment show that if the input is power expanded, chances is that we get singular matrix

[3]We use sklearn in task1, and tensorflow in task2

[4]The settings are a little bit different in task2.

```
def map_1(line):
    # input lines of the dataset
    for feature_name, feature_value in enumerate(line):
        if not feature_value.is_null():
            emit(feature_name, feature_value)



def reduce_1(key, value_list):
    """each key corresponds to one feature"""
    emit(key, value_list.mean())



##############missing value imputation###############



def map_2(line):
    # input lines of the dataset
    if line.contain_null():
        i = line.find_na()
        line[i] = feature_mean[i]  # computed in mapreduce 1
    emit(line, "")  # emit whole line, value unused in this case.



def reduce_2(key, value_list):
    """each key is one line of new data records"""
    emit(key, "")

##############feature encoding###############

def map_3(line):
    # input lines of the dataset,
    genre = line.genre
    # corpus contains all genres.
    embedding = [0 for _ in range(len(corpus))]
    for i, v in enumerate(corpus):
        if v in genre:
            embedding[i] = 1
    line.join(embedding)
    emit(line, "")  # emit whole line, value unused in this case.


def reduce_3(key, value_list):
    """each key is one line of new data records"""
    emit(key, "")


##############user rating table generation################
# SELECT * from rating
# WHERE user_id = 53698

def map_4(line):
    # assume each line is a record in database
    if line.user_id == 53698:
        emit(line, null)


def reduce_4(key, value_list):
    """each key corresponds to one line of data record of a particular user"""
    # simply emit them for aggregation
    emit(key, null)
```

References

[1] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[2] David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.

[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[5] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.