Classified: TalkingData

Jordi Beernink, Thijs Werrij, Roel Bouman, Gerdriaan Mulder, Jeffrey Luppes

Radboud University

Abstract. The goal of this project was to build a prediction model based on user's demographic characteristics. With as goal that millions of advertisers could pursue marketing efforts relevant to their users and catered to their preferences. The model was build with three different methods in mind as to find the most optimal algorithm to build the model with. These were LogisticRegression, XGBoost and DeepLearning. These algorithms were used in a pipeline to analyze the features of the dataset. This resulted in varying scores, where DeepLearning resulted in the best ranking of 645. This could be because of the PreLU which adaptively learns and which can improve the accuracy.

Table of Contents

1 Introduction

This report details the work done on the TalkingData Competition data set by the Classified team. While the competition closed in September of 2016, the data set remained public and open to submissions but the leaderboard was frozen.

In this report a number of approaches that were tried are discussed along with their results. Finally, some other ideas for follow-up work are discussed as well as individual contributions by team members. All of the code referenced in this report can be found in a Github repository linked to at the end of this report.

1.1 Data set

The data set was collected from Kaggle and consisted of eight csv files totalling 1.2 GB. In order to generate a single data set these files were merged based on the device ID, which generated a data set of 4.2 GB. This data was fairly sparse, as many entries only had a few associated events, while others ranged in the thousands. This was true also for the phone brand types with some phones appearing only a few times while others had quite a massive share.

A short schema of the raw data is as follows:

- app_events: *event_ id, app_ id, is_ installed, is_ active*
- app_labels: *app_ id, label_ id*
- events: *event_ id, device_ id, timestamp, coordinates*
- gender_age: *device_ id, gender, age, group*
- phone_brand_device_model: *device_ id, phone_ brand, device_ model*

1.2 Problem Description

"Nothing is more comforting than being greeted by your favorite drink just as you walk through the door of the corner café." – TalkingData Competition Page

The goal of the competition is to predict mobile behaviour based upon app usage, phone brand and location information. As an abstraction upon this, the goal is to accurately predict in which gender and age group the owner of a mobile device resides. The predictions are accepted as twelve categories: six age groups for each gender.

1.3 Evaluation

Submissions are evaluated using the multi-class logarithmic loss algorithm. For each device the output is a row in a csv file with the device_id followed by twelve probabilities between 0 and 1 corresponding to the prediction for each class.

2 Approach

The following section discusses the approach to this problem.

2.1 Pre-Processing

The pre-processing step was the most tedious phase. After merging the .csv files together the data was still incredibly sparse: over 2/3rds of all devices had no events linked to them. With that knowledge the data files were instead combined on basis of the device properties and the installed applications.

2.2 Feature Extraction

The feature extraction was the same as the pre-processing as this was done in two different methods. The manual method was done by creating a co-occurrence

matrix of each possible combination of features and the age gender group, gender and age. These matrices were made into histograms to visualize possible preferences of features per age gender group. These features were: Brand, Model, Apps installed and the amount of events present. The automatic method consisted of

One Hot Encoding all the features that were mentioned in the manual method, excluding events participation[1]. These One Hot Encoding arrays were then converted into a sparse matrix and were then dumped into pickle files. Which could be used for the pipeline.

3 Classifiers

In this section the different algorithms that were used in this project will be globally explained and how they were implemented in this project.

3.1 Simple Classification Algorithms

The first algorithm that was used in this project were Random Forest and LogisticRegression.

Random Forest To establish a baseline for a Random Forest Classifier from the Scikit-learn package was implemented, as it was persumed that it could serve as a starting point for more complex models.

A 1000-tree model scored poorly, with the multi-class loss as reported by kaggle being 6.60. Attempts to improve upon this score were hindered by the sparse nature of the data and the RF implementation quickly became a low priority.

Logistic Regression Another baseline model was the Logistic Regression classifier. The best working solver for the large sparse dataset was 'lbfgs'.

3.2 XGBoost

One of the methods that was used in this project was XGBOOST. Which is an acronym for Extreme Gradient Boosting. Which is a variation on the machine learning algorithm: Gradient Boosting [2]. Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models. It builds the model in a stage-wise fashion like other boosting methods and it generalizes them by allowing optimization of an arbitrary differentiable loss function [2].

The difference between the two terms lies in the fact that XGBOOST uses a more regularized model formalization to control over-fitting. Which should give it a better performance [3]. The parameters of XGBOOST can be divided into three main aspects [4]:

- General: *Which guides the overall functioning*
- Booster: *Guides the individual booster(tree/regression) at each step*
- Learning *Task: Guides the optimization*

For this project the iterations, type of model (tree-based, linear), depth, weight and loss of the leaf nodes and the learning rate(known as eta) were used to get the optimal results out of the dataset [4]. Due to the lack of knowledge concerning this algorithm, it was not possible to find the optimum values for these parameters to get the optimal results.

3.3 Deep Learning/Neural networks

Another algorithm that was used in this project was Deep Learning/Neural Networks. This was done by using the library of Keras with as backend Theano and Tensorflow.The implementation of Keras models is pretty straightforward: Create a model and add layers to it via already implemented methods. The model used in this project consisted of three different layers. The first was the Dense layer, which is just a normal densely-connected neural network layer, where you have to define how many neural units you want to use. The second layer was the Dropout layer. This layer sets a given fraction of inputs to zero, to compensate for overfitting. The last layer was PReLU, or Parametric Rectified Linear Unit, which ‘adaptively learns the parameters of the rectifiers, and improves accuracy at negligible extra computational cost.’ [5]

This model was first tested by using a sample of the input data to check if the algorithm worked correctly. Afterwards the algorithm was implemented into the pipeline.

The variables that were defined for testing were the number of iterations (splits for cross-validation) and the number of epochs (separately for the initial training cycles and the final training). Furthermore, the Dropout rates and Dense units could have been changed, but because the th, but we tried to keep these constant by using commonly used values and values that returned better results.

4 Results

In this section the Kaggle results of the different methods will be shown and the rank in the leaderboard corresponding to that specific result.

4.1 Simple Classifications Algorithms

In the case of the Simple Classification algorithms the first algorithms to be tested were RandomForests and LogisticRegression. Which resulted into the following scores.

RandomForestClassifier

- Score: 3.2
- Position: 1667

LogisticRegression

- Score: 2.265
- Position: 666

Other classification algorithms were tested to see if they could match with LogisticRegression. These were: DecisionTree, KNeighbors, SVC, Bagging and ExtraTrees.These performed even worse than Random Forest and were not taken into account for the rest of the paper.

4.2 XGBoost Results

For the Kaggle submissions, both a linear model and the tree structure were used to test which resulted in a better score.

XGBoost Linear Model:

- Score: 2.26853
- Position: 776

XGBoost Tree Model:

- Score: 2.27968
- Position: 996

4.3 Deep Learning Results

In the case of Deep Learning only the commonly used values for the parameters and a low amount of iterations were used to get the following result. A higher amount of iterations would have resulted in a high training time.

Deep Learning:

- Score: 2.25992
- Position: 645

5 Discussion

One of the things that immediately was made clear with these results was that even with the default settings the scores of the three different algorithms was pretty good. Though because of time constraints, the parameters were unable to be optimized with an optimization algorithm. Otherwise the scores could have been better, which could have resulted into a lower score in the leaderboard.

6 Other ideas

A number of concepts were considered but not explored. For features the many options with the spatial coordinates were not fully used. It could have been possible to reverse geocode the coordinate pairs to trace devices to cities and use that information as a feature. Additionally, many more features could have been involved that involve distance, density or time.

7 Individual Contributions

By alphanummeric ordering:

- Jordi Beernink: *Project leader, Pre-processing, XGBoost implementation*
- Roel Bouman: *Classification Pipeline and RF implementation*
- Jeffrey Luppes: *XGBoost implementation, Report writing*
- Gerdriaan Mulder: *Pre-processing, installation on the lilo servers*
- Thijs Werrij - *Deep Learning*

7.1 Github Repository

All code accompanying this report may be retrieved from `https://github.com/RoelBouman/Classified/tree/first_competition` where it has been bundled as a release. The README on that page alsohas extensive material on how to run the code on the lilo.science.ru.nl servers where it has been prepared for a demo.

Bibliography

[1] Harris, David and Harris, Sarah. *: Digital design and computer architecture*, 2nd Edition, San Francisco, Calif, Morgan Kaufmann, p. 129, ISBN 978-0-12-394424-5, 24th July 2014

[2] Chen Tianqi *: TalkingData - Linear Model on Apps and Labels*, https://www.quora.com/What-is-the-difference-between-the-R-gbm-gradient-boosting-machine-and-xgboost-extreme-gradient-boosting, Quora, 15th September 2015.

[3] Chen Tianqi *: Digital design and computer architecture*, A Gentle Introduction to Gradient Boosting, University of Washington, 22nd October 2014

[4] Aarshay Jain. *: Digital design and computer architecture*, https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/ Complete Guide to Parameter Tuning in XGBoost (with codes in Python) 1st March 2016

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun *: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, http://arxiv.org/abs/1502.01852, CoRR, 2nd March 2015.