Talking Data Mobile User Demographics Prediction Report

Huy Doan Ravindra Manjunatha Colin Maxfield
Saharsh Oza Chin Wei Yeap

University of Texas at Austin

In this project, we analyzed Kaggle Talking Data dataset [7] and applied multiple machine learning models to predict the gender and age group based on mobile user app activity, time, and location. We built derived features based on specific demographic profiles. We experimented with many ensemble methods, such as XGBoost, AdaBoost, and auto sklearn (Auto ML), and benchmarked their relative performance. In our experiment, we found that neural network and logistic regression performed the best while boosting and ensembling improved our results significantly.

Keywords: Data Mining, Machine Learning, Demographics Prediction, Neural Network, Logistic Regression, XGBoost, AdaBoost, sklearn Auto ML

1 Introduction and Background

1.1 Motivation and Previous Research

In 2016, there were 4.61 billion mobile phone users worldwide and 2.6 billion of them were smartphone users [2]. With an increasing number of smartphone apps using location and context-aware sensors, smartphones collect large amounts of data on mobile users. This provides opportunity to develop machine learning models that can predict user demographics such as gender and age based on mobile user app activity. By building such models that predict the demographics, the consumer product marketing profile and the user experience can be personalized to capture target audience more successfully. This is based on the assumption that mobile user app activities reflect their shopping behavior, social behavior, habit, personality, and income level (purchasing power).

In previous research paper, Qin et al proposed using correlation matrix to link app category to user demographics [9]. They suggested using category imbalance to improve Bayesian pre-

diction from the pre-built correlation matrix [9]. They also applied smoothing on the standalone orphan user and category with category neighbors and user neighbors while applied Single Value Decomposition (SVD) and Pearson Correlation Coefficient (PCC) to measure similarities of users and categories with their neighboring values [9]. They also used the Term Frequency - Inverse Document Frequency (TF-IDF) method to reweight and normalize each user and category due to class imbalance [9]. In another research paper, Aarthi et al proposed modeling mobile user demographics of churners (unpaid customers) such as social and economic status from mobile social network and monthly mobile call and message activity to reduce the churn rate (customer turnover rate) [1]. Besides, Duong et al suggested to use e-commerce product catalog mobile browsing history and time to predict gender, age, and education of users using multi-classifiers, random forest algorithm, SVM, Bayesian network, cost sensitive learning, resampling, and class balancing methods [3]. From these research papers, we conclude that we need a sparse matrix of label categories that can capture similarities across users and also need to use the class imbalance problem as a feature.

In this project, we used Kaggles TalkingData dataset to predict mobile user demographics (gender and age) based on downloaded app, app activity, location history, and activity time. We were interested to see which feature is the highest demographics predictor and which features are weak demographics predictors.

1.2 Dataset

1.2.1 Data Schema

TalkingData is a middleware that runs on a users mobile device. It samples the users activities periodically. It takes snapshots of the users device periodically. Each such snapshot is referred to as an event. The event captures the time and location of the snapshot along with information on the running and installed applications. The dataset is granular at a device id level.

1. Each event has attributes:
    - $device\_id$: Unique user identity
    - $time\_stamp$: Event time
    - $location$: Latitude and longitude
    - app_id : Application identity for all the installed apps when the $event\_id$ sample is taken
    - is_active : Specifies which of the given apps are active at a time
    - $is\_installed$: This is 1 for all. So, it provides no inference

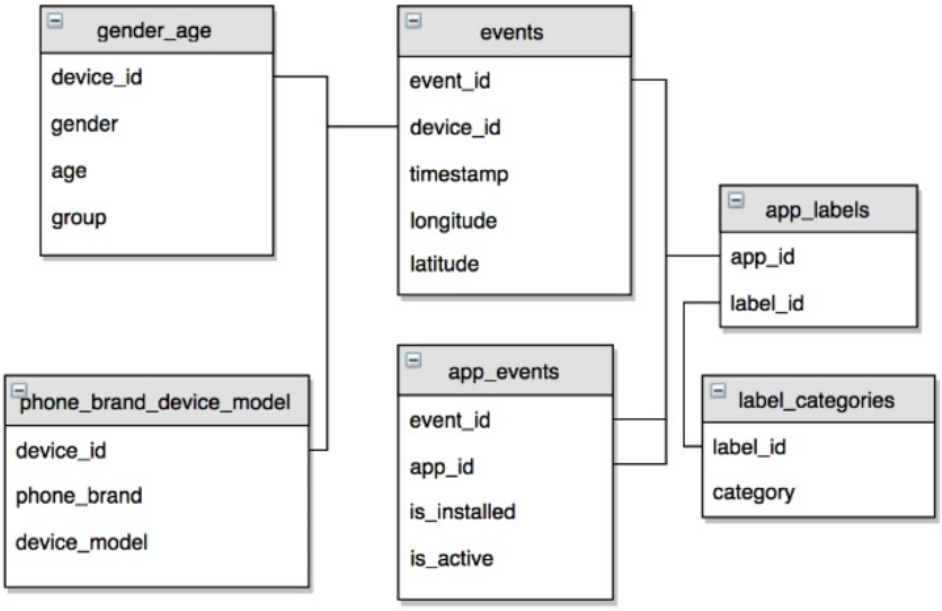


Figure 1: Entity Relationship Diagram (ERD)

2. Each app has a category associated with it. For example: finance, gaming, and health.
3. Each *device_id* has attributes:
    - Phone brand and model
    - Gender
    - Age
    - Group: this is a combination of gender and age interval. For example: F21-23 represents a female in the age group of 21-23

There were 74,645 train users and 112,071 test users. Among train users, 23,309 users had events while 51,336 users did not have events. There were 130 phone brands, 1599 phone models, 3.2 million events, 32 million app events, 19,237 total apps, and 930 app label categories. All dataset were provided in multiple *csv* files.

1.2.2 Dataset Challenges

The data provided had been collected from real users and as such, came with its own set of inherent challenges that we needed to overcome. This section will discuss how the challenges and quirks of the dataset guided the model selection process.

The dataset challenges can be broadly classified as those posed due to the nature (bug or feature!) of data aggregation processes and those that are inherent to the data distribution. The

former can be best exemplified by:

(a) Only 30% of the users had any event data. This meant a bulk of the prediction had to rely only on the phone model and brand name. This limits the predictions to two data points that might not even be related to the age and gender of the users.

(b) Multiple rows with missing or incorrect location values. Missing location values can be due to weak GPS signal, turning off location services, or not giving permission to the sdk to access location data.

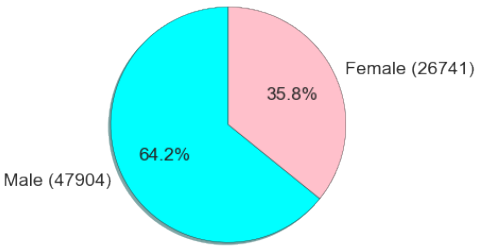
(c) App categories with non applicable values.

(d) Some device ids were duplicated.

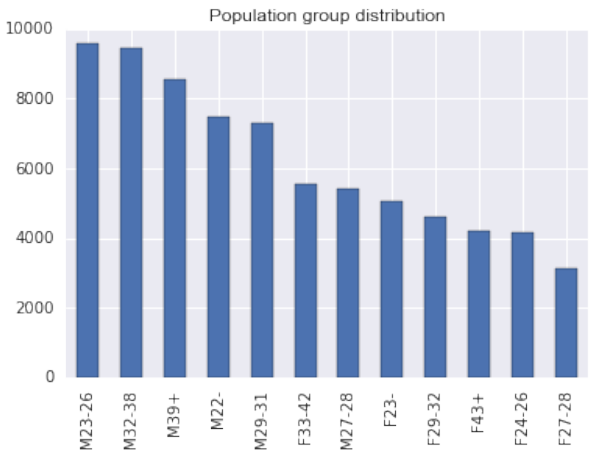
Some quirks of the dataset that guided the model choices made for prediction include:

(a) Class imbalance: Data was skewed toward men over women. Among the 12 predicted classes, the F27-28 class had about a third of the values that the most prevalent class did. Also the female classes in general had a much smaller amount of values and would result in our models struggling to predict the female groups.

(b) Skewed distribution: The data distribution of each feature was highly skewed to the left. The data was highly non-Gaussian. Hence, Bayesian methods that make a Gaussian assumption of data distribution could not be applied directly to the data.



(a) Male and female distribution



(b) Gender and group distribution

Figure 2: Data distribution

1.3 Problem Statement

This section will describe the specific prediction problem tackled in the project and the evaluation metric used. Given the user activity and device specifications in the data schema explained above, the problem statement was to predict the users gender and age bucket. Specifically, the age was divided into 6 buckets for each gender, resulting in a 12-class classification problem.

The challenge was to compute the probabilities for the 12 classes of every user. See example in Table 1. Using this list of 12 probabilities for every device, the performance metric was log

device_id	F23-	F24-26	F27-28	F29-32	F33-42	F43+	M22-	M23-26	M27-28	M29-31	M32-38	M39+
1234	0.0833	0.0833	0.0833	0.0833	0.0833	0.0833	0.0833	0.0833	0.0833	0.0833	0.0833	0.0833

Table 1: 12-class probabilities

loss as presented in the equation below.

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij}) \tag{1}$$

Here, N is the number of devices in the test set. M is the number of class labels. y_{ij} is 1 if device i belongs to class j and 0 otherwise. p_{ij} is the predicted probability that observation i belongs to class j.

As the example solution above suggests, the baseline is a random guess of 1/12 for each class. This gives a log loss of 2.48. The models presented below must perform better than this.

2 Design and Methodology

2.1 Overview

During the exploration data analysis phase, we visualized the general trends and patterns of the dataset using different graphics and statistics plots learned in data mining class, Then, we preprocessed, cleaned, and reorganized the data by handling the missing and duplicate data to prepare for model training, tuning, and testing. We put the main figures (Entity Relationship Diagram, formulas, and results) along with the explanation. We put other visualization figures (charts, plots, and diagrams) in Appendix section for reference. Finally, we summarized our results and provided justifications on relative performance results across different models.

2.2 Feature Selection and Preprocessing

For the original features, we discarded the *device_id*, *event_id*, *app_event_id*, and *app_id*, but kept the *label_id* by transforming it to one-hot coding features. We also kept the phone brand and manufacturer since they were especially useful to predict demographics for no-event users (about 70% of users). For event users (about 30% of users), *label_category* was used as additional features to predict demographics. Besides, male users were more likely to record app events compared to female users. This introduced gender imbalance problem. Also, the top two phone brands (XiaoMi and Samsung) covered the top seven phone models (MI models and Galaxy models) while the top ten app labels were mostly for productivity (Industry tag, Services), entertainment (Game, Fun), social (Relatives), and hot Chinese property housing market (Property Industry 2.0 and New). Moreover, there was a bias on high-activity app users, so we normalized some events frequency to lower granularity like low, medium, and high activity levels instead of numeric values.

The original features were limited, so more features had to be derived. We wanted to see how different demographics groups app activity location and time change during weekdays and weekends. We also wanted to compare how app labels can be joined, weighted, and derived to predict gender and age. The following aggregate statistics features were created:

- Total apps installed and active per user
- Total apps per label category
- Total events by hour, label category, and location
- Mean and variance of app activity time and location

We also generated one-hot coding for app label category and removed duplicate and missing data for consistency. We performed forward feature selection from two main features (phone brand and model) and expanded to 512 derived features. We selected the most dominant features while discarding the less dominant features. By combining all dominant predictors, we can improve our prediction probability on each gender and age group. We had 3.2 millions events and 32 millions app events, so we preprocessed the events and filtered the features. We saved preprocessed data frame in *csv* file for future reuse and later model training.

We observed that among the total 130 brands, Vivo and Samsung were strong predictors for demographics. Among the total 929 app categories, game racing, game shooting, car owners, and automotive news were male preferred apps. Mother, pregnant, child, baby, and parenting were female preferred apps. College students, game, and social were youth and student preferred apps. Wealth management and property industry were middle-aged adults preferred

apps. Some app category labels, like unknown label, custom label, and other label, were hard to draw conclusion since they were ambiguous and only related to the associated neighbor labels located close to their indexed locations.

2.3 Derived Features

We created some derived features to model different demographic profiles subgroups, such as men, women, parent, student, retiree, home owner, car owner, and other demographic profiles, based on the male count, female count, and age sum for each app label category subgroup. The formulas were shown below.

$$PredictAge = \frac{AgeSum}{MaleCount + FemaleCount} \tag{2}$$

$$PredictMale = \begin{cases} true & \text{if MaleCount} > \text{FemaleCount} \\ false & \text{otherwise} \end{cases} \tag{3}$$

We observed that app labels can be categorized into 6 main groups: male-preferred, female-preferred, gender-neutral, youth-preferred, older-preferred, and age-neutral. Broad categories like sports, entertainment, and health were gender-neutral and age-neutral. Specific categories like parent and financial were older-preferred but gender-neutral. Other specific categories like car, automotive news, racing game, and shooting game were male-preferred but age-neutral. We derived these observations based on the male count, female count, and mean age for the demographic profile built from multiple app labels.

3 Model Selection

We wanted to see how each model would perform individually, so we could learn which models showed the most promise and which ones to focus on.

- **Bayesian Methods (Model likelihood):** Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA)
- **Discriminant Methods (Model posterior):** Support Vector Machines (SVM) (Linear, Polynomial, and Radial Basis Functions (RBF)), k-Nearest Neighbors (kNN), Logistic Regression (LR)

- **Ensemble Techniques:** XGBoost (eXtreme Gradient Boosting), AdaBoost (Adaptive Boosting), Random Forest, Extra Tree Classifier, Passive Aggressive Classifier (PAC)
- **Automated Techniques:** sklearn Auto ML (Machine Learning)

Initially, we thought LDA, QDA would predict well on our dataset. After some experiments, we found that these methods gave log loss (2.50) very close to random guess (2.48). We concluded that since the data was highly non-Gaussian, the models that expect Gaussian distribution will not perform as well without proper transformation.

Tree-based models have been shown to be more successful in high-dimensional, structured, and partly discrete problems [5]. Tree based models can adapt to missing values very well. Since our dataset matched most of the above descriptions, we experimented with decision tree classifiers, such as random forest, extra tree classifier, XGBoost, and AdaBoost. Random forest is easy to interpret and extract feature (variable) importance, and controls high variance and overfitting problems of regular decision tree by averaging. We tested different parameters including maximum tree depth, maximum tree nodes, splitting criteria (Entropy and Gini), and maximum features to compare per split. The log loss improved from 2.38 to 2.34 with occasional fluctuation in intermediate values when we increased the maximum tree depths and maximum tree nodes. Our results showed Gini was better than Entropy, since Gini was used to find the largest class (class can be less than 50%) while Entropy was used to find the majority class (class of more than 50%). Random forest gave the log loss (2.36). We also tried extra tree classifier, which is random forest without bootstrapping. It gave a very similar log loss value (2.37) as random forest (2.36). In addition, we also tried ensemble methods AdaBoost and XGBoost which are robust to outliers and control overfitting by reducing variance. AdaBoost classifier didnt perform well compared to Random Forest and gave log loss (2.38) while XGBoost gave log loss (2.34).

Meanwhile, we tried SVM since the data found to be nonlinear in our EDA. We tried SVM with RBF and linear kernels and they performed slightly better (log loss 2.42) than random guess (2.48). SVM took lot of time for training with our initial hyper parameter tuning, so we decided not to pursue further as improvement in accuracy was low.

Additionally, we tried it with KNN. First, we performed PCA on the derived dataset with 512 features (Figure 11a and 11b). Then, we ran KNN on the 300 top features that captured 95% variance of the dataset. However, the result was unexpected with the log loss of more than 15. We went back and performed KNN on the 512 features and obtained fairly good score of 2.39 at k=300. Furthermore, we moved to the original dataset, which has more than 21000 features. Unfortunately, the machine we had did not have enough memory and storage for the

KNN. We stopped with KNN at this point.

After our initial experiment with multiple models, we concluded Random Forest and XGBoost were competitive learners for the final ensemble.

Nowadays, the automated hyperparameter tuning method is used to quickly filter competitive learners and shorten the time required to tune hyperparameters. Machine learning practitioners spend lot of time in tuning hyperparameter of the wrong models. We used auto-sklearn to quickly validate our initial experiment results of random forest and XGBoost as competitive learner. Auto ML (Automated Machine Learning) is an automated hyperparameter tuning framework that uses Bayesian optimisation method for optimizing the hyperparameters for machine learning models [5] [4]. It contains 15 ensemble models, 14 feature preprocessing techniques, and 4 data preprocessing techniques. The internal process of Auto ML is shown in Figure 3.

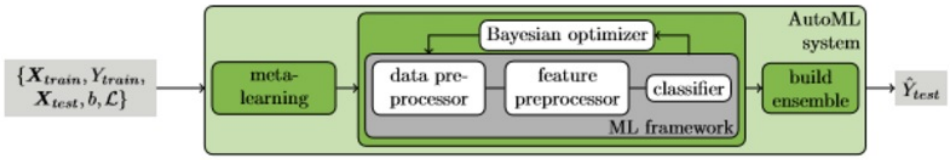


Figure 3: Auto ML system framework

Since auto-sklearn is new and open source, it has less documentation. This presented challenge for us, as budgeting the CPU time and memory is very much important to get good results. By reading through the code and issues on github, we budgeted 56 hours of CPU on 52 GB memory machine on Google Cloud and ran our code in Sequential Model-based Algorithm Configuration (SMAC) mode with 8 processes in parallel. Our initial model as shown in Figure 4 gave us a log loss (2.33). Auto sklearn suggested XGBoost, Random Forest (RF) and Passive Aggressive classifier (PAC) as competitive learners for our dataset. It also included LDA and SVM as one of competitive learners. From our previous knowledge, we decided to remove LDA and SVM from ensemble. Since PAC performed worse than RF, we decided to build an ensemble model only with XGBoost and PAC, which gave us the log loss (2.31) as shown in Figure 5. Adding RF to the above ensemble didnt improve the accuracy. Auto-sklearn helped us to identify competitive learners and optimal hyperparameters.

We examined our feature importance value of Random Forest and XGBoost models. Many of 21035 features didnt contribute much to predicting the demographics. Tree models are known to perform poorly when there are huge number of features with less information gained at each step. Hence, we decided to reduce our original feature space. We applied Chi-squared test in

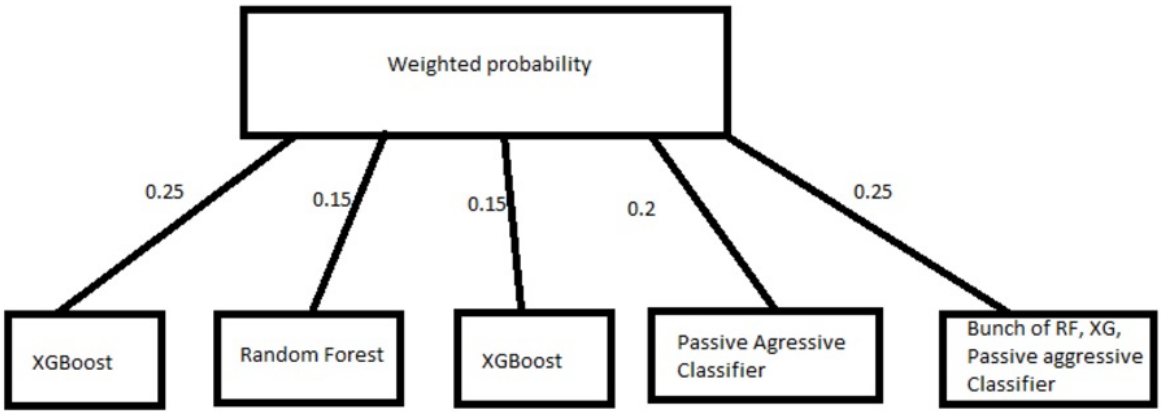


Figure 4: Initial ensemble models (XGBoost1, Random Forest, XGBoost2, PAC, RF+XG+PAC)

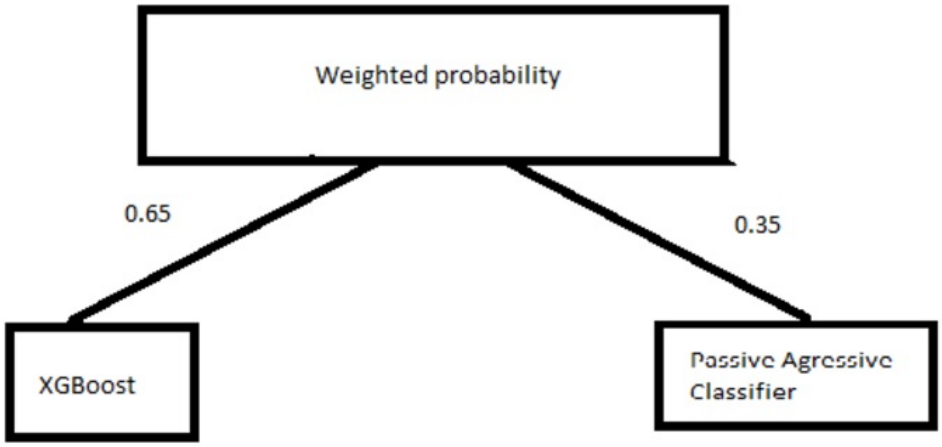


Figure 5: Final optimal ensemble models (XGBoost, PAC)

selecting the best 1000 features from our original feature space. By training XGBoost on this, the reduced feature space gave best log loss of 2.28.

For Logistic Regression (LR), our original plan was to use it to classify age only instead of trying to choose the gender-age group of each device. As we decided not to classify age and gender separately, we ended up using LR to classify the group. We modeled posterior of demographics directly by using LR, with our original feature space (21,035 features) with L2 penalty. Logistic Regression outperformed all the previous models and gave us log loss of 2.26. Since LR is monotonic, it can capture nonlinear behavior of app users activity. We tried LR with L1 penalty to reduce the feature set, and it reduced the accuracy further to log loss of 2.28. LR with the best 1000 features from Chi-squared test also gave us log loss of 2.28. We also

combined XGBoost, PAC, and LR as our next ensemble model which gave us log loss of 2.27.

It became clear to us that logistic regression was better fit for this dataset and decided to try neural network as our next model with original feature space. Our simple feed forward neural network is shown in Figure 6.

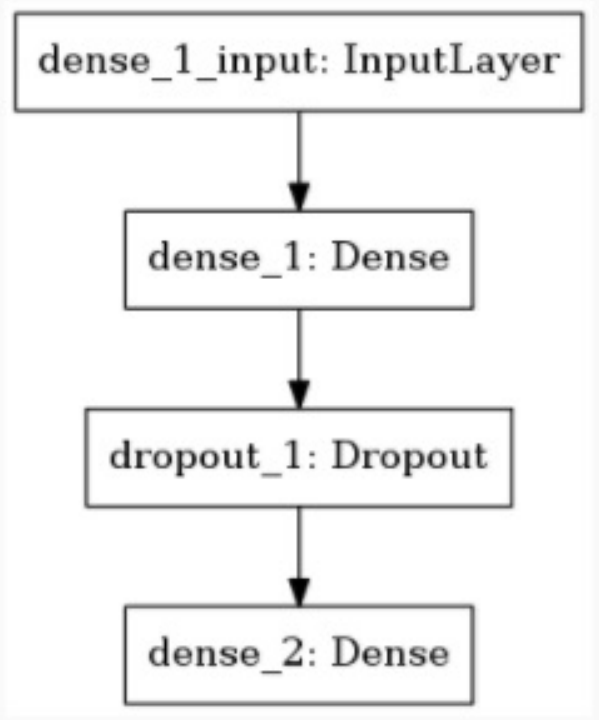


Figure 6: Deep learning neural network stages

Our first dense layer (*dense_1*) had 60 perceptrons with hyperbolic tangent function (tanh) as an activation function with all 21,035 features feeding as input. Dense layer was followed by drop out layer with 0.4 probability of inactivation. Dropout is simple way to prevent overfitting as common deep neural networks have a large number of parameters to act as very powerful machine learning systems [11]. The key idea is to randomly dropping units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. Our output dense layer (*dense_2*) had 12 perceptrons with sigmoid (S curve) activation function, followed by softmax activation function to normalize output. We used Keras to build our network. We experimented with RMSProp, Adam, and AdaDelta [11] optimizer, and found AdaDelta to be the best fit for our dataset and neural network. Network was trained up to 30 epochs with batch size matching full training set. It gave us log loss of 2.24.

4 Hyperparameters Tuning

To search for the best hyperparameter values, we experimented with Stratified CV (cross-validation), Grid Search CV, Randomized Search CV, aggressive early stopping convergence, and models stacking using sklearn Auto ML. These methods use adaptive feature and data subsampling for fast training and high accuracy across very large search size permutation of feature space, model space, and hyperparameter space (curse of dimensionality).

Stratified CV ensures each fold has at least half of representative data from each class. Our dataset had classes that were very heavily imbalanced and if we were going to have positive results it was necessary to overcome this limitation of our data. Since our performance metric was log loss, stratified CV gave better log loss than non-stratified CV for the same dataset. Grid Search CV was slow, but it can run in parallel using multiple cores of CPU. We tried faster method like Randomized Search CV, but got inconsistent results across different runs. We learned that Randomized Search CV log loss requires averaging across multiple runs since the distribution of dataset is unknown.

For cross validation (CV), we tried both 5-fold and 10-fold and both gave very similar results, so we decided to use 5-fold to save training time. We used CV mainly to tune regularization penalty to minimize log loss. Regularization penalty controls the overfitting problem (SVM). We also noticed different penalty values affect the total model training time. Early stopping convergence was used in classifiers like multi-layer perceptron (MLP) and neural network (NN) to stop training earlier when the convergence criteria was met. We did this because we were worried about overfitting our models.

For random forest, we swept the maximum tree depth from 10 to 150, and found that maximum tree depth of 50 was adequate for our dataset (Figure 13). We also swept the number of estimators (Figure 12). Most ensembled classifiers like random forest (RF), gradient boosting decision tree (GBDT) require the adequate number of estimators to improve prediction accuracy. We used 300 estimators for random forest and GBDT training. AdaBoost and XGBoost were also trained with the number of estimators from 300 to 3000. The best value was 1200 estimators for XGBoost with log loss of 2.28. Another hyperparameter is called the learning rate. For XGBoost, we tried learning rate between 0.01 and 0.05 and got the best results with learning rate of 0.05 with *warm_start* set to True. For AdaBoost, we used stubs, decision tree with maximum depth set to 1, and algorithm set to Stagewise Additive Modeling using Multi-class Exponential loss function (SAMME).

For KNN, we tuned the number of neighbors parameter k, in binary search manner. The log loss got much better with large values of k. We saw a huge improvement in the log loss when k

is about 200 compared to when k is in the range of 1-100. Then the improvement rate decreased and stayed almost stable in the range of 350-450. We decided that 350 would be the best k for this dataset.

For ensemble, we stacked multiple models with different weights to differentiate the strong and weak learners using sklearn Auto ML. Auto ML hyperparameters tuning was fully automated and had higher prediction accuracy, but it took much longer to train and test (7 hours) compared to regular models (1 hour). We learned to trade-off between accuracy, training data size, and training time on different models and aimed to get reasonable accuracy (70%) with representative data size ($>$ 50% of all training data) within reasonable training time (1 hour). We also learned to duplicate python script to run batch training and later combining the results to utilize all the CPU cores available. For sklearn APIs that allow parallelization, we set parameter *n_jobs* to value -1 to use all available CPU cores.

5 Resources

We chose Python instead of R since most of our team members are more familiar with Python than R and most advanced machine learning libraries are available in Python. We used basic libraries such as sklearn, pandas, matplotlib, and seaborn for model training and visualization, and more advanced libraries such as XGBoost, Keras, and sklearn Auto ML for better training model performance, Pandas merge was slow, so we imported all dataset to MySQL database server for faster indexing, merging, and querying. To overcome the computation limit of local laptop, we used remote cloud Google Compute Engine (GCE) that can scale up and scale out quickly to run batch and serial training (10 times faster on a 16 core CPU 4 GHz 104 GB RAM GCE instance compared to a local laptop). We utilized long overnight time to train complex models such as neural network, SVM, and kNN while using daytime to train simple models such as logistic regression and random forest.

6 Results and Evaluation

We summarized results in Table 2 and Figure 7. We found logistic regression and neural networks to be the best model for our dataset. From our exploratory data analysis, we had found that data is highly non linear. Hence, models such as LDA, QDA, and SVM proved to be poor performing models. Tree models performed well, but training with more non-dominant features resulted in better results. Logistic regression is monotonic in nature, when the classes are well

Method	Log loss
Random guess	2.48
SVM	2.42
KNN	2.39
Random forest	2.33
AdaBoost	2.38
XGBoost	2.28
XGB + PAC + LR	2.27
Logistic regression	2.26
Neural Network	2.24
Kaggle champion	2.13

Table 2: Model results

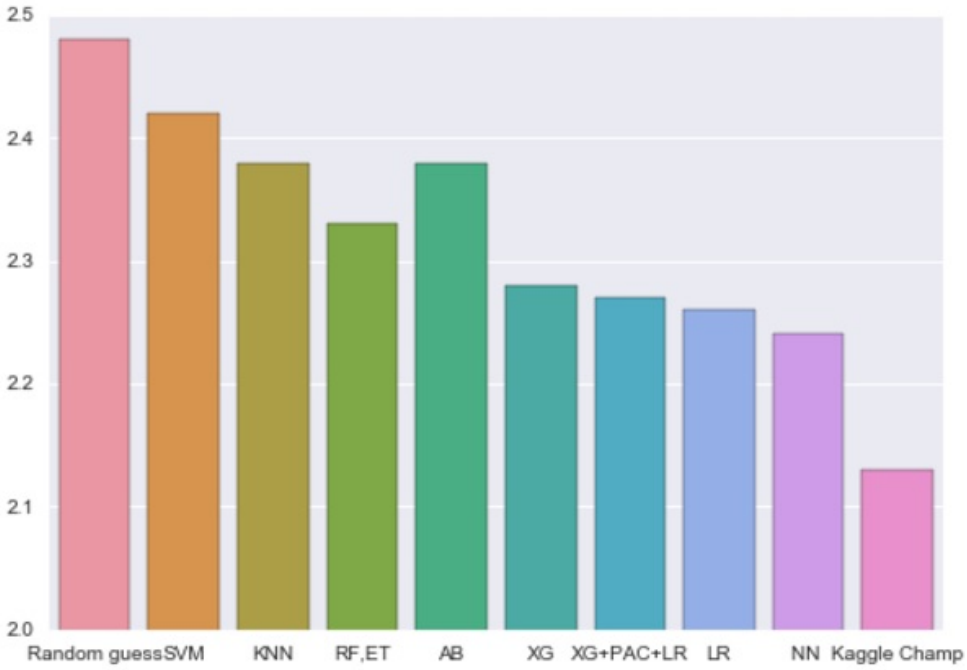


Figure 7: Model results chart

separated, SVMs tend to perform better than logistic regression. However, in more overlapping regimes, logistic regression is often preferred. Many Kaggle winning teams combined many models to improve their score. Our initial ensemble efforts always improved log loss by 0.02. We plan to ensemble our neural network, XGBoost and PAC with more hyperparameter tuning.

Figure 8 showed the confusion matrix for 12 age-gender classes. Our models clearly failed to differentiate between M22- group and M23-26 group. Same problem occurred when differentiating between F23- group and F24-26 group. Many of younger groups exhibit very similar

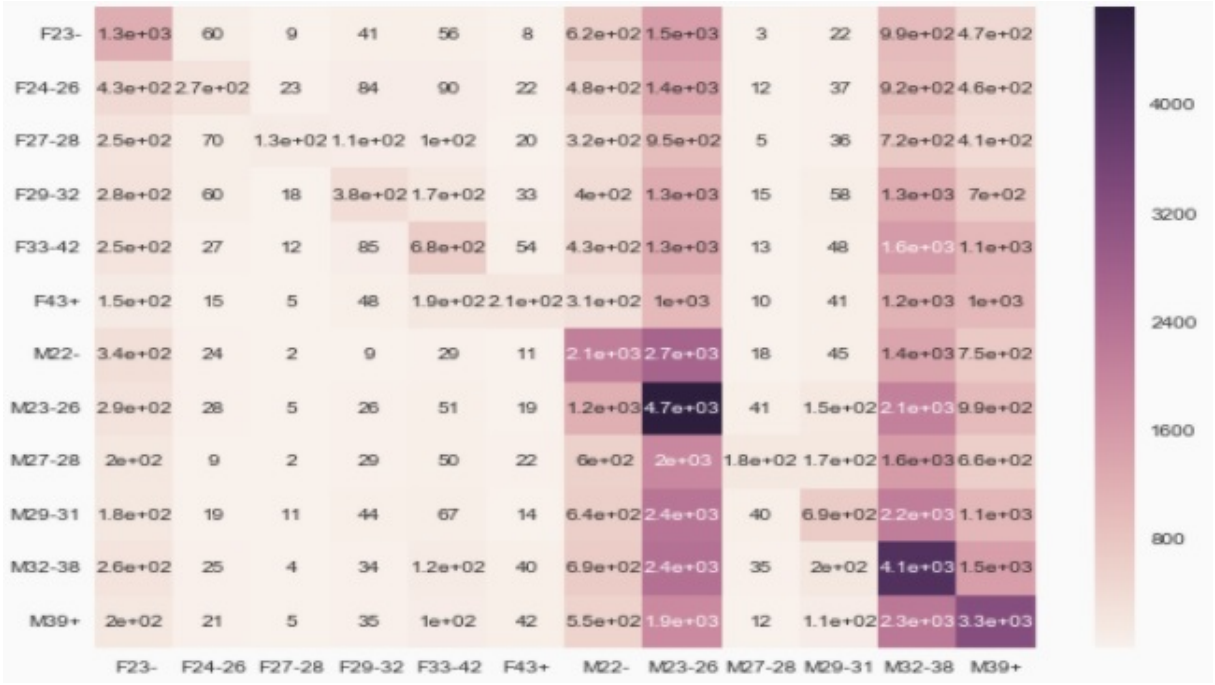


Figure 8: Confusion matrix

app activity behavior. Since app activity was collected over just 9 days, we need more derived features to improve our accuracy in those age groups.

Our team was able to get best log loss of 2.24 using neural networks, compared to winning solution of Kaggle competition champion, which achieved log loss of 2.13. Many top performing models relied on tuning the models via data leakage. Data Leakage is the creation of unexpected additional information in the training data, allowing a model or machine learning algorithm to make unrealistically good prediction [6]. Data leakage is bad, because models overfit test data and perform poorly on future data. Leakage is a pervasive challenge in applied machine learning, and causes models to over-represent their generalization error and often render them useless in the real world. It can caused by human or mechanical error, and can be intentional or unintentional in both cases. Kaufman proposed to use learn-predict separation method before releasing competition dataset to avoid leakage problem [8]. Some types of data leakage include:

- Leaking test data into the training data.
- Leaking the correct prediction or ground truth into the test data.
- Leaking of information from the future into the past.

We refrained tuning our model by data leakage and believed that our models could be im-

proved further by other ethical scientific methods as mentioned in future work section.

7 Lessons Learned

In this project, we made some mistakes and learned a lot of lessons when using existing tools and choosing models since we are still very new to data mining. We did not use cloud machines initially, because we thought we can start testing small dataset on local laptops. We quickly ran into performance bottlenecks and ended up not able to do any other tasks when local machines were training at 100% CPU utilization and sometimes our systems completely froze. In the future, we would dedicate local machines to coding and remote cloud machines for training models. With this project, we underestimated the complexity of the dataset. It was difficult for us to understand what features to derive from location and app data which could have made our models weaker. If we looked further into movement patterns it might be possible to improve the accuracy of our results. In general, it is important to realize the importance of domain knowledge when approaching a problem such as this. Anyone can take the time to create a model for a set of data but if you take the time to understand the domain you are trying to predict then it is possible to produce much better results.

The poor results from some of our models with respect to the amount of time and effort to develop them was a disappointment. Specifically, we spent a fair amount of time with LDA and QDA and found that they performed worse than a random guess. We also spent a lot of time trying to create SVM models using a couple of different kernels. Not only did we find that we were taking anywhere from 10 - 24 hours to create one model, but that they did not perform very well either. With SVM being so heavily used in modern data mining we expected to see it perform much better than it did and we were disappointed with its results. These experiences, however, only further solidify the idea that the newest trends are not always the best and it is important to analyze the problem to choose a model that works for your specific data. Logistic regression has been around for a very long time and it was able to outperform most of the newer data mining techniques with ease.

8 Future Work

We derived many aggregate features from app usage and label categories. There are many ways we can improve further on the accuracy of our models:

- Many Kaggle participants in the forum described that adding Term Frequency - Inverse

Document Frequency (TF-IDF) features on app label categories, phone and brand improved predictions.

- We also believed adding interactive features between app usage and range of apps being used would improve prediction.
- We could improve our neural network model in many ways, such as adding more layers to our neural network and adding batch normalization layer between layers to improve learning and dropout to input layer.
- We plan to build multi-stage model rather than directly predicting gender-age group. Our first stage model would predict the gender of user as male/female. This obtained gender probability can be used as separate feature for our next model to predict age group.
- After experiments mentioned above, we could ensemble many competitive learners such as XGBoost, Passive Aggressive Classifier (PAC), Logistic Regression, and Neural Network to give weighted probability. Our earlier experiment showed this ensemble of many strong learning models minimized the log loss.
- We will use frameworks like Hadoop and Spark in the future to solve the dataset scalability problem.

9 Conclusion

We showed that our models performed better with advanced techniques such as stacking, bagging, and boosting. We saved a lot of computation time by filtering out unnecessary features, underperforming models, insignificant hyperparameters, and terminating early based on convergence criteria during hyperparameter tuning. We learned many tools, preprocessing methods, feature engineering methods, ensemble methods, and valuable lessons in this project. We also learned that parallelization and distributed systems were very important for data mining when data size scaled up quickly. We also spent a lot of time on data preprocessing and feature selection. Sparse matrix can be used to expand feature set and improve accuracy, but it can incur higher training computation cost and time. Automatic machine learning could be used as filter for determining the competitive learners for ensemble and it does save lot of time for data science practitioner, but it should be used with caution. Success factor of winning Kaggle competition mostly attributes to creative features selection and model stacking, which is an art that requires intuition and experimentation. We plan to improve these skills in the future.

References

[1] S. Aarthi, S. Bharanidharan, M. Saravanan & V. Anand (2011): *Predicting Customer Demographics in a Mobile Social Network*. In: *2011 International Conference on Advances in Social Networks Analysis and Mining*, pp. 553–554, doi:10.1109/ASONAM.2011.13.

[2] Devicee Atlas: *16 mobile market statistics you should know in 2016*. Available at `https://deviceatlas.com/blog/16-mobile-market-statistics-you-should-know-2016`.

[3] Duc Duong, Hanh Tan & Son Pham (2016): *Customer gender prediction based on E-commerce data*. In: *2016 Eighth International Conference on Knowledge and Systems Engineering (KSE)*, pp. 91–95, doi:10.1109/KSE.2016.7758035.

[4] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos & K. Leyton-Brown (2013): *Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters*. In: *NIPS workshop on Bayesian Optimization in Theory and Practice*. Available at `http://www.automl.org/papers/13-BayesOpt_EmpiricalFoundation.pdf`.

[5] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum & Frank Hutter (2015): *Efficient and Robust Automated Machine Learning*. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, MIT Press, Cambridge, MA, USA, pp. 2755–2763. Available at `http://dl.acm.org/citation.cfm?id=2969442.2969547`.

[6] Kaggle: *Data Leakage*. Available at `https://www.kaggle.com/wiki/Leakage`.

[7] Kaggle: *Talking Data Mobile User Demographics*. Available at `https://www.kaggle.com/c/talkingdata-mobile-user-demographics`.

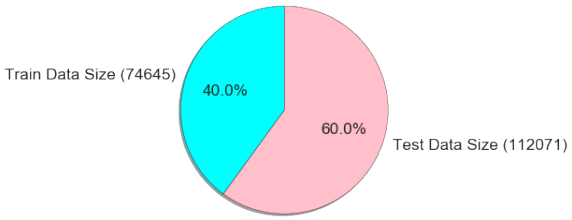
[8] Shachar Kaufman, Saharon Rosset & Claudia Perlich (2011): *Leakage in Data Mining: Formulation, Detection, and Avoidance*. In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, ACM, New York, NY, USA, pp. 556–563, doi:10.1145/2020408.2020496. Available at `http://doi.acm.org/10.1145/2020408.2020496`.

[9] Z. Qin, Y. Wang, H. Cheng, Y. Zhou, Z. Sheng & V. Leung (2016): *Demographic Information Prediction: A Portrait of Smartphone Application Users*. *IEEE Transactions on Emerging Topics in Computing* PP(99), pp. 1–1, doi:10.1109/TETC.2016.2570603.

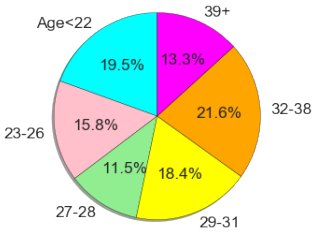
[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever & Ruslan Salakhutdinov (2014): *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. *J. Mach. Learn. Res.* 15(1), pp. 1929–1958. Available at `http://dl.acm.org/citation.cfm?id=2627435.2670313`.

[11] Matthew D. Zeiler (2012): *ADADELTA: An Adaptive Learning Rate Method*. *CoRR* abs/1212.5701. Available at `http://arxiv.org/abs/1212.5701`.

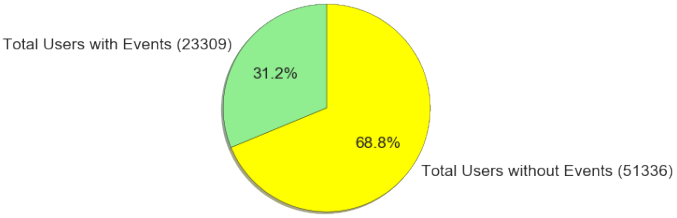
A Appendix



(a) Train and test data size distribution



(b) Age distribution



(c) Total users with events and without events distribution

Figure 9: Data distribution

Top 10 Phone Brand

Brand	Count
小米(XiaoMi)	43210
三星(Samsung)	34286
华为(Huawei)	32564
vivo	14395
OPPO	14289
魅族(Meizu)	11853
酷派(Coolpad)	8407
联想(Lenovo)	6761
金立(Gionee)	2768
HTC	2682

(a) Top phone brands

Top 10 Phone Model

Model	Count
红米(Red Mi)note	7358
MI 3	5712
MI 2S	5308
Galaxy Note 3	5019
MI 4	4798
Galaxy S4	4059
Galaxy Note 2	3993
荣耀(Honor)6	3076
荣耀畅玩(Honor)4X	2754
荣耀(Honor)3C	2598

(b) Top phone models

Top 10 App Label Category

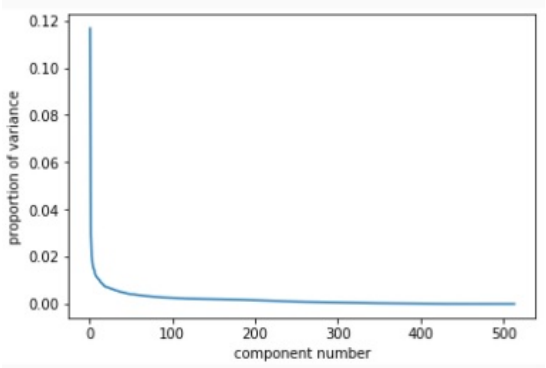
ID	Category	Count
548	Industry tag	56902
405	Custom label	53936
794	Tencent	49320
795	game	48707
704	Property Industry 2.0	45697
714	1 free	19083
713	Services 1	11840
854	Property Industry new	9955
710	Relatives 1	9027
711	Irritation / Fun 1	8831

(c) Top app labels

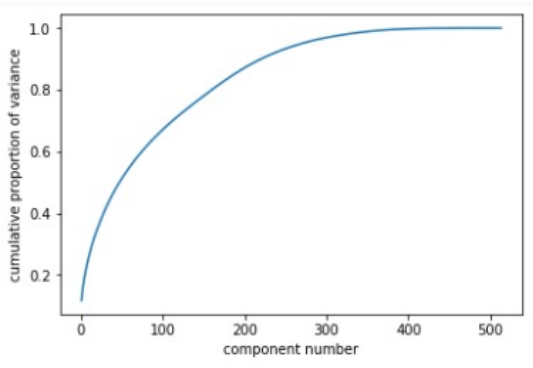
	label_id	category	M	F	AgeSum	PredictAge	PredictMale
16	17	game-Racing	60	16	2467	32.460526	True
35	36	game-shooting	102	27	4235	32.829457	True
203	222	Car Owners	661	168	28646	34.554885	True
260	279	Automotive News	367	69	14439	33.116972	True
251	270	Mother	1	4	156	31.200000	False
252	271	Prepare pregnant pregnancy	117	147	8145	30.852273	False
253	272	Parenting	35	46	2863	35.345679	False
254	273	Maternal and child supplies	49	98	4697	31.952381	False
759	842	Families with babies	192	214	13381	32.958128	False
760	843	Pregnant baby	202	223	13978	32.889412	False
761	844	Parenting stage	153	186	11125	32.817109	False
276	317	College Students	76	44	3123	26.025000	True
247	266	Housing Advice	438	152	19949	33.811864	True
233	252	Wealth Management	2804	1364	143795	34.499760	True

(d) Derived features

Figure 10: Data details



(a) Scree plot



(b) Cumulative variance plot

Figure 11: PCA

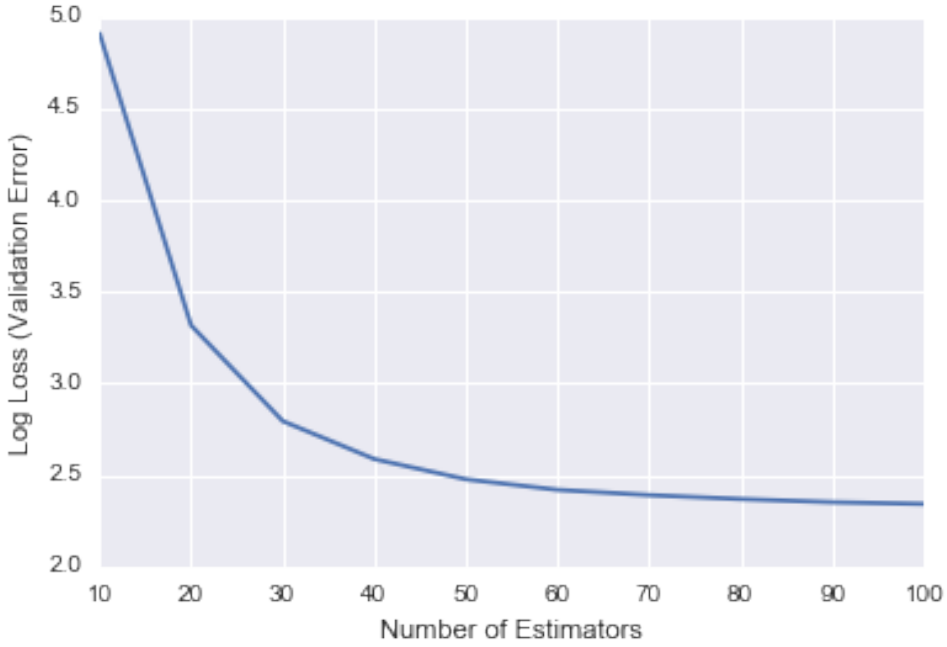


Figure 12: Random forest fixed max tree depth of 100

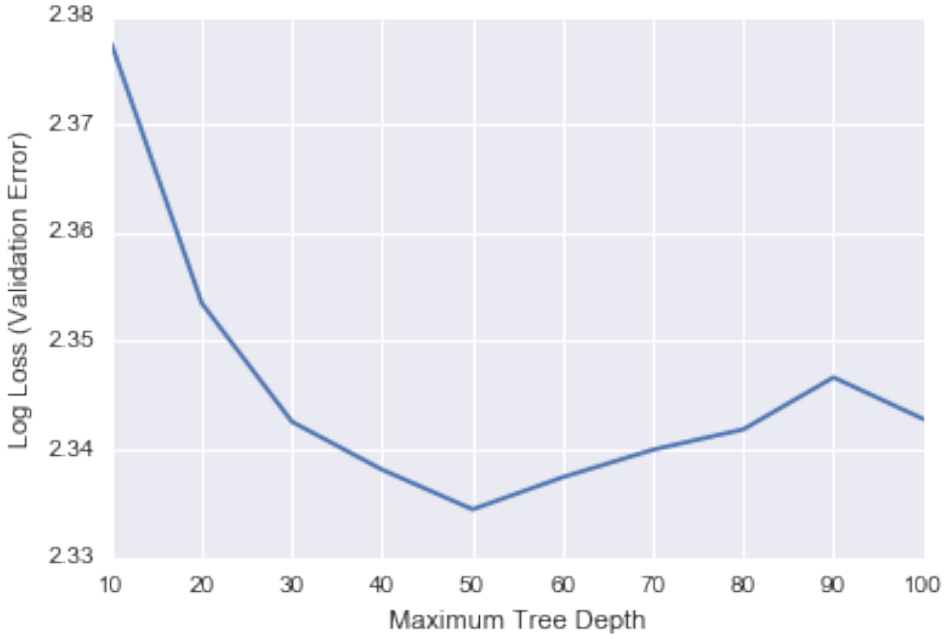


Figure 13: Random forest fixed number of estimators of 100