UNSW SYDNEY

COMP9417 - Machine learning & Data Mining

Project:
Trachack

Prepared by team:

Final fantasy

***Supervisor*: Omar Ghattas**

Team Members:

- Mingxuan Chen z5301850
- Guanzhu Hou z5325417
- Lucas Warburton z5312100
- Shangyuan Wu z5352630
- Wangqing yang z5325987

The date of project submitted: 20/4/2022

Table of Contents

Introduction

Trachack is a project that is based on Tracfone which was the first company in the United States to offer mobile phones and plans without requiring considerable credit, making cell phones more accessible to a broader segment of the population. The USA Emergency Internet Benefit Program helps low-income households stay connected during the COVID-19 pandemic by providing funding for broadband services and certain equipment. For qualifying users, the Emergency Broadband Benefit (EBB) Program will provide a monthly discount of up to $50 on broadband services. Consumers who live on qualified tribal grounds can receive up to $75 per month in additional subsidies for broadband services. Many Tracfone users enrolled in the EBB program, which provides savings and improved customer service. Many more, on the other hand, are either unaware or uninformed that they may be eligible for and benefit from the EBB program. Our task is to use machine learning knowledge to predict customers who can qualify for the EBB program.

Exploratory Data Analysis

We find that there are plenty of features to look at but many are insignificant. Taking a look at ebb1.csv for instance, there are around 40 features after we clean the data and for 'opt_out_loyalty_email' and 'opt_out_loyalty_sms', there were only 79 non-null and 84 non-null of them. The total dataset has around 50000 data, thus making this irrelevant for the purpose of our study. Columns with more than 70% missing values are deleted because they have little impact on the final prediction.

Generally speaking, most models will only accept float or int data, such is also the case for our model. In the original dataset, some features are stored in DateTime or string format, we must therefore first convert them into int or float formats before they can be introduced into our model. There are several ways to convert data and different data types should have different ways to convert.

For the 'operating system' feature, there are many operating systems that are seldomly used by customers. The majority will use 'android', and 'ios'. Therefore, for this feature, if the value

contains ‘android’ or ‘ios’, we convert it to ‘ANDROID’ or ‘IOS’ respectively. Otherwise, if it contains a value ‘not known’, we convert it to ‘Not Known’. Finally, if the value does not contain the three values above, we convert it to ‘Other’. We do this because the miscellaneous values will not contribute a lot to the final result, and this operation will reduce the different values of the ‘operating system’ feature. This will enable the use of easier encoding methods, such as one-hot, or directly used in the catboost model.

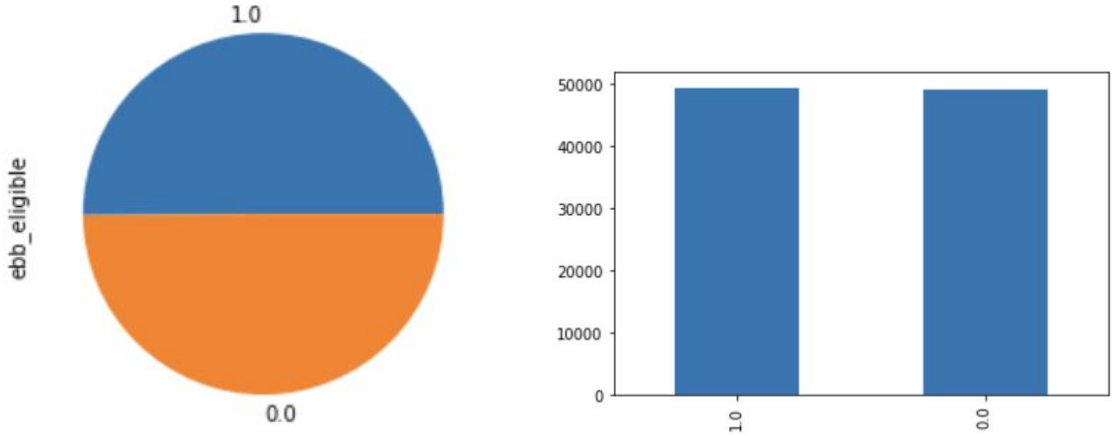
The next thing is the ‘DateTime’ feature. This data type must first be converted into integer formats before it can be used in the models. Therefore, we try to convert it to an integer. The most intuitive method is to calculate the difference between the value and a specific day. We have several choices for that specific date. For Unix time, that day is 1970-1-1 00:00:00 GMT. We can also use the difference between the average day of all the training data. Our choice is used today to subtract the values from the feature, which will result in a positive value.

In order to deal with some duplication or potentially missing data in integer or float types, we first fill the missing data with a value, usually 0. Then we have three ways to convert them into non-duplicated ones. The first way is to use value counts to count the times that each customer occurs. The second way is to use the sum command to calculate the sum of the features of each customer. The third way is to use the mean command to calculate the average of the features of each customer.

Some datasets contain duplicate values, such as the ‘activations’ dataset, ‘deactivations’ dataset, ‘reactivations’ dataset, and ‘interactions’ dataset. For such datasets, we use the value_counts function from the pandas' library to count the times that each customer occurs in that dataset. This will result in an integer value for each customer without duplication. As a result, it is easy to left join the main dataset.

For the dataset ‘ivr_calls’, we first count values as previously mentioned. Then, we calculate the mean values of the ‘isCompleted’ feature. Finally, if the value is bigger than 0, we assign the value 1, if the value is less than 0, we assign the value -1. Otherwise, 0 will be assigned. For the dataset ‘network’, we added the values of features ‘total_kb’, ‘voice_minutes’, and ‘total_sms’ up to avoid duplicates while providing distinguished value for different customers. For the dataset ‘phone_data’, the mean of the values of ‘battery_available’, ‘data_roaming’, ‘memory_available’, and ‘temperature’ can represent the features better than simply adding them up. Therefore, we use their mean value.

There are two classes in the dataset. Class 1 is 1 in the original dataset and 0 is NaN in the original dataset. Therefore, we fill NaN by zero and we find that there are around 50000 Class 1 and around 50000 Class 0. The number of them are almost the same which means the class distribution is quite average. Here is the pie chart and the bar chart of them.



Here is another way, for the state field, it is replaced by the state_income downloaded from the government so that the type can be changed from str to int. For operatering system, we only consider if it is ios system or not, if it is ios system, we marked them with 1, else we marked them with 0. For manufacturer part, let manufacturers = {'samsung' : 263, 'apple': 825, 'motorola': 200, 'lg': 200}. For numeric data, if the fillna shows (0), then fill with 0, we use binary classification to classify 'opt_out_mobiles_ads' and if -1 return true, else return false. Use hash(x) for string processing to obtain the hash value of the string, also Perform binary classification of 0, 1 (True False) for strings with only true or false or yes or no. We used mean of 'memory_available', 'revenues' and other fields which is unified with the previous data processing. Lastly, for duplicate value processing, we select the most frequent value among all the values of each consumer_id.

Methodology

For the initial composition of D4, involving main csv files, we stripped most of the features with very sparse values, such as the opt_out values and a few other meaningless features like date values. Categorical values like manufacturer and state were mapped to relative device price and real personal income respectively (this had a very small positive impact relative to just hashing the values). For OS since it was primarily android and IOS devices we just turned this into a

binary feature of 1 for IOS and 0 for everything else, since this represents the class divide in operating system usership and was thought to correlate to eligibility.

A variety of functions were devised that modified and appended features from the supplementary data to the aforementioned D4 set. For categorical data this typically involved taking the most common category value for a customer, hashing that and appending to the primary set. For numerical values we typically took the mean or sum of a customer's given values, for instance it was found with network values like ‘voice_minutes’, a customers average number of minutes was marginally better for prediction than their total minutes.

To test these preprocessing functions, we used sklearn’s metric module’s different score functions (accuracy_score, precision_score, recall_score, f1_score) as a manual calculation of specificity using sklearn’s confusion matrix function in order to ascertain whether our development was shifting in the right direction.

We felt accuracy, the proportion of true results among the total number of results, was an important metric since the distribution of customer eligibility is largely balanced. Furthermore, precision was deemed an important metric as we felt that TracHack would be embarrassed to inform customers of eligibility only to have the government reject their applications. Recall was important to us since we want as many EBB eligible people as possible to be notified that they can apply so they can enjoy the government benefit. And lastly f1 score was our guiding light in evaluating a models performance, as it balances the previous two features and is the best evaluation metric in isolation.

Results from these modified data sets were then compared to the performance of a default sklearn decision tree fitted to the primary data set. Below are the performance differences using our chosen metrics:

Function	accuracy	precision	recall	f1	specificity
add_suspensions	0.01	0.00	0.00	0	-0.00

add_deactivations	0	0.01	0	0	0
add_ivr	-0.00	0.00	-0.00	-0.00	0.00
add_throttling	0.00	0.00	0.01	0.00	0.00
add_auto_refill_de_date	0.00	0.01	-0.00	0.00	0.01
add_notifying	0.00	0.00	0.01	0.00	-0.00
add_network_mean	0.03	0.03	0.04	0.03	0.17
add_phone_data	0.01	0.01	0.01	0.01	0.01
add_redemptions	0.13	0.13	0.14	0.14	0.11
add_support	0.16	0.16	0.17	0.16	0.15
add_lease_history	0	0	0	0	0
add_react	0.02	0.02	0.02	0.02	0.01
add_loyalty	0.03	0.03	0.04	0.04	0.02

Functions that yielded >= 1% increase in performance in two or more metrics were used in the final preprocessing process.

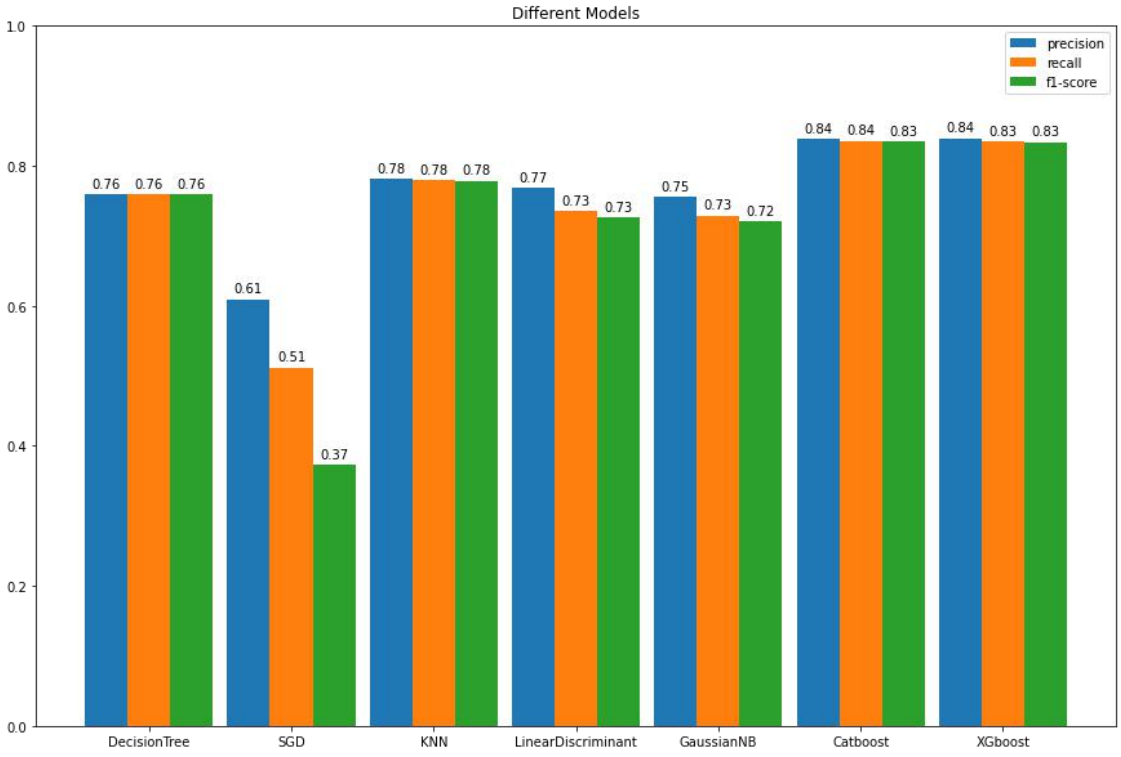
We went through the sklearn documentation and tested this dataset on all the relevant ensemble and linear models and eventually it was found that Sklearns gradient boosting model produced the best F1 scores. It's histogram variant based on Microsoft's LightGBM was eventually chosen for the final model after having been found to be 15 times faster at fitting than the standard model.

Attempts to adjust hyper-parameters during initial testing with the decision tree model were highly fruitful, with different tree-sizes yielding radically different results. However, for the final histogram GB model there was no yield to changing from the default values as can be seen below.

Hyper-parameter	Change in metrics when increasing relative to default value	Change in metrics when decreasing relative to default value
learning_rate	All metrics lower	All metrics lower
max_iter	Marginal difference in metrics	All metrics lower
max_leaf_nodes	Marginal difference in metrics	All metrics lower
max_depth	All metrics lower	n/a
min_samples_leaf	Marginal difference in metrics	All metrics lower
warm_start	All metrics lower	n/a
early_stopping	n/a	All metrics lower

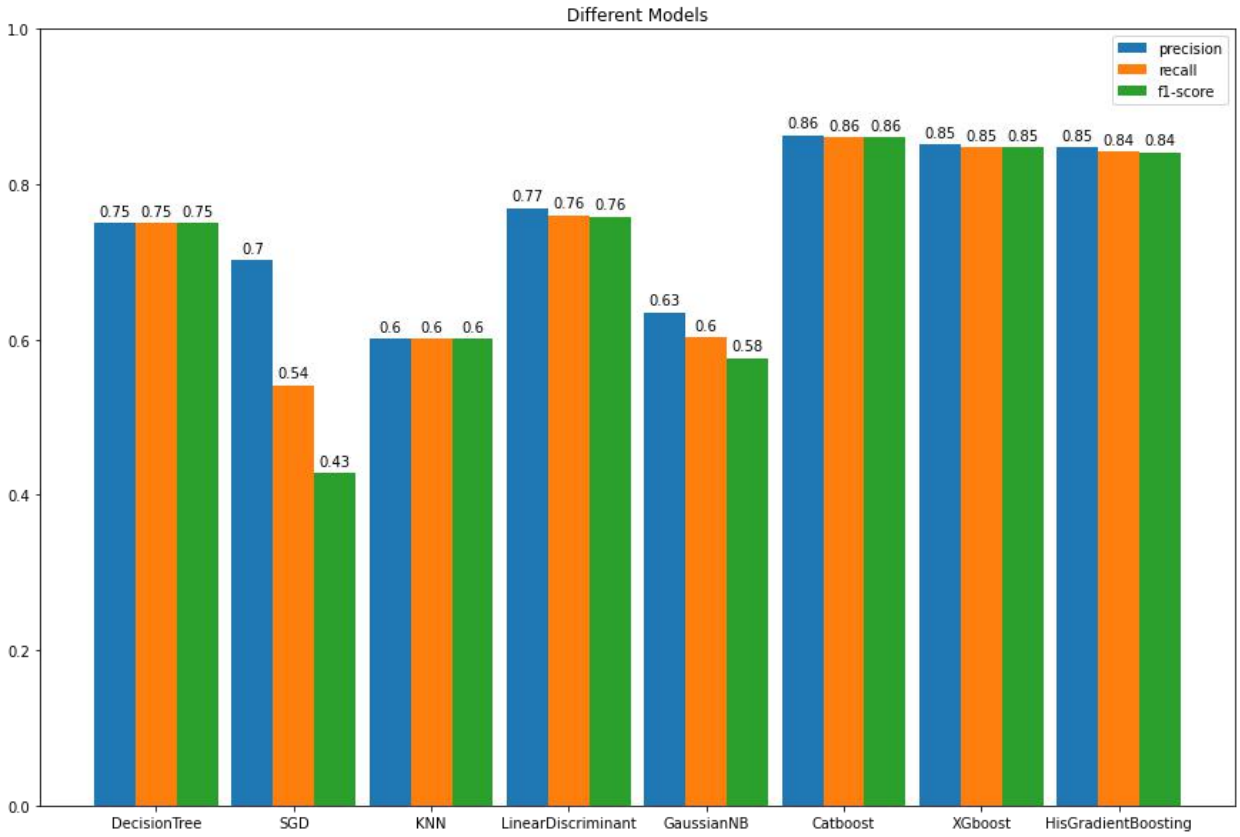
Results

D1: Original dataset



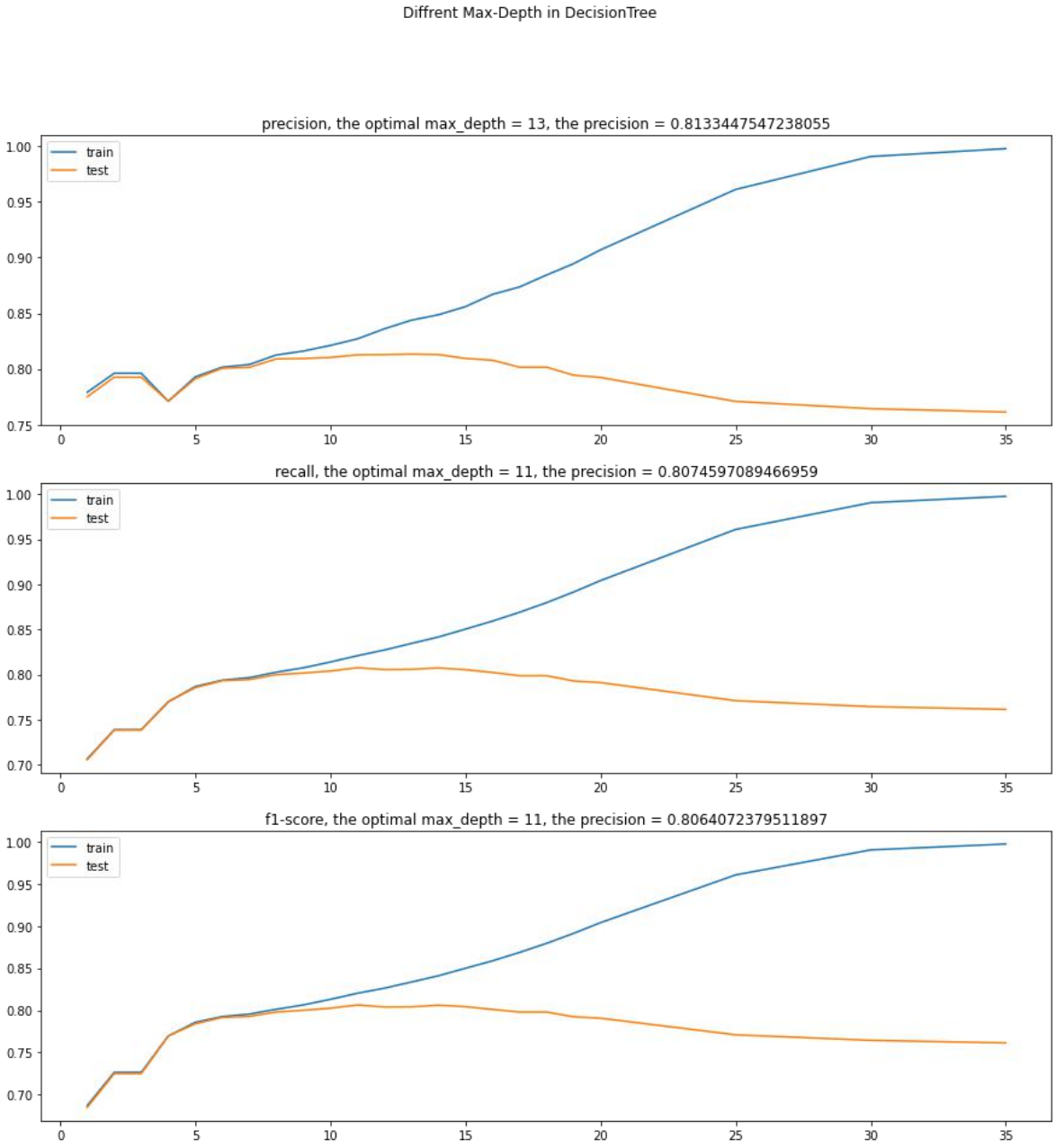
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 98389 entries, 0 to 49220
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   customer_id            98389 non-null  object
 1   last_redemption_date   98389 non-null  int64
 2   first_activation_date  98389 non-null  int64
 3   total_redemptions      98389 non-null  int64
 4   tenure                 98389 non-null  int64
 5   number_upgrades        98389 non-null  int64
 6   year                   98389 non-null  int64
 7   manufacturer           98389 non-null  int64
 8   operating_system       98389 non-null  int64
 9   language_preference    9394 non-null   object
 10  opt_out_email          5619 non-null   float64
 11  opt_out_loyalty_email  159 non-null    float64
 12  opt_out_loyalty_sms    173 non-null    float64
 13  opt_out_mobiles_ads    49077 non-null  float64
 14  opt_out_phone          8573 non-null   float64
 15  state                  98389 non-null  int64
 16  total_revenues_bucket  98389 non-null  int64
 17  marketing_comms_1      3827 non-null   float64
 18  marketing_comms_2      3817 non-null   float64
 19  ebb_eligible           98389 non-null  int64
 20  date1_diff             98389 non-null  int64
dtypes: float64(7), int64(12), object(2)
memory usage: 16.5+ MB
```

D2: GH final dataset



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 98389 entries, 0 to 49220
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   total_redemptions      98389 non-null  int64
 1   tenure                 98389 non-null  int64
 2   number_upgrades        98389 non-null  int64
 3   year                   98389 non-null  int64
 4   total_revenues_bucket  98389 non-null  int64
 5   state                  98389 non-null  object
 6   last_redemption        98389 non-null  int64
 7   first_activation       98389 non-null  int64
 8   redemption_duaration   98389 non-null  int64
 9   operating_system       98389 non-null  object
 10  opt_out_mobiles_ads    98389 non-null  int64
 11  deactivation_counts    98389 non-null  int64
 12  interection_counts     98389 non-null  int64
 13  ival_counts            98389 non-null  int64
 14  iscompleted            98389 non-null  int64
 15  total_kb               98389 non-null  int64
 16  voice_minutes          98389 non-null  int64
 17  total_sms              98389 non-null  int64
 18  reactivitaion_counts   98389 non-null  int64
 19  redenption_counts      98389 non-null  int64
 20  date                   98389 non-null  int64
dtypes: int64(19), object(2)
memory usage: 16.5+ MB
```

We have tested the 8 models on different dataset, one is the D1, the original one(ebb_set1+ebb_set2), the other is D2, the original one combined with some more sheets. As the above two diagrams show we can notice that for the first five basic models, only decision tree and Linear Discriminant have a better performance, so we focus on these two models. Since the training speed of the decision tree is much faster than other models, we test with the decision tree at first and we find that the max_depth parameter has a great impact on the performance of the model in the training set. Based on this result, we decided to tune the max_depth parameter on the original dataset. By tuning the max_depth parameter we find the results for the train and test training sets fluctuate greatly when the result of max_depth is from 1-20. Therefore, we decided to select 1-20, 25, 30, 35 as the candidate parameters of max_depth and we got the following diagram.



From the above diagrams, we can see that it is under fitted when the max_depth is below 5 and it is overfitted when the max_depth is over 10(the precision, recall, and F1 score of the train increased, but the precision, recall and F1 score if the test is decreased). At the end, we found that when the max_depth is at 11, the performance is the best. However, the accuracy of the base model still does not change significantly as the training set increases.

Similar to Model Decision tree, we use Grid search on depth and learning rate on Model Catboost to find the best parameters in the classification. The range of depth is [4, 6, 10] and the

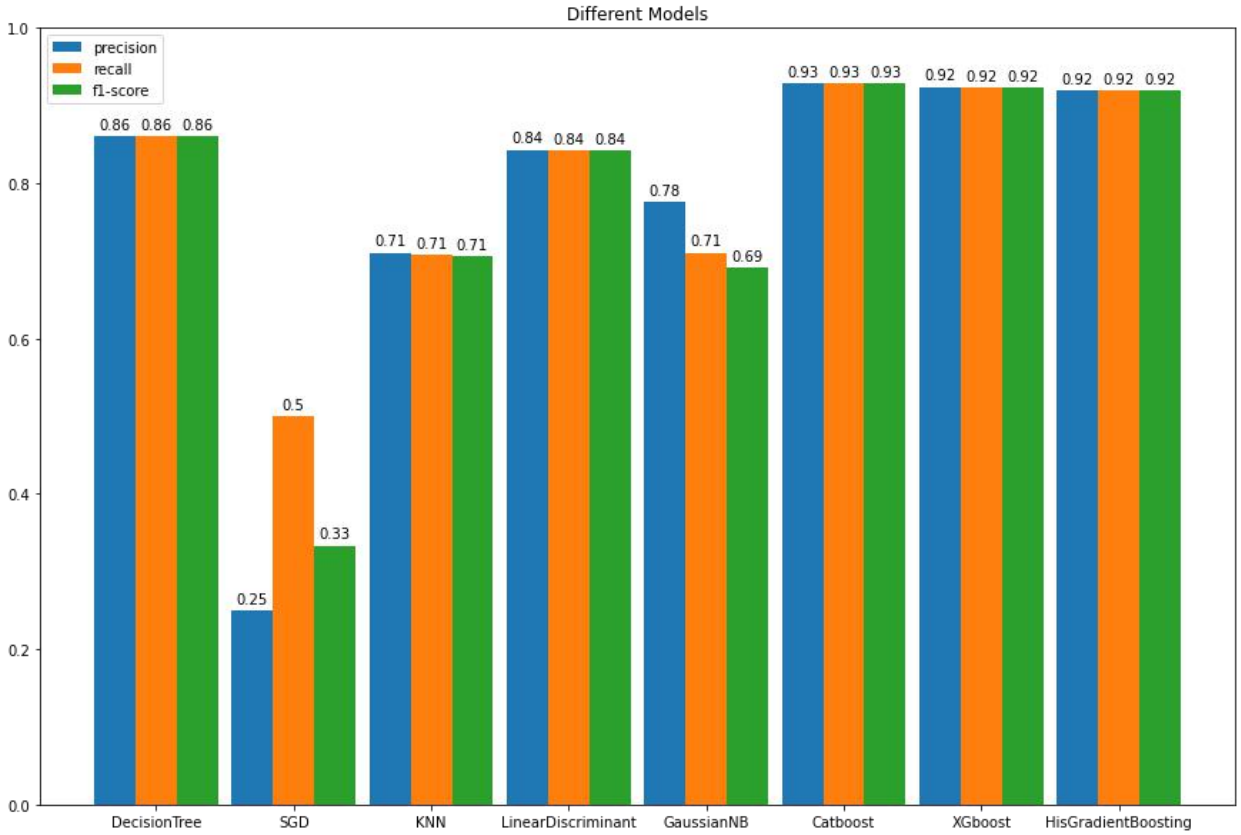
range of learning rate is [0.01, 0.05, 0.1]. Finally, the best parameters found by grid search are depth 6 and learning rate 0.1. And during this seach, the best F1 score is 0.8561.

We also try to increase the F1 score by adding the procedure of cross validation. After applying Kfold (Nfold = 5) and StratifiedShuffleSplit (n_split = 5) on Catboost with dataset D2 respectively, the F1 score has increased to 0.8737 and 0.8733.

```
Shrink model to first 1401 iterations.
F1 score test 0.8737467146889905
```

```
Shrink model to first 1679 iterations.
score 0.8733794716833999
```

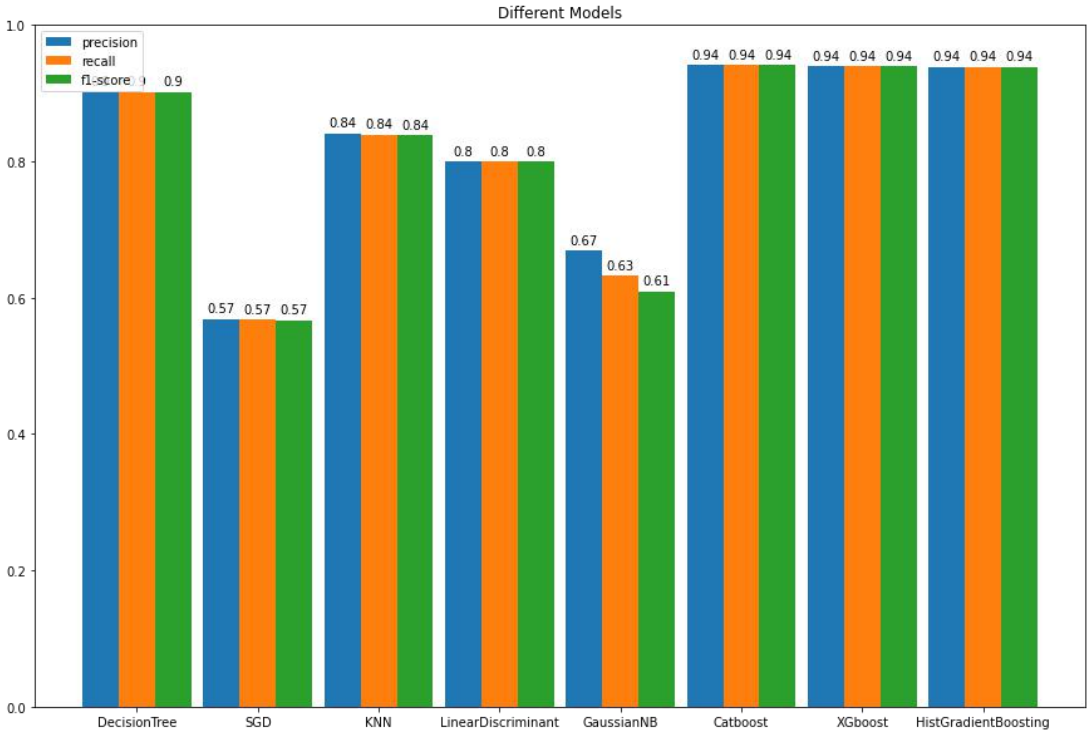
D3: Wanqing final dataset



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 98389 entries, 0 to 49220
Data columns (total 37 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   last_redemption_date   98389 non-null  int64
 1   first_activation_date  98389 non-null  int64
 2   total_redemptions      98389 non-null  int64
 3   tenure                 98389 non-null  int64
 4   number_upgrades        98389 non-null  int64
 5   year                   98389 non-null  int64
 6   manufacturer           98389 non-null  int64
 7   operating_system       98389 non-null  int64
 8   opt_out_mobiles_ads    98389 non-null  float64
 9   state                  98389 non-null  int64
 10  total_revenues_bucket  98389 non-null  int64
 11  activation_counts      98389 non-null  float64
 12  deactivation_counts    98389 non-null  float64
 13  interection_counts     98389 non-null  float64
 14  ival_counts            98389 non-null  float64
 15  iscompleted            98389 non-null  float64
 16  total_kb               98389 non-null  float64
 17  voice_minutes          98389 non-null  float64
 18  total_sms              98389 non-null  float64
 19  notify_counts          98389 non-null  float64
...
 35  first_activation       98389 non-null  float64
 36  redemption_duaration   98389 non-null  float64
dtypes: float64(25), int64(12)
```

D3 has some improvements over D2. Firstly, the colums with 70% missing values were deleted in stead of 50%, and the missing values were changed in three ways: 0, mean and mode. Through comparison, we found that the prediction accuracy is higher when the missing values are changed to 0. Secondly, the data in suspensions_ebb and loyalty_program_ebb are added to expand the scope of training data in general. This modification improved the F1 score to approximately 0.923.

D4: Lucas final dataset




```
<class 'pandas.core.frame.DataFrame'>
Index: 98389 entries, e3da0f9e3076cdf64decd7cb79dc174a51f874a3 to 56ae
Data columns (total 37 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   total_quantity           98389 non-null  float64
 1   auto_refill_enroll_date  98389 non-null  int64
 2   reactivation_channel     98389 non-null  float64
 3   lease_status             98389 non-null  int64
 4   case_type                98389 non-null  float64
 5   activation_channel       98389 non-null  float64
 6   revenues                 98389 non-null  float64
 7   memory_available         98389 non-null  float64
 8   data_network_type        98389 non-null  float64
 9   sd_storage_present       98389 non-null  object
 10  bluetooth_on             98389 non-null  bool
 11  memory_total             98389 non-null  float64
 12  language                 98389 non-null  float64
 13  data_roaming             98389 non-null  bool
 14  hotspot_kb               98389 non-null  float64
 15  total_sms                98389 non-null  float64
 16  voice_minutes            98389 non-null  float64
 17  total_kb                 98389 non-null  float64
 18  last_redemption_date     98389 non-null  object
 19  first_activation_date    98389 non-null  object
 20  total_redemptions        98389 non-null  int64
 21  tenure                   98389 non-null  int64
 22  number_upgrades          98389 non-null  int64
 23  year                     98389 non-null  int64
 24  manufacturer             97571 non-null  object
 25  operating_system         97571 non-null  object
 26  language_preference      9394 non-null   object
 27  opt_out_email            5619 non-null   float64
 28  opt_out_loyalty_email    159 non-null    float64
 29  opt_out_loyalty_sms      173 non-null    float64
 30  opt_out_mobiles_ads      49077 non-null  float64
 31  opt_out_phone            8573 non-null   float64
 32  state                    74402 non-null  object
 33  total_revenues_bucket    98389 non-null  int64
 34  marketing_comms_1        3827 non-null   float64
 35  marketing_comms_2        3817 non-null   float64
 36  ebb_eligible             98389 non-null  int64
dtypes: bool(2), float64(20), int64(8), object(7)
```

team15	2022-04-17_final	0.957859

Discussion

We have used eight different models to analyze the data which are decision tree, Stochastic Gradient Descent(SGD), k-nearest neighbors(KNN), Linear Discriminant, Naive Bayes, Catboost, XGboost and LightGBM. After comparing all the different models we have found that LightGBM is the best choice in this case.

The decision tree is a tree structure in which each internal node represents a test on an attribute, each branch represents a test output, and each leaf node represents a category. It is based on the known probability of occurrence of various situations, by forming a decision tree to find the probability that the expected value of the net present value is greater than or equal to zero, evaluate the project risk, and judge its feasibility. Decision tree is easy to understand and explain, it can be analyzed visually, and rules can be easily extracted. It can process nominal and numerical data at the same time and it is more suitable for processing samples with missing attributes. Also it is able to handle irrelevant features. However, for the decision tree, overfitting is prone to occur and it is easy to ignore the correlation of attributes in the data set. Also, for those data with inconsistent numbers of samples in each category in the decision tree. When classifying attributes, different judgment criteria will bring different attribute selection tendencies, the information gain criterion has a preference for a larger number of attributes, while the gain rate criterion (CART) has a preference for a smaller number of attributes, but when CART divides attributes, it no longer simply uses the gain rate to divide it attentively, but adopts a heuristic rule.

For SGD, it is iteratively updated once per sample. If the sample size is large, then only tens of thousands or thousands of samples may be used to iterate theta to the optimal solution.
Compared with the Batch gradient descent requires hundreds of thousands of training samples for one iteration. One iteration cannot be optimal. If there are 10 iterations, the training samples need to be traversed 10 times. This model has a very fast training speed, but the accuracy will be dropped and it is not globally optimal, also it is not easy to implement in parallel.

The KNN algorithm is very simple and very efficient. The model representation of KNN is the entire training dataset. KNNs can require a lot of memory or space to store all the data, but only compute when predictions are needed. The advantages of the KNN algorithm is that the theory is

mature and the thinking is simple, which can be used for both classification and regression and it can be used for nonlinear classification. The training time complexity of the KNN algorithm is O(n) which is quite fast and it has no assumptions about the data, high accuracy, and is insensitive to outliers. KNN is an online technology where new data can be added directly to the dataset without retraining and the theory of KNN is simple and easy to implement. On the other hand, when some classes have a large number of samples and others have a small number of samples, the sample imbalance problem will occur. It requires a lot of memory and for data sets with large sample sizes, the amount of calculation is relatively large. When the sample is unbalanced, the prediction bias is relatively large. Also, KNN will perform a global operation again for each classification and the choice of k value size is not theoretically optimal, and it is often combined with K-fold cross-validation to obtain the optimal k value selection.

Linear Discriminant contains statistical properties of the data, computed for each class. For a single input variable, it contains the average value for each category and the variance calculated across all categories.

Naive Bayes is a simple but surprisingly powerful predictive modeling algorithm. The model consists of two types of probabilities that can be computed directly from your training data which are the probability of each class and gives the conditional probability of each class for each x value. Once calculated, probabilistic models can be used to make predictions on new data using Bayes' theorem. When your data are real-valued, a Gaussian distribution is usually assumed so that we can easily estimate these probabilities. The Naive Bayesian model has stable classification efficiency since it originated from classical mathematical theory and has a solid mathematical foundation. It has a high speed when training and querying large numbers. Even with very large training sets, there is usually only a relatively small number of features per item, and training and classifying items is nothing more than a mathematical operation of feature probabilities. It performs well on small-scale data, can handle multi-classification tasks, and is suitable for incremental training. It is also hasless sensitive to missing data and the algorithm is relatively simple which means Naive Bayes is easy to understand for interpretation of results. At the same time, the Naive Bayes’ theorem will have some issues such as needing to calculate the prior probability and there is an error rate in classification decisions. It is also sensitive to the representation of the input data. In the end, It does not work well if the sample attributes are

correlated due to the assumption of independence of the sample attributes.

CatBoost is a GBDT framework with fewer parameters, support for categorical variables and high accuracy based on an oblivious tree as the base learner. [1]The main pain point to solve is to efficiently and reasonably process categorical features. In addition, CatBoost also solves the problem of gradient bias and prediction bias, thereby reducing the occurrence of overfitting, thereby improving the accuracy and generalization ability of the algorithm. It Embedded innovative algorithms that automatically process categorical features into numerical features. First, do some statistics on categorical features, calculate the frequency of a certain category feature, and then add hyperparameters to generate new numerical features. Catboost also uses combined category features, which can make use of the relationship between features, which greatly enriches the feature dimension. The method of ranking boosting is used to fight against the noise points in the training set, so as to avoid the bias of the gradient estimation, and then solve the problem of prediction offset.

XGboost is an algorithm toolkit (including engineering implementation) based on the Boosting framework, which is very powerful in parallel computing efficiency, missing value processing, and prediction performance. It has High accuracy, high concurrency and can support custom loss function. It can also output feature importance, speed block, suitable as a weapon for high-dimensional feature selection. Adding a regular term to the objective function controls the complexity of the model and avoids overfitting. Xgboost Support column sampling, which is a random selection feature, which enhances robustness and insensitive to missing values, can learn a split vector containing missing value features. However, In each iteration, the entire training data must be traversed many times. If the entire training data is loaded into the memory, the size of the training data will be limited, if it is not loaded into the memory, repeatedly reading and writing the training data will consume a lot of time. The pre-sort method consumes a lot of space, since XGboost uses feature-based parallel computation, features are sorted before each computation. Such an algorithm needs to save the eigenvalues of the data, and also save the result of the feature sorting and this requires twice as much memory as the training data.

LightGBM is boosting framework originally developed by Microsoft that sklearn has used as the inspiration for their function HistGradientBoostingClassifier. The benefit of this model is that it runs very quickly relative to its performance, which was helpful during the closing period of the

competition where server downtime was commonplace and model testing needed to be done quickly. LightGBM is useful for large datasets of greater than 10000 and especially for achieving high accuracy. It is quite similar to XGboost, however varies in that the tree does not grow level-wise but instead leaf-wise. Furthermore it doesn't use the sorted-based decision tree learning algorithm that XGboost and many other frameworks use, it instead implements a histogram-based decision tree algorithm, which is how it is able to run so quickly and use so little memory.

After comparing all the above models, we can find that for the first five basic models which are decision tree, Stochastic Gradient Descent(SGD), k-nearest neighbors(KNN), Linear Discriminant, Naive Bayes, tuning parameters will not help the final result have a higher precision, recall and F1 score. Even for KNN and Naive Bayes, they had a worse precision, recall and F1 score after tuning parameters. For the last two models: Catboost and XGboost, we can know that before tuning parameters the F1 score after using these two models are around 0.83 which is quite high and after tuning parameters the F1 score of using XGboost is around 0.85, of using Catboost is around 0.86. Therefore with the data support, we can clearly know that Catboost and XGboost are way better than the first five basic models.

Conclusion

In conclusion, our project is to use machine learning knowledge to predict customers who can qualify for the EBB program. During our working, we analyzed the data with many different ways to find the best dataset. After that, with the analyzed data we tried several models to train them to find the best performance, with many times of tuning parameter and testing with different models we finally get 0.957859 as our final result. With many times of testing and tuning parameters we found that Catboost, XGboost and HistGradiantboost have better performance than other 5 basic models. Although we get a quite high result, time limit is still a big problem. Since each test need to take about 20 minutes and the web server is not stable which means sometimes we might disconnected when we running the test and it cost us a lot of fixing these issues. Therefore, if we have more time, we can try more models with many more different datasets to get a higher result in the future.

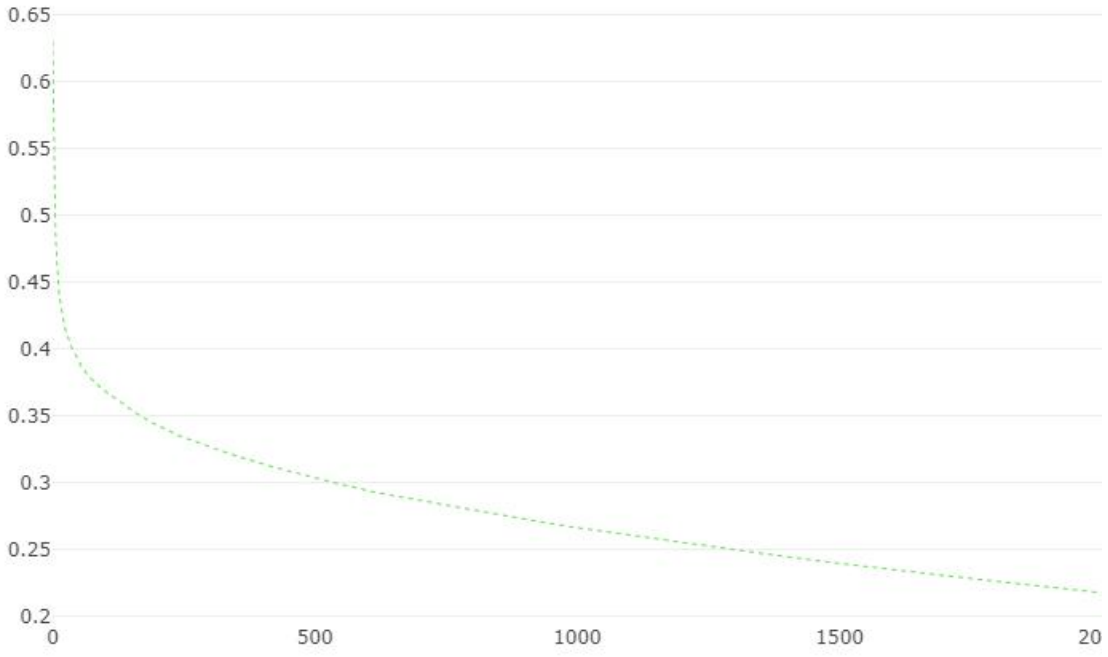
Reference

[1]Catboost document [Online]. Available: https://catboost.ai/en/docs/

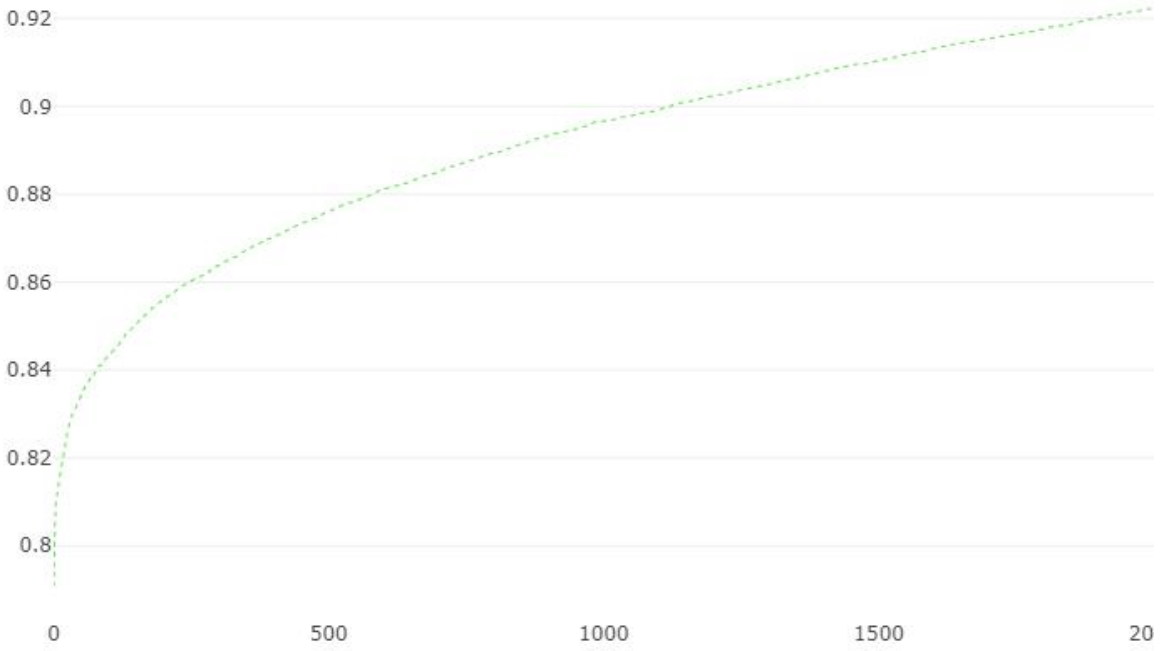
https://www.bea.gov/data/income-saving/real-personal-income-states-and-metropolitan-areas

https://www.forbes.com/sites/andrewdepietro/2021/12/28/us-per-capita-income-by-state-in-2021/?sh=2cc69f5737be

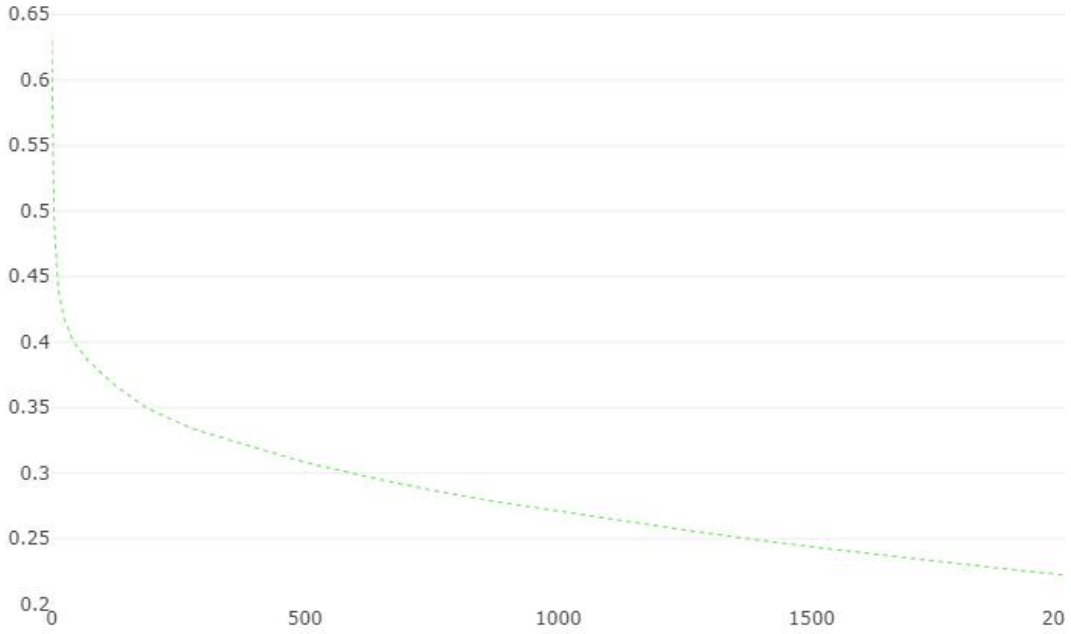
Appendix



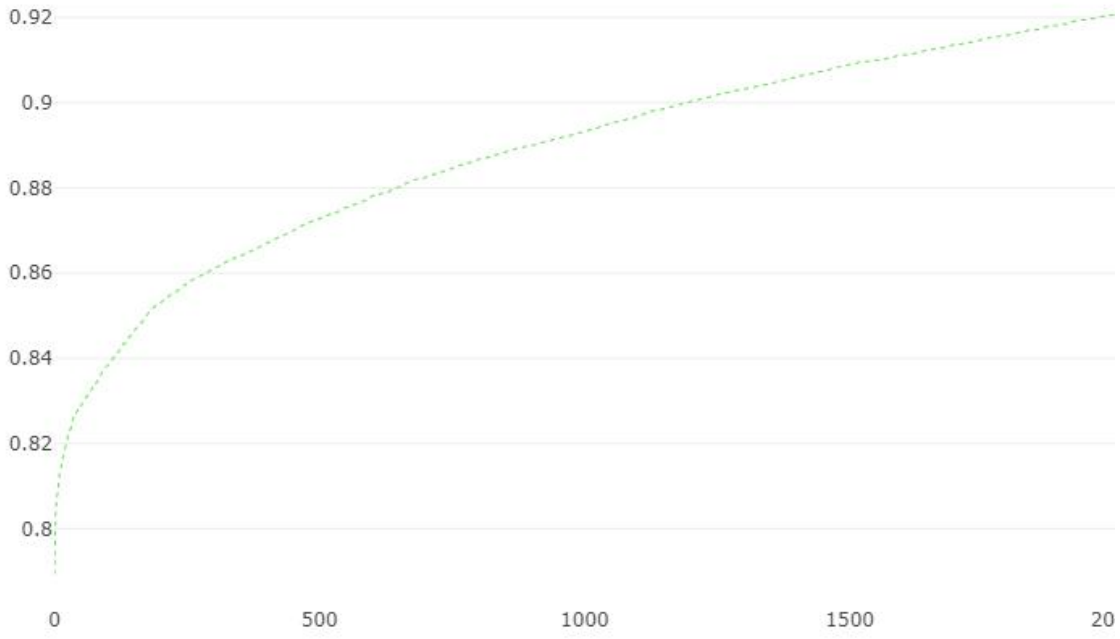
ce_all_D2



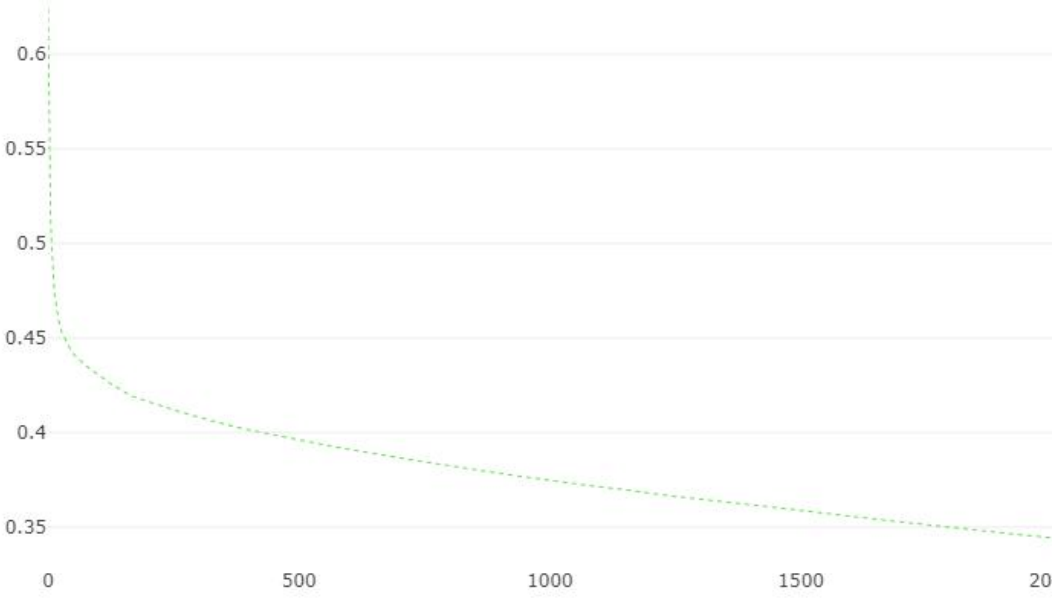
F1_all_D2



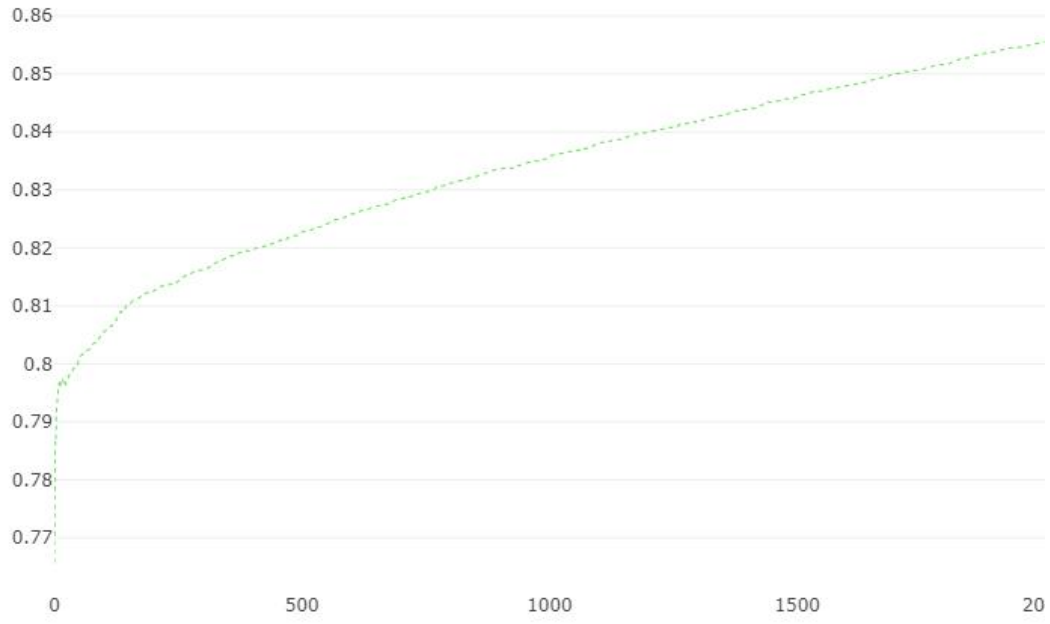
Fillmean_ce_D2



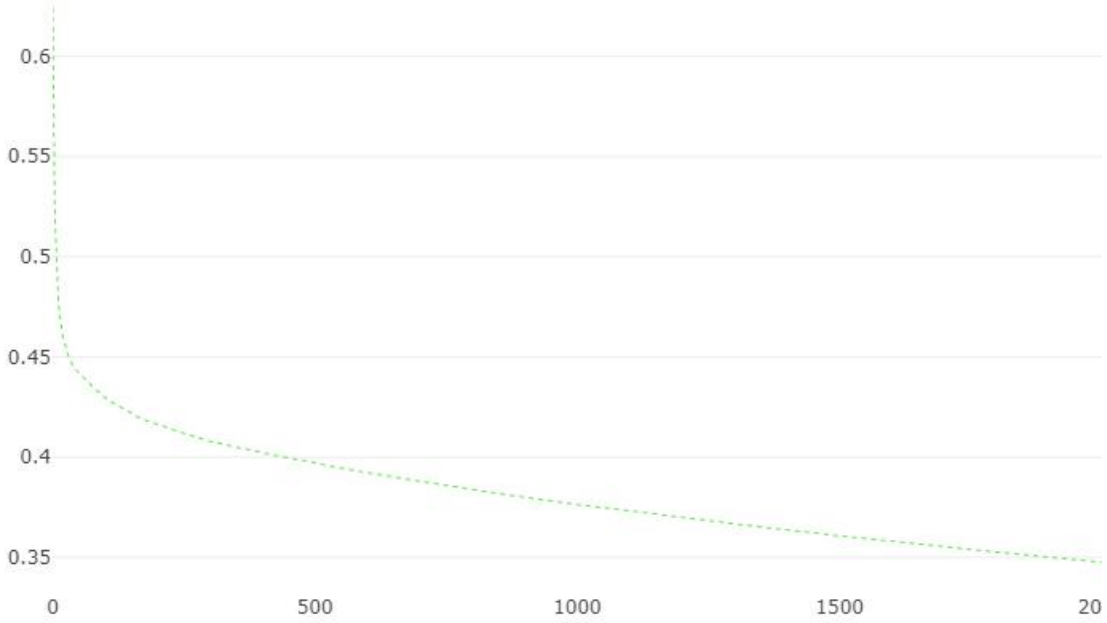
Fillmean_f1_D2



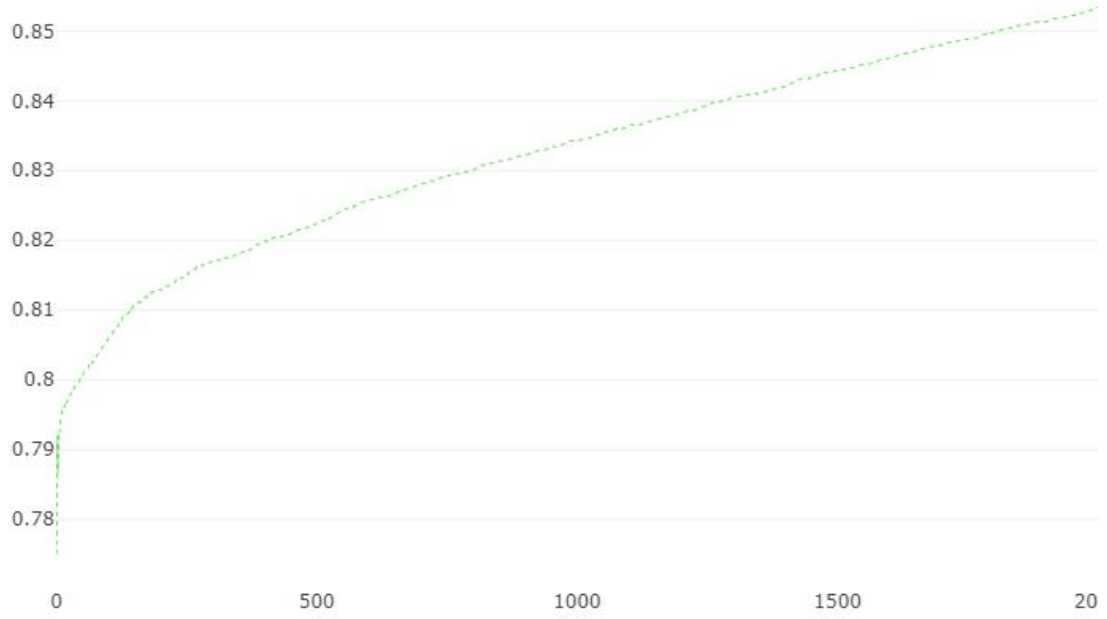
Onehot_ce_D2



Onehot_f1_D2



Origianl_sample_catboost_ce_D2



Origianl_sample_catboost_F1_D2