TalkingData Ad Tracking Fraud Detection

- Yipeng Hong, Zihan Zhang, Yidi Xu

ABSTRACT

TalkingData Ad Tracking Fraud Detection is a challenge that took place on Kaggle Competition Three months ago. This project chose this competition as topic to predict whether it is a fraudulent click for mobile app ads covered by the test set, using information available three months ago. This project used basic features provided by dataset, proposed several new features to improve the accuracy of prediction, and applied four machine learning models – Random Forest, Gradient Boosting & AdaBoost, Logistic Regression, and SVM, three deep learning algorithms, ANN, RNN and MLP. The solution is evaluated with ROC curve. The best result is of 100%.

Keywords: TalkingData Ad Tracking Fraud, Random Forest, Gradient Boosting & AdaBoost, Logistic Regression, SVM, ANN, RNN, MLP

1 Introduction

1.1 Background

Fraud risk is everywhere, but for companies that advertise online, click fraud can happen at an overwhelming volume, resulting in misleading click data and wasted money. Ad channels can drive up costs by simply clicking on the ad at a large scale. Ad fraud is defined as any deliberate activity that prevents ads from being served to human users. It may cause illegal bot activity, historically fraudulent URLs, or specific page-level fraud. Furthermore, it costs the industry $8.2 billion a year in the U.S. alone,

according to the IAB. As advertising budgets keep shifting to digital, fraudsters have found ways to game the system and earn money by serving ads in ways that have no potential to be seen by a real person. Needless to say, this has a negative effect on the entire advertising community.

fig 1. The real cost of ad fraud

China is the largest mobile market in the world and therefore suffers from huge volumes of fraudulent traffic, With over 1 billion smart mobile devices in active use every month.

TalkingData, China's largest independent big data service platform, covers over 70% of active mobile devices nationwide. It handles 3 billion clicks per day, of which 90% are potentially fraudulent. In order to always be one step ahead of fraudsters, we will build algorithms that predicts whether a user will download an app after clicking a mobile app ad. In this way, if the result is satisfactory, it can generate a model for TalkingData to build up its strategy to prevent from users clicking a fraudulent ad.

1.2 Given Datasets

Four datasets are provided with the information for the TalkingData Ad Tracking Fraud Detection Competition:

1. train.csv - the training set;
2. train_sample.csv - 100,000 randomly-selected rows of training data, to inspect data before downloading full set;
3. test.csv - the test set;
4. sampleSubmission.csv - a sample submission file in the correct format.

Each row of the training data contains a click record, with the following features.

- Ip: ip address of click.
- App: app id for marketing.
- Device: device type id of user mobile phone (e.g., iphone 6 plus, iphone 7, huawei mate 7, etc.)
- Os: os version id of user mobile phone
- Channel: channel id of mobile ad publisher
- Click_time: timestamp of click (UTC)
- Attributed_time: if user download the app for after clicking an ad, this is the time of the app download
- Is_attibuted: the target that is to be predicted, indicating the app was downloaded

Note that ip, app, device, os, and channel are encoded.

The test data is similar, with the following differences:

- Click_id: reference for making predictions
- Is_attributed: not included

1.3 Submission and Evaluation

For this competition, Kaggle evaluates the submissions on area under the ROC curve between the predicted probability and the observed target. ROC Curve is A receiver operating characteristic curve, i.e. ROC

curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

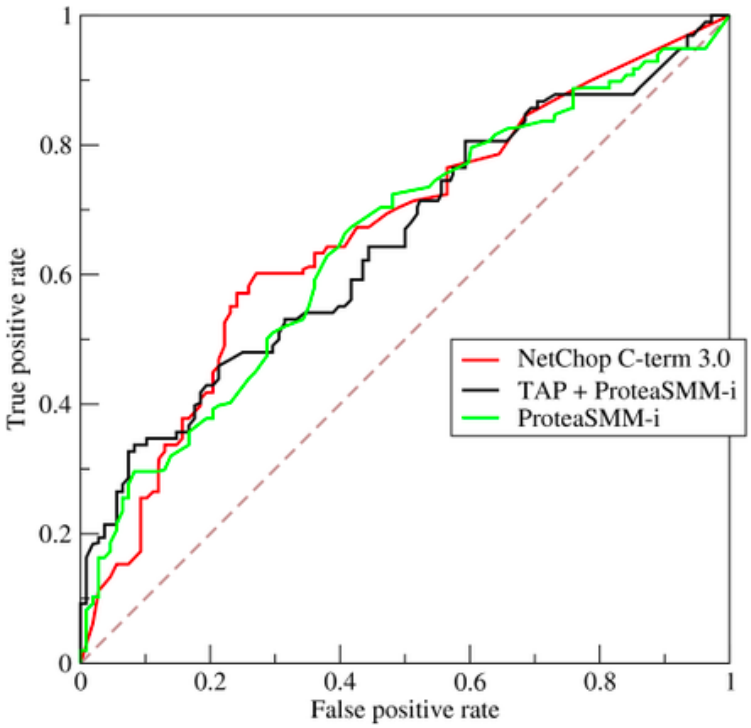


fig 2. ROC Curve of three predictors of peptide cleaving in the proteasome

When using normalized units, the area under the curve (often referred to as simply the AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative'). This can be seen as follows: the area under the curve is given by (the integral boundaries are reversed as large T has a lower value on the x-axis)

$$A = \int_{\infty}^{-\infty} \mathrm{TPR}(T)\mathrm{FPR}'(T)\,dT = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(T' > T) f_1(T') f_0(T)\,dT'$$

where X_1 the score for a positive instance and X_0 is the score for a negative instance, and f_0 and f_1 are probability densities as defined in previous section. [1]

1.4 Competition Notebook

We did some research before starting our project, which is like kagglers did. The summary is as follows:

1. By the end of this competition, there were 3,967 teams, 4,758 competitors took part in, with 69,320 entries.
2. It is a big challenge for those competitors, of which 90% were using EDA and LightGBM to explore data, with fewer ones using other methods, for example, Xgboost.
3. The reason why this happen is that EDA could maximize insight into the TalkingData Ad Fraud data set, uncover underlying stricture of it, extract important variables, detect outliers and anomalies, test underlying assumptions, develop parsimonious models and determine optimal factor settings. LightGBM could faster training speed and higher efficiency, lower memory usage, better accuracy than any other boosting algorithm, compatibility with large datasets, and parallel learning support.
4. Some good points illustrated in this competition:

- There is little relationship among ip, app, os, device and channel, only app somehow correlates with channel**.**
- The most important feature by gain is app, by cover and frequency is channe.
- There is no clear hourly time pattern in ratios.
- There is a definite pattern in frequency of clicks based on time of day.
- In LightGBM, it looks like tuning of scale_pos_weight can make the model better.
- The best score of LB is 0.9769.

There will be some extension in our project:

1. Except analysis the basic features, this project will create additional features in data exploration to give more categories.
2. Using more algorithms, including five machine learning models – Random Forest, Gradient Boosting & AdaBoost, Logistic Regression, SVM, Ensemble Learning and the combination of the five models, two deep learning, ANN & CNN to improve the accuracy of prediction.
3. Creating a new algorithm, which is created by allocating the percentage to each better algorithm and sum up these to get a new one.
4. Comparing the two outputs, using the observed one and the predicted one.

2 Data Cleaning and Exploration

2.1 Data Cleaning

The original training set is too large, in this project, it will be divided into two subsets, of which we will use 200,000 observations to do the exploration.

The testing set is the complement of the training set, that is, it is composed of the 50,000 observations.

After getting the tow datasets, this project obtained some simple descriptive statistics.

```
In [878]: df_train.head(10)
```

Out[878]:

	ip	app	device	os	channel	click_time	attributed_time	is_attributed
0	83230	3	1	13	379	2017-11-06 14:32:21	NaN	0
1	17357	3	1	19	379	2017-11-06 14:33:34	NaN	0
2	35810	3	1	13	379	2017-11-06 14:34:12	NaN	0
3	45745	14	1	13	478	2017-11-06 14:34:52	NaN	0
4	161007	3	1	13	379	2017-11-06 14:35:08	NaN	0
5	18787	3	1	16	379	2017-11-06 14:36:26	NaN	0
6	103022	3	1	23	379	2017-11-06 14:37:44	NaN	0
7	114221	3	1	19	379	2017-11-06 14:37:59	NaN	0
8	165970	3	1	13	379	2017-11-06 14:38:10	NaN	0
9	74544	64	1	22	459	2017-11-06 14:38:23	NaN	0

fig 3. The Head of Training Set

```
In [879]: df_train['is_attributed'].value_counts()
```

```
Out[879]: 0    199652
          1       348
          Name: is_attributed, dtype: int64
```

```
In [880]: sns.countplot(x='is_attributed', data=df_train, palette='hls')
          plt.show()
```

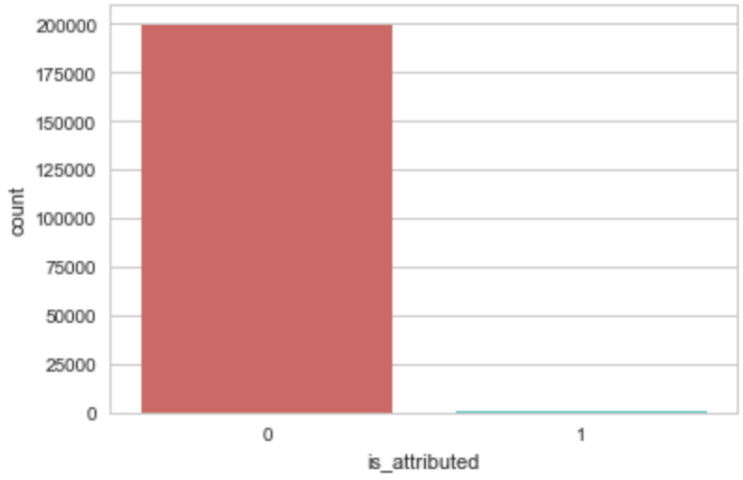


fig 4. The Count Plot of 'is_attributed' Column

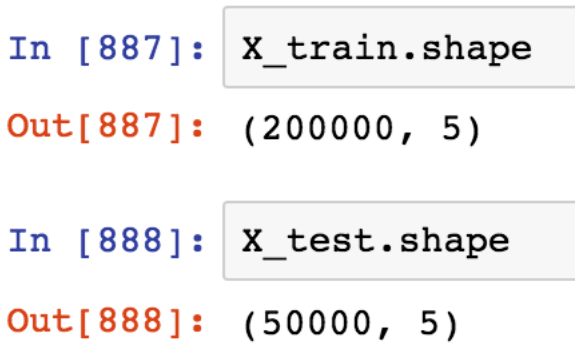


fig 5. Basic Shape of Training and Testing Set

From these statistics, we could observe several important features:

- There is little relationship among ip, app, os, device and channel, only app somehow correlates with channel**.**
- There is no clear hourly time pattern in ratios.
- There are more values lie in 1 rather than 0 of column 'is_attributed'.

2.2 Data Exploration /Visualization

Based on the given features, we could explore the relationship between any two of them.

Basic Features

For this part, we could learn from the kaggle competition that there is little relationship among ip, app, os, device and channel, only app somehow correlates with channel. However, the basic features are not enough to get the most important accuracy. To improve the accuracy of this project, we will create additional features to support.

Additional Features

We will create five additional features: Confidence rate for is_attributed, Group-by-aggregation, Extracting time information, Time till next click and Click on app ads before & after.

2.2.1 Confidence rate for is_attributed

$$\text{conf}_{\text{is\_attributed}} = \frac{\log(\text{views}_{\text{category\_1}})}{\log(100000)}$$

This is the mathematics formula of computing the confidence rate of column 'is_attributed'.

```
ATTRIBUTION_CATEGORIES = [
    # V1 Features #
    ###############
    ['ip'], ['app'], ['device'], ['os'], ['channel'],

    # V2 Features #
    ###############
    ['app', 'channel'],
    ['app', 'os'],
    ['app', 'device'],

    # V3 Features #
    ###############
    ['channel', 'os'],
    ['channel', 'device'],
    ['os', 'device']
]
```

fig 6. Attribution Categories

```
# Aggregation function
def rate_calculation(x):
    """Calculate the attributed rate. Scale by confidence"""
    rate = x.sum() / float(x.count())
    conf = np.min([1, np.log(x.count()) / log_group])
    return rate * conf

# Perform the merge
X_train = X_train.merge(
    group_object['is_attributed']. \
        apply(rate_calculation). \
        reset_index(). \
        rename(
            index=str,
            columns={'is_attributed': new_feature}
        )[cols + [new_feature]],
    on=cols, how='left'
)
```

fig 7. Aggregation Function

app_confRate_y	device_confRate_y	os_confRate_y	channel_confRate_y	app_channel_confRate_y	app_os_confRate_y	app_device_confRate_y	channel_os_confRate_y	channe
0.000446	0.001193	0.001146	0.000497	0.000360	0.000366	0.000444	0.000000	
0.000446	0.001193	0.001394	0.000497	0.000360	0.000275	0.000444	0.000235	
0.000446	0.001193	0.001146	0.000497	0.000360	0.000366	0.000444	0.000000	
0.000431	0.001193	0.001146	0.004071	0.004164	0.000201	0.000445	0.001835	
0.000446	0.001193	0.001146	0.000497	0.000360	0.000366	0.000444	0.000000	

fig 8. Output of Confidence Rate of 'is_attributed'

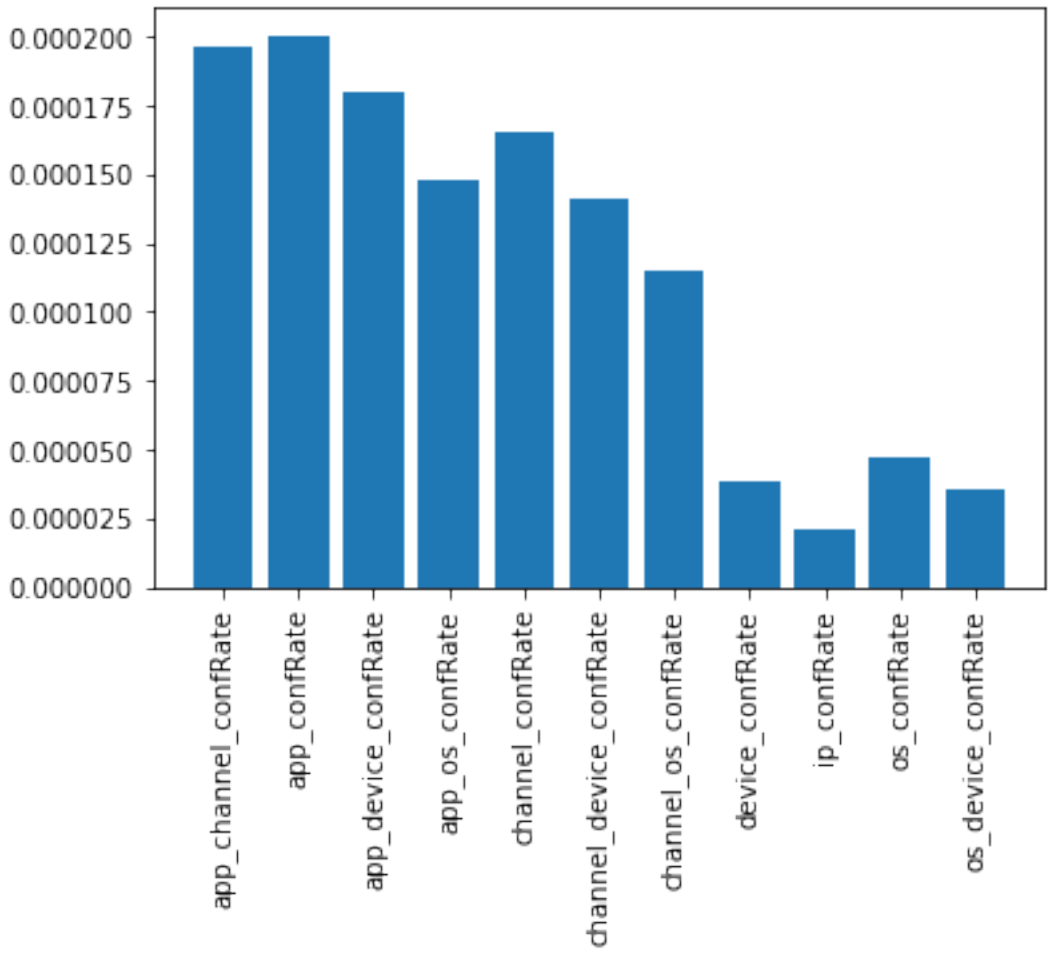


fig 9. Correlation of Confidence Rate

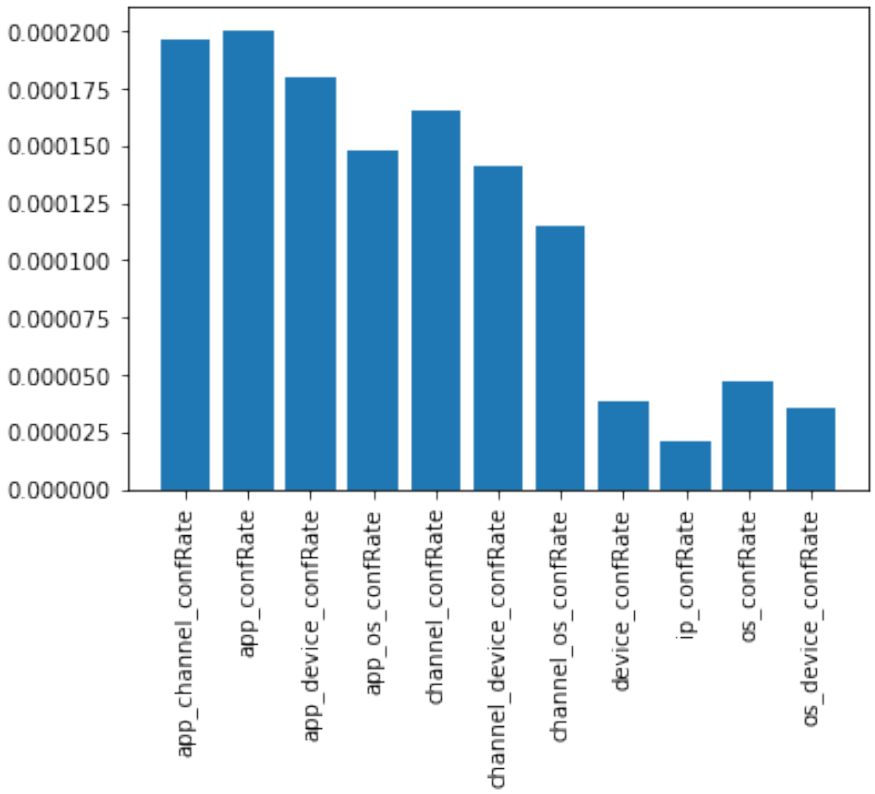


fig 10. Covariance of Confidence Rate

2.2.2 Group-by-aggregation

```
# V3 - GroupBy Features                                                #
# https://www.kaggle.com/bk0000/non-blending-lightgbm-model-lb-0-977 #
########################################################################
{'groupby': ['ip'], 'select': 'channel', 'agg': 'nunique'},
{'groupby': ['ip'], 'select': 'app', 'agg': 'nunique'},
{'groupby': ['ip','day'], 'select': 'hour', 'agg': 'nunique'},
{'groupby': ['ip','app'], 'select': 'os', 'agg': 'nunique'},
{'groupby': ['ip'], 'select': 'device', 'agg': 'nunique'},
{'groupby': ['app'], 'select': 'channel', 'agg': 'nunique'},
{'groupby': ['ip', 'device', 'os'], 'select': 'app', 'agg': 'nunique'},
{'groupby': ['ip','device','os'], 'select': 'app', 'agg': 'cumcount'},
{'groupby': ['ip'], 'select': 'app', 'agg': 'cumcount'},
{'groupby': ['ip'], 'select': 'os', 'agg': 'cumcount'},
{'groupby': ['ip','day','channel'], 'select': 'hour', 'agg': 'var'}
```

```
# V1 - GroupBy Features #
#########################
# Variance in day, for ip-app-channel
{'groupby': ['ip','app','channel'], 'select': 'day', 'agg': 'var'},
# Variance in hour, for ip-app-os
{'groupby': ['ip','app','os'], 'select': 'hour', 'agg': 'var'},
# Variance in hour, for ip-day-channel
{'groupby': ['ip','day','channel'], 'select': 'hour', 'agg': 'var'},
# Count, for ip-day-hour
{'groupby': ['ip','day','hour'], 'select': 'channel', 'agg': 'count'},
# Count, for ip-app
{'groupby': ['ip', 'app'], 'select': 'channel', 'agg': 'count'},
# Count, for ip-app-os
{'groupby': ['ip', 'app', 'os'], 'select': 'channel', 'agg': 'count'},
# Count, for ip-app-day-hour
{'groupby': ['ip','app','day','hour'], 'select': 'channel', 'agg': 'count'},
# Mean hour, for ip-app-channel
{'groupby': ['ip','app','channel'], 'select': 'hour', 'agg': 'mean'},

# V2 - GroupBy Features #
#########################
# Average clicks on app by distinct users; is it an app they return to?
{'groupby': ['app'],
 'select': 'ip',
 'agg': lambda x: float(len(x)) / len(x.unique()),
 'agg_name': 'AvgViewPerDistinct'
},
# How popular is the app or channel?
{'groupby': ['app'], 'select': 'channel', 'agg': 'count'},
{'groupby': ['channel'], 'select': 'app', 'agg': 'count'},
```

fig 11. Group-By Feature

```
# Perform the groupby
gp = X_train[all_features]. \
    groupby(spec['groupby'])[spec['select']]. \
    agg(spec['agg']). \
    reset_index(). \
    rename(index=str, columns={spec['select']: new_feature})

# Merge back to X_total
if 'cumcount' == spec['agg']:
    X_train[new_feature] = gp[0].values
else:
    X_train = X_train.merge(gp, on=spec['groupby'], how='left')
```

fig 12. Perform the Group-by

ip_nunique_device	app_nunique_channel	ip_device_os_nunique_app	ip_device_os_cumcount_app	ip_cumcount_app	ip_cumcount_os	ip_day_channel_var_hour_y
1	34	16	0	0	0	1.333333
1	34	11	0	0	0	1.000000
3	34	10	0	0	0	2.000000
6	26	24	0	0	0	1.000000
1	34	1	0	0	0	NaN

fig 13. Output of Group-by Feature

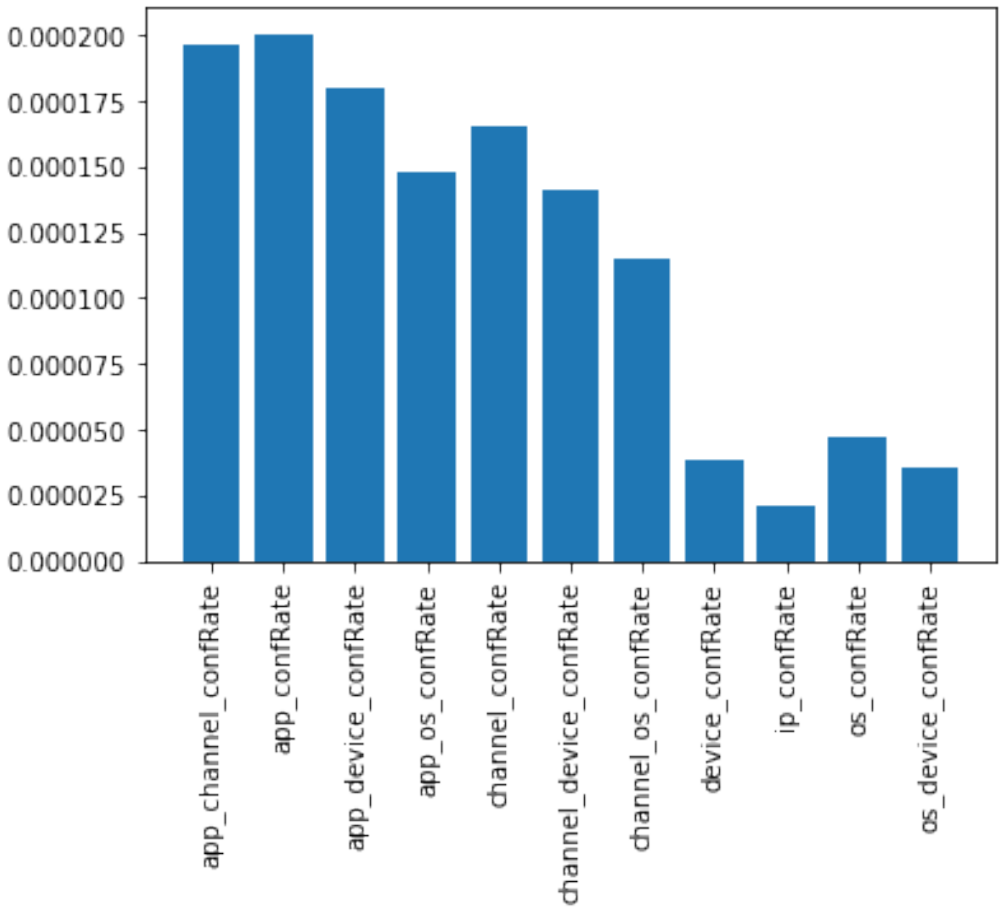


fig 14. Correlation of Group-by-Aggregation

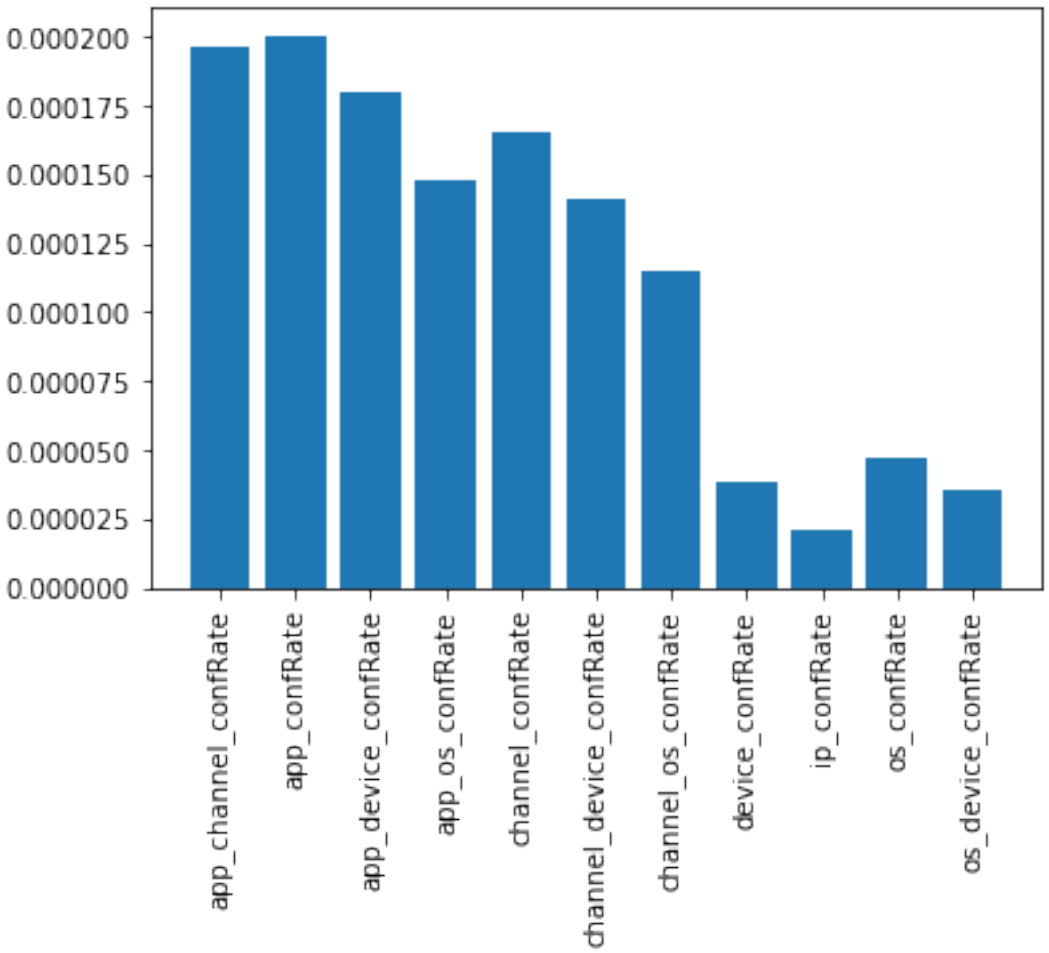


fig 15. Covariance of Group-by-Aggregation

2.2.3 Extracting time information

```
X_train['day'] = X_train['click_time'].dt.day.astype('uint8')
X_train['hour'] = X_train['click_time'].dt.hour.astype('uint8')
X_train['minute'] = X_train['click_time'].dt.minute.astype('uint8')
X_train['second'] = X_train['click_time'].dt.second.astype('uint8')
X_train.head()
```

fig 16. Extracting Time

	ip	app	device	os	channel	click_time	attributed_time	is_attributed	day	hour	minute	second
0	83230	3	1	13	379	2017-11-06 14:32:21	NaN	0	6	14	32	21
1	17357	3	1	19	379	2017-11-06 14:33:34	NaN	0	6	14	33	34
2	35810	3	1	13	379	2017-11-06 14:34:12	NaN	0	6	14	34	12
3	45745	14	1	13	478	2017-11-06 14:34:52	NaN	0	6	14	34	52
4	161007	3	1	13	379	2017-11-06 14:35:08	NaN	0	6	14	35	8

fig 17. Output of Extracting Time Information

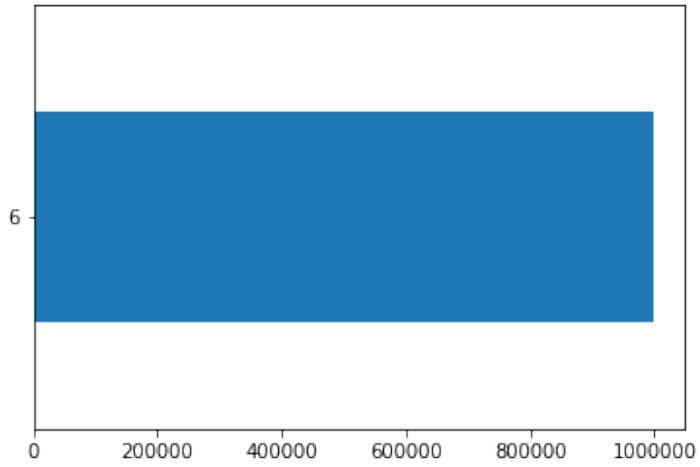


fig 18. Extracting to Day

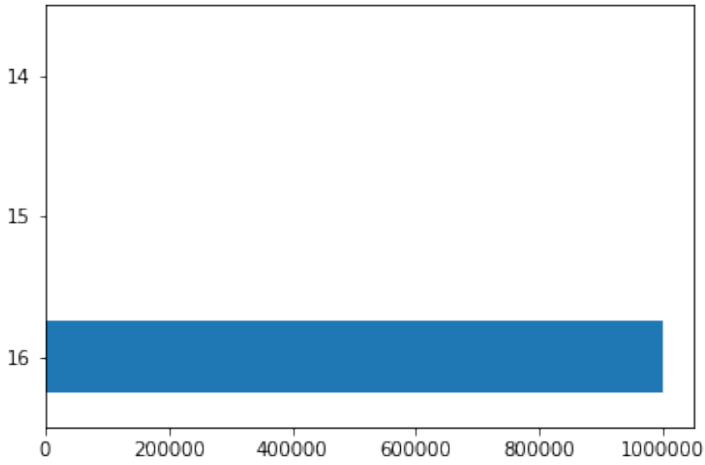


fig 19. Extracting to Hour

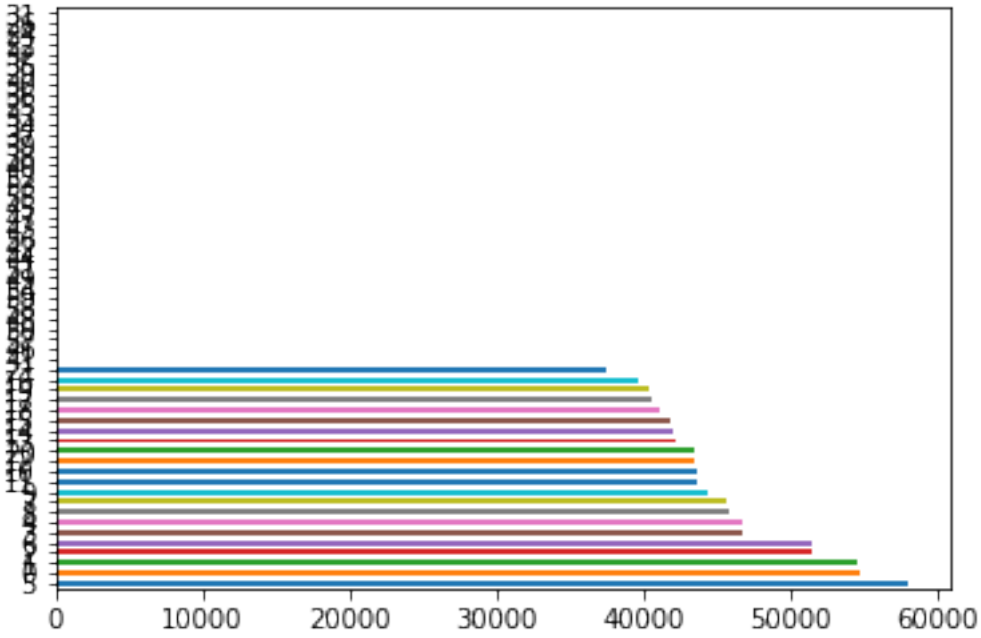


fig 20. Extracting to Minute

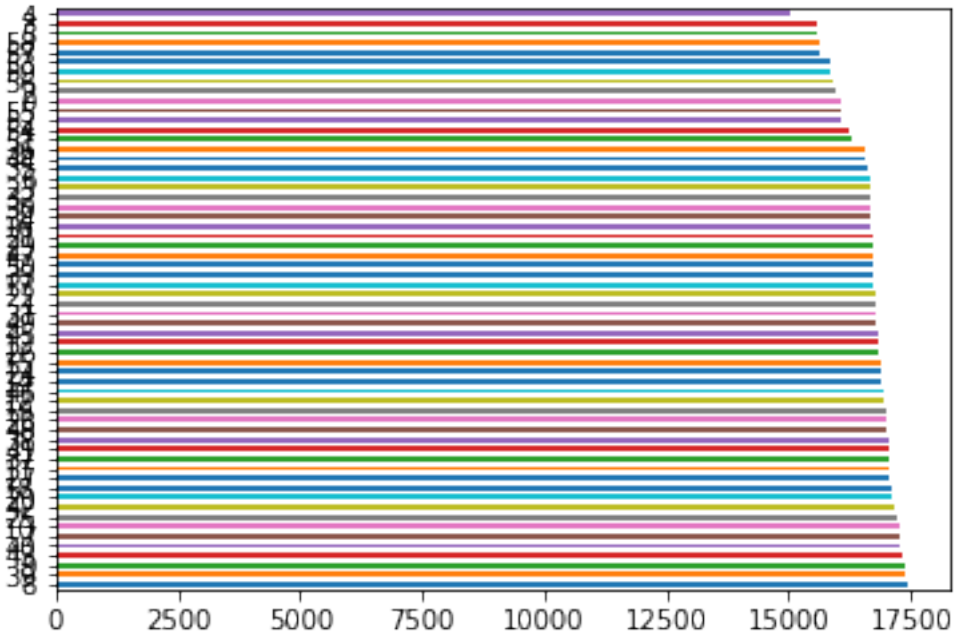


fig 21. Extracting to Second

2.2.4 Time till next click

```
GROUP_BY_NEXT_CLICKS = [

    # V1
    {'groupby': ['ip']},
    {'groupby': ['ip', 'app']},
    {'groupby': ['ip', 'channel']},
    {'groupby': ['ip', 'os']},

    # V3
    {'groupby': ['ip', 'app', 'device', 'os', 'channel']},
    {'groupby': ['ip', 'os', 'device']},
    {'groupby': ['ip', 'os', 'device', 'app']}
]

# Calculate the time to next click for each group
```

```
X_train[new_feature] = X_train[all_features].groupby(spec['groupby']).click_time.transform(lambda x: x.diff().
                                                                                           shift(-1)).dt.seconds
```

fig 22. Time till Next Click Feature

ip_nextClick	ip_app_nextClick	ip_channel_nextClick	ip_os_nextClick	ip_app_device_os_channel_nextClick	ip_os_device_nextClick	ip_os_device_app_nextClick
5290.0	5340.0	5444.0	5307.0	NaN	5307.0	5340.0
5177.0	5177.0	5177.0	5239.0	NaN	5239.0	5547.0
5175.0	5175.0	6005.0	5205.0	NaN	5205.0	5925.0
5108.0	5110.0	5137.0	5108.0	NaN	5108.0	5110.0
NaN	NaN	NaN	NaN	NaN	NaN	NaN

fig 23. Output of Time till Next Click

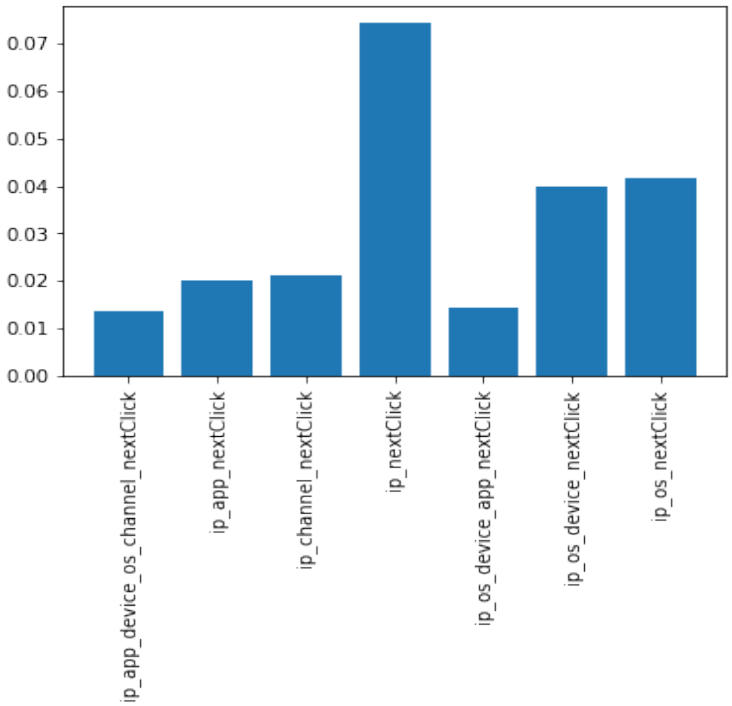


fig 24. Correlation of Time till Next Click

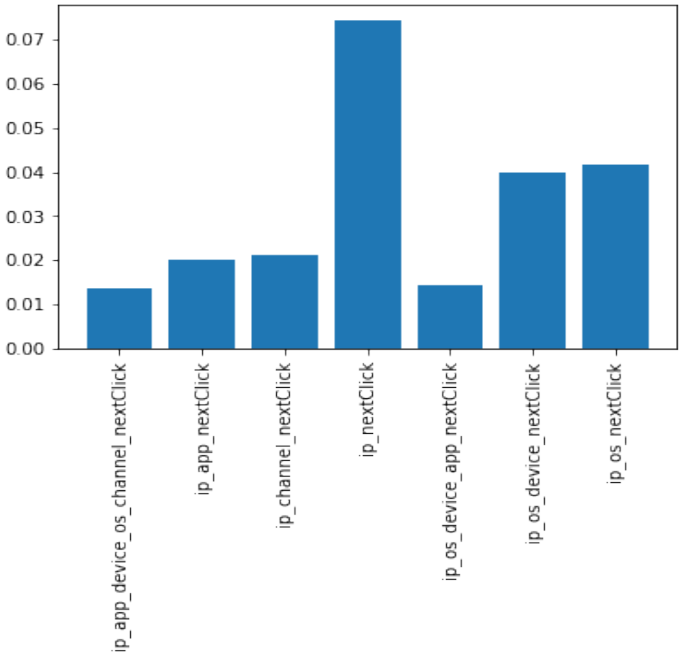


fig 25. Covariance of Time till Next Click

2.2.5 Click on app ads before & after

```
HISTORY_CLICKS = {
    'identical_clicks': ['ip', 'app', 'device', 'os', 'channel'],
    'app_clicks': ['ip', 'app']
}
```

```
# Go through different group-by combinations
for fname, fset in HISTORY_CLICKS.items():

    # Clicks in the past
    X_train['prev_'+fname] = X_train. \
        groupby(fset). \
        cumcount(). \
        rename('prev_'+fname)

    # Clicks in the future
    X_train['future_'+fname] = X_train.iloc[::-1]. \
        groupby(fset). \
        cumcount(). \
        rename('future_'+fname).iloc[::-1]
```

fig 26. Click on App Ads Before & After Feature

_device_os_channel_nextClick	ip_os_device_nextClick	ip_os_device_app_nextClick	prev_identical_clicks	future_identical_clicks	prev_app_clicks	future_app_clicks
NaN	5307.0	5340.0	0	0	0	18
NaN	5239.0	5547.0	0	0	0	22
NaN	5205.0	5925.0	0	0	0	9
NaN	5108.0	5110.0	0	0	0	68
NaN	NaN	NaN	0	0	0	0

fig 27. Output of Click on App Ads Before & After Feature

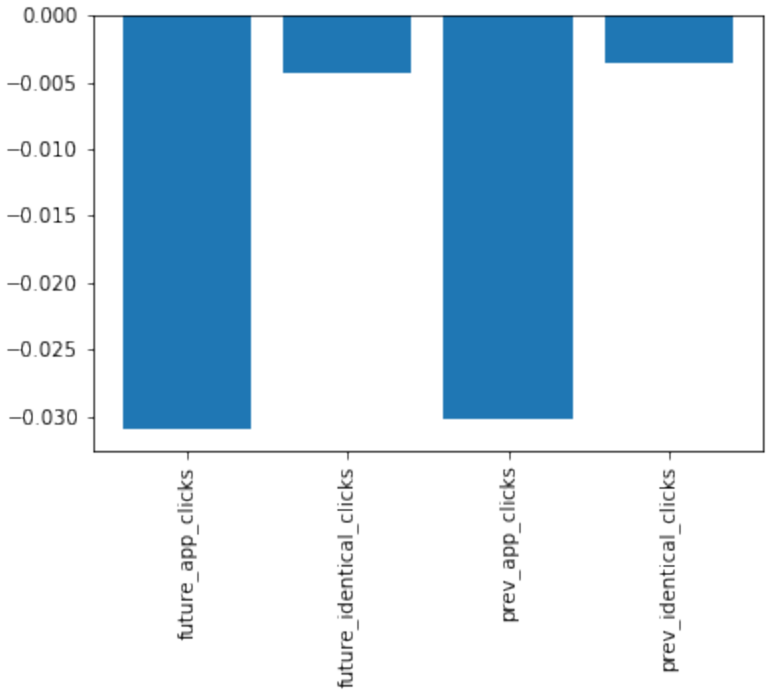


fig 28. Correlation of Click on App Ads Before & After Feature

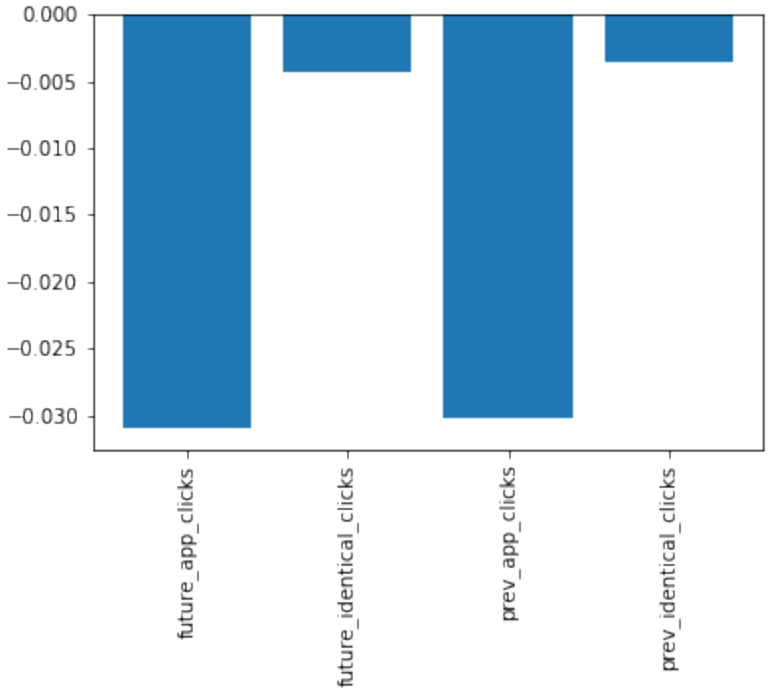


fig 29. Covariance of Click on App Ads Before & After Feature

2.2.6 Evaluation

```
clf_xgBoost = xgb.XGBClassifier(
    max_depth = 4,
    subsample = 0.8,
    colsample_bytree = 0.7,
    colsample_bylevel = 0.7,
    scale_pos_weight = 9,
    min_child_weight = 0,
    reg_alpha = 4,
    n_jobs = 4,
    objective = 'binary:logistic'
```

```
from sklearn import preprocessing

# Get xgBoost importances
importance_dict = {}
for import_type in ['weight', 'gain', 'cover']:
    importance_dict['xgBoost-'+import_type] = clf_xgBoost.get_booster().get_score(importance_type=import_type)

# MinMax scale all importances
importance_df = pd.DataFrame(importance_dict).fillna(0)
importance_df = pd.DataFrame(
    preprocessing.MinMaxScaler().fit_transform(importance_df),
    columns=importance_df.columns,
    index=importance_df.index
)

# Create mean column
importance_df['mean'] = importance_df.mean(axis=1)

# Plot the feature importances
importance_df.sort_values('mean').plot(kind='bar', figsize=(20, 7))
```

fig 30. Evaluation of Creating features

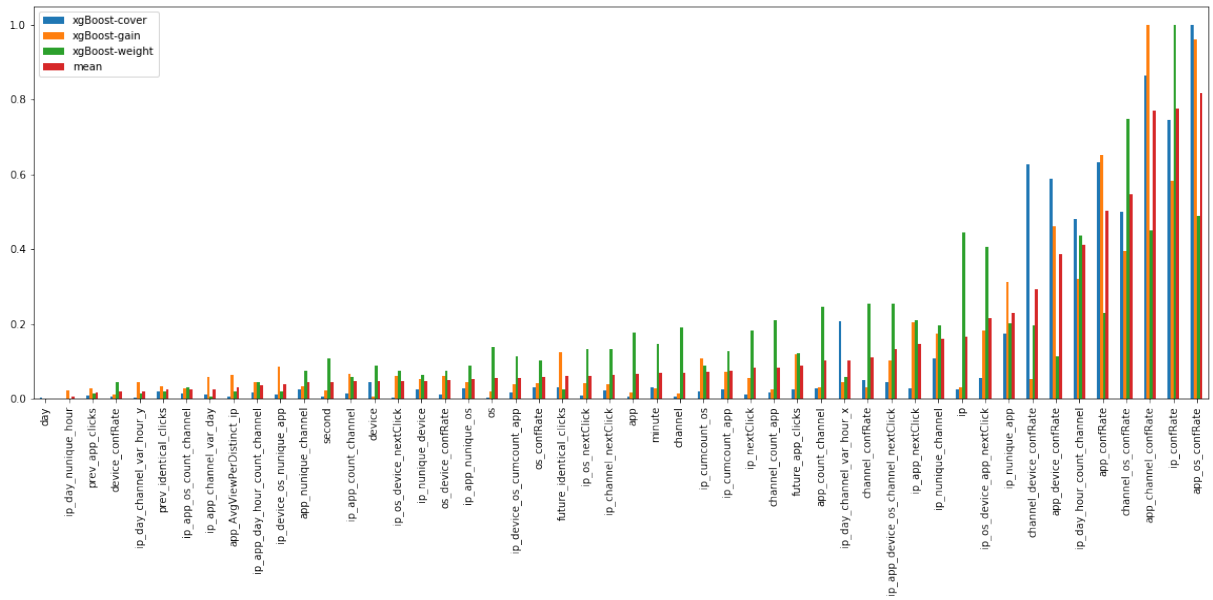


fig 31. Evaluation of Feature Importance

Based on the image above, it mainly demonstrated that the creating features are related with other basic features, of which some are more essential than the others, such as confidence rate, it has an important effect on is_attributed.

3 Model Selection and Algorithms

As mentioned above, this project will apply five machine learning models – Random Forest, Gradient Boosting & AdaBoost, Logistic Regression, SVM, Ensemble Learning and the combination of the five models, one deep learning algorithm, ANN. This part will illustrate these algorithms and realization one by one.

It is essential to find out the relations of all features with target variable(is_attributed). The correlation plot shows that the target variable has a strong relationship with basic features, such as app, channel, os, etc. In addition, it has more important relationship with features we created, such as confidence rate. It is indicated that we should pay more attention to these features.

3.1 Data Preparation for Model

3.1.1 Data Split for Model

In data mining, source dataset is split to two subsets for different use. The training set is used to build the predictive models, the testing set is unseen data when building models, which is used to evaluate the models' performance.

This project will split training set and testing set with different rows. The training set will have 200,000 rows to build the models, the other will have 50,000 rows to compare with the output of training set.

3.2 Machine Learning

3.2.1 Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. [2]

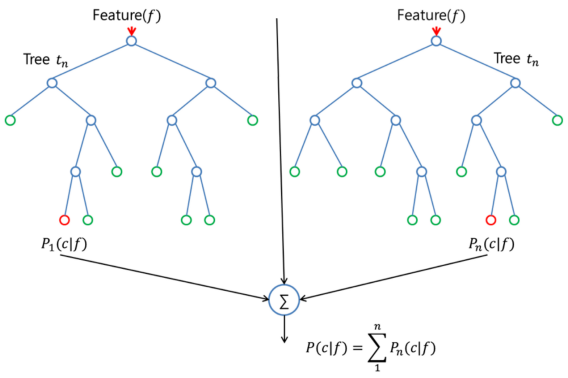


fig 32. Workflow of random forest

Advantages of using Random Forest:

- The process of averaging or combining the results of different decision trees helps to overcome the problem of overfitting.
- Having less variance than a single decision tree.
- Extremely flexible and have very high accuracy.
- Do not require preparation of the input data.
- It also maintains accuracy even when a large proportion of the data are missing.

Disadvantage:

- Complexity. They are much harder and time-consuming to construct than decision trees.
- Require more computational resources and are also less intuitive.
- The prediction process is time-consuming than other algorithms. [3]

This project will use RandomForestRegressor, the meaning of parameters:

- n_estimators: The number of trees in the forest.
- max_depth: The maximum depth of the tree. If none, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- min_samples_split: The minimum number of samples required of split an internal node.
- min_samples_leaf: The minimum number of samples required to be at a leaf node.
- n_jobs: The number of jobs to run in parallel for bot fit and predict. If _1, then the number of jobs is set to the number of cores.

3.2.2 Logistics Regression

Logistic regression, or logit regression, or is a regression model where the outcome variable is categorical. Often this is used when the variable is binary (e.g. yes/no, survived/dead, pass/fail, etc.)

Logistic regression measures the relationship between the categorical response variable and one or more predictor variables by estimating probabilities. [4]

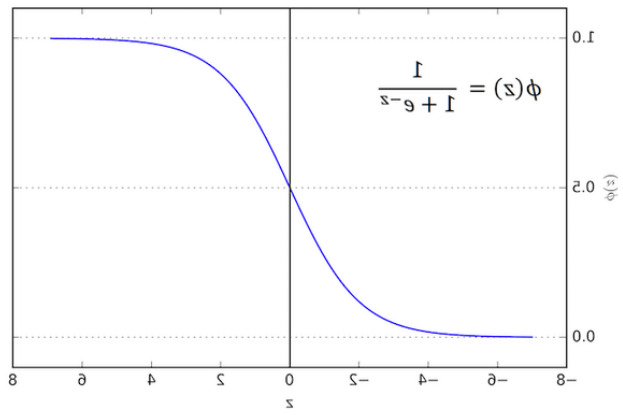


fig 33. Logistics Regression

It also has both strengths and weaknesses when applying this technology:

- **Strengths:** Outputs have a nice probabilistic interpretation, and the algorithm can be regularized to avoid overfitting. Logistic models can be updated easily with new data using stochastic gradient descent.
- **Weaknesses:** Logistic regression tends to underperform when there are multiple or non-linear decision boundaries. They are not flexible enough to naturally capture more complex relationships. [3]

3.2.3 Support Vector Machines

A support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Given a set of data points that belong to either of two classes, an SVM finds the hyperplane that:

- Leaves the largest possible fraction of points of the same class on the same side.
- Maximizes the distance of either class from the hyperplane.

- Find the optimal separating hyperplane that minimizes the risk of misclassifying the training samples and unseen test samples. [4]

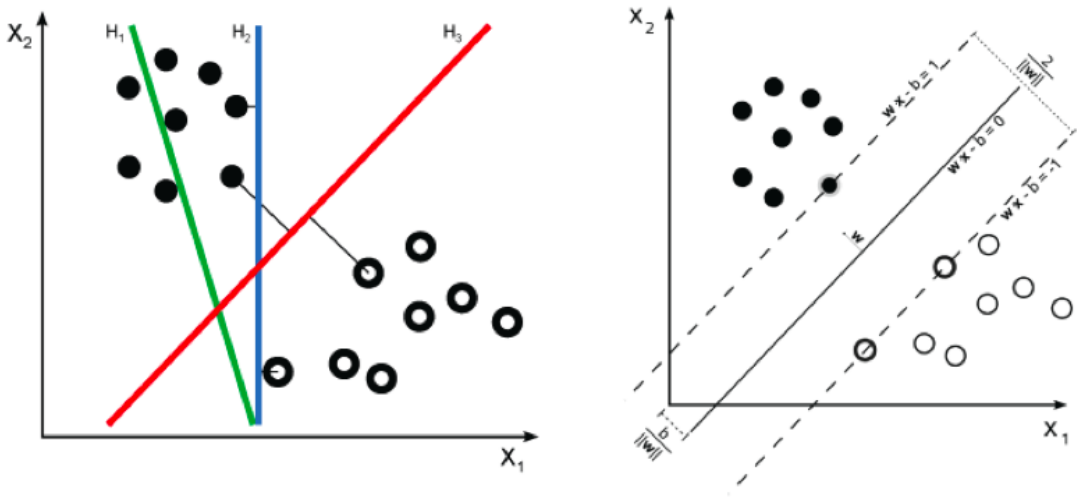


fig 34. Support Vector Machine

It also has both strengths and weaknesses when applying this technology:

- **Strengths:** SVM's can model non-linear decision boundaries, and there are many kernels to choose from. They are also fairly robust against overfitting, especially in high-dimensional space.
- **Weaknesses:** However, SVM's are memory intensive, trickier to tune due to the importance of picking the right kernel, and don't scale well to larger datasets. Currently in the industry, random forests are usually preferred over SVM's. [5]

3.2.4 Gradient Boosting & AdaBoost

While the functional mathematics of the AdaBoost technique are quite intimidating, the philosophy is quite simple. First select a base classifier which makes predictions on the given set. Note down the misclassified instances. The weights of the misclassified instances are increased. A second classifier is

trained on the training set with updated weights.

In simple terms, run a Classifier and make predictions. Run another classifier to fit the previously misclassified instances and make predictions. Repeat until all/most of the training instances are fitted.

Instead of a Decision Tree, AdaBoost uses a *Decision Stump* which is a decision tree with max_depth = 1, i.e., Tree of single decision node and two leaf nodes. The n_estimatorsparameters in AdaBoost sets the number of Decision Stumps.

Similar to AdaBoost, Gradient Boosting also works with successive predictive models added to the ensemble. Instead of updating the weights of the training instances like AdaBoost, Gradient Boosting fits the new model to the residual errors.

Put simply, Fit a model to the given Training set. Calculate the Residual Errors which become the new training instances. A new model is trained on these and so on. An addition of all the models is selected for making predictions. [6]

3.2.5 Realization of Four Models

Random Forest, Gradient Boosting & AdaBoost, Support Vector Machine, Logistics Regression and Ensemble Learning all belong to machine learning. Here using Scikit-learn to build machine learning models. Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

- **NumPy**: Base n-dimensional array package
- **SciPy**: Fundamental library for scientific computing
- **Matplotlib**: Comprehensive 2D/3D plotting
- **IPython**: Enhanced interactive console

- **Sympy**: Symbolic mathematics
- **Pandas**: Data structures and analysis

The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data. For these features, refer to NumPy and Pandas. [29]

All objects within scikit-learn share a uniform common basic API consisting of three complementary interfaces: an estimator interface for building and fitting models, a predictor interface for making predictions and a transformer interface for converting data.

The estimator interface is at the core of the library. It defines instantiation mechanisms of objects and exposes a fit method for learning a model from training data. All supervised and unsupervised learning algorithms (e.g., for classification, regression or clustering) are offered as objects implementing this interface. Machine learning tasks like feature extraction, feature selection or dimensionality reduction are also provided as estimators.

Actual learning is performed by the fit method. This method is called with training data (e.g., supplied as two arrays X train and y train in supervised learning estimators). Its task is to run a learning algorithm and to determine model-specific parameters from the training data and set these as attributes on the estimator object. As a convention, the parameters learned by an estimator are exposed as public attributes with names suffixed with a trailing underscore (e.g., coef for the learned coefficients of a linear model), again to facilitate model inspection. In the partial application view, fit is a function from data to a model of that data. It always returns the estimator object it was called on, which now serves as a model of its input and can be used to perform predictions or transformations of input data. [30]

In this project, the steps for the models as follows:

1. Import packages from library
2. Define the function of prediction using models included in the library
3. Train the model
4. Get the prediction of the test data
5. Compare the predictions with the observed values
6. Plot data

```
def randomForest_pre(X_train, y_train, X_test, y_test):
    X_train = np.nan_to_num(X_train)
    y_train = np.nan_to_num(y_train)
    X_test = np.nan_to_num(X_test)
    y_test = np.nan_to_num(y_test)
    RF_model = RandomForestClassifier()
    # Train the model
    RF_fit = RF_model.fit(X_train, y_train)
    # Get the prediction of the test data
    RF_predict = RF_model.predict(X_test)
    # Compare the prediction with the known values
    RF_acc = sklearn.metrics.accuracy_score(np.array(RF_predict)[:],
                                            np.array(y_test_val)[:])
    # Plot the data
    plt.figure(figsize=(10,5))
    plt.plot(RF_predict, color='red', label='Prediction')
    plt.plot(y_test, label='Y_test')
    plt.legend(['Prediction', 'Y_test'])
    _ = plt.ylim()

    return RF_fit, RF_predict, RF_acc, RF_model
```

fig 35. Random Forest Function

```
def logisticRegression_pre(X_train, y_train, X_test, y_test):
    X_train = np.nan_to_num(X_train)
    y_train = np.nan_to_num(y_train)
    X_test = np.nan_to_num(X_test)
    y_test = np.nan_to_num(y_test)
    LG_model = LogisticRegression()
    # Train the model
    LG_fit = LG_model.fit(X_train, y_train)
    # Get the prediction of the test data
    LG_predict = LG_model.predict(X_test)
    # Compare the prediction with the known values
    LG_acc = sklearn.metrics.accuracy_score(np.array(LG_predict)[:],
                                            np.array(y_test_val)[:])

    return LG_fit, LG_predict, LG_acc, LG_model
```

fig 36. Logistic Regression Function

```
def svm_pre(X_train, y_train, X_test, y_test):
    X_train = np.nan_to_num(X_train)
    y_train = np.nan_to_num(y_train)
    X_test = np.nan_to_num(X_test)
    y_test = np.nan_to_num(y_test)
    svm_model = svm.SVC()
    uniq = np.unique(y_train[9000:10000])
    # Train the model
    SVM_fit = svm_model.fit(X_train[9000:10000], y_train[9000:10000])
    # Get the prediction of the test data
    SVM_predict = svm_model.predict(X_test)
    # Compare the prediction with the known values
    SVM_acc = sklearn.metrics.accuracy_score(np.array(LG_predict)[:],
                                             np.array(y_test_val)[:])
    return SVM_fit, SVM_predict, SVM_acc, svm_model
```

fig 37. Support Vector Machine Function

```
def gradientBoosting_pre(X_train, y_train, X_test, y_test):
    X_train = np.nan_to_num(X_train)
    y_train = np.nan_to_num(y_train)
    X_test = np.nan_to_num(X_test)
    y_test = np.nan_to_num(y_test)
    GB_model = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0)
    # Train the model
    GB_fit = GB_model.fit(X_train, y_train)
    # Get the prediction of the test data
    GB_predict = GB_model.predict(X_test)
    # Compare the prediction with the known values
    GB_acc = sklearn.metrics.accuracy_score(np.array(GB_predict)[:],
                                            np.array(y_test_val)[:])

    # Plot the data
    plt.figure(figsize=(10,5))
    plt.plot(GB_predict, color='red', label='Prediction')
    plt.plot(y_test, label='Y_test')
    plt.legend(['Prediction', 'Y_test'])
    _ = plt.ylim()

    return GB_fit, GB_predict, GB_acc, GB_model
```

fig 38. Gradient Boosting Function

In this project, we will build four new functions with RF, SVM, GB and LG machine learning models to predict the accuracy of is_attributed, as the images shown above.

3.3 Deep Learning

3.3.1 ANN

Artificial neural networks (ANNs) or [connectionist] systems are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve performance) to do tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as “cat” or “no cat” and using the analytic results to identify cats in other images. They have found most use in applications difficult to express in a traditional computer algorithm using rule-based programming. [27]

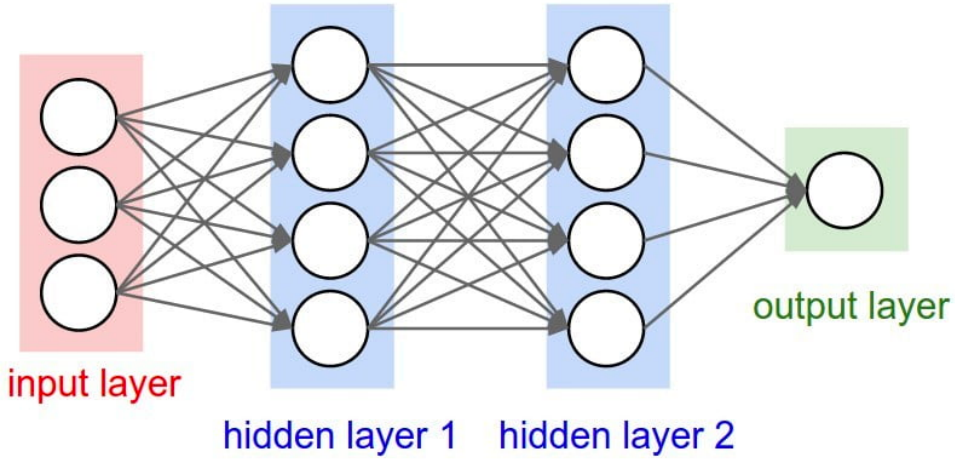


fig 39. Layers of ANN

Advantages of Artificial Neural Networks (ANN)

► **Storing information on the entire network**: Information such as in traditional **programming** is stored on the entire network, not on a database. The disappearance of a few pieces of information in one place does not prevent the network from functioning.

► **Ability to work with incomplete knowledge:** After ANN training, the data may produce output even with incomplete information. The loss of performance here depends on the importance of the missing information.

► **Having fault tolerance:** Corruption of one or more cells of ANN does not prevent it from generating output. This feature makes the networks fault tolerant.

► **Having a distributed memory:** In order for ANN to be able to learn, it is necessary to determine the examples and to teach the network according to the desired output by showing these examples to the network. The network's success is directly proportional to the selected instances, and if the event cannot be shown to the network in all its aspects, the network can produce false output

► **Gradual corruption:** A network slows over time and undergoes relative degradation. The network problem does not immediately corrode immediately.

► **Ability to make machine learning:** Artificial neural networks learn events and make decisions by commenting on similar events.

► **Parallel processing capability:** Artificial neural networks have numerical strength that can perform more than one job at the same time.

Disadvantages of Artificial Neural Networks (ANN)

► **Hardware dependence:** Artificial neural networks require processors with parallel processing power, in accordance with their

structure. For this reason, the realization of the equipment is dependent.

► **Unexplained behavior of the network:** This is the most important problem of ANN. When ANN produces a probing solution, it does not give a clue as to why and how. This reduces trust in the network.

► **Determination of proper network structure:** There is no specific rule for determining the structure of artificial neural networks. Appropriate network structure is achieved through experience and trial and error.

► **Difficulty of showing the problem to the network:** ANNs can work with numerical information. Problems have to be translated into numerical values before being introduced to ANN. The display mechanism to be determined here will directly influence the performance of the network. This depends on the user's ability.

► **The duration of the network is unknown:** The network is reduced to a certain value of the error on the sample means that the training has been completed. This value does not give us optimum results. [28]

It is crucial to determine controllable factors that would have the most important effects on the performance of ANN. These factors are described as follows:

The percentage of training data: The data are divided into three subsets: training, validation and testing set. The training set is used for computing the gradient and updating the network weights and biases. As the network begins to over-fit the data, the error on the validation set typically begins to rise. After the specified number of iterations (which can be defined by the researcher) and the validation error increases, the training is stopped, and the weights and biases of the epoch, with minimum validation errors are returned as the final ANN structure.

In this study, the percentage of validation data is considered 5% of the data which is assumed to be a constant value. In other words, for example, if the parameter level is taken as 0.7, it means that 70% of the data are used for training the network, while 25% and 5% of the data are used for testing and validating, respectively.

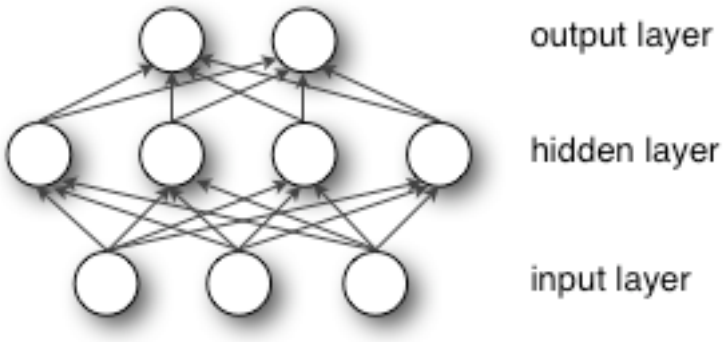
The number of neurons in the first layer: This factor is one of the most effective parameters in performance of ANN. Although more neurons require more computation, their implementation might result more efficiently for solving complex problems.

The number of neurons in the second layer: This factor determines the number of neurons in the second layer, and moreover, helps us to determine whether the network is one layered or not. In other words, for example, if the parameter level is taken as zero, it means that the network has only one hidden layer. Although more layers require more computation, their implementation might result more efficiently for solving complex problems. [31]

3.3.2 MLP

An MLP can be viewed as a logistic regression classifier where the input is first transformed using a learnt non-linear transformation Φ. This transformation projects the input data into a space where it becomes linearly separable. This intermediate layer is referred to as a hidden layer. A single hidden layer is sufficient to make MLPs a universal approximator.

An MLP (or Artificial Neural Network - ANN) with a single hidden layer can be represented graphically as follows:



Formally, a one-hidden-layer MLP is a function $f : R^D \rightarrow R^L$, where D is the size of input vector x and L is the size of the output vector $f(x)$, such that, in matrix notation:

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} +$$

with bias vectors $b^{(1)}$, $b^{(2)}$; weight matrices $W^{(1)}$, $W^{(2)}$ and activation functions G and s.

The vector

$$h(x) = \Phi(x) = s(b^{(1)} + W^{(1)}x)$$

constitutes the hidden layer. $W^{(1)} \in R^{D \times D_h}$ is the weight matrix connecting the input vector to the hidden layer. Each column $W^{(1)}_{\cdot i}$ represents the weights from the input units to the i-th hidden unit. Typical choices for s include $tanh$, with

$$tanh(a) = (e^a - e^{-a})/(e^a + e^{-a})$$

, or the logistic $sigmoid$ function, with

$$sigmoid(a) = 1/(1 + e^{-a})$$

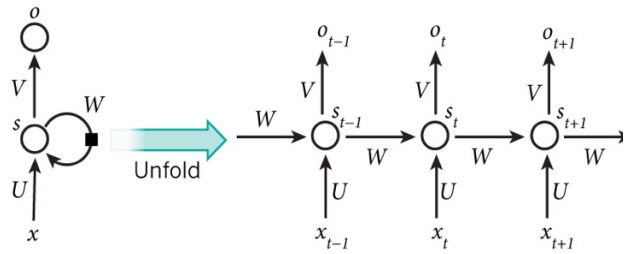
.

We will be using $tanh$ in this tutorial because it typically yields to faster training (and sometimes also to better local minima). Both the $tanh$ and $sigmoid$ are scalar-to-scalar functions but their natural extension to vectors and tensors consists in applying them element-wise (e.g. separately on each element of the vector, yielding a same-size vector). [32]

3.3.3 RNN

The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before

it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps (more on this later). Here is what a typical RNN looks like:



The above diagram shows a RNN being unrolled (or unfolded) into a full network. By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word. The formulas that govern the computation happening in a RNN are as follows:

- x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It's the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. The function f usually is a nonlinearity such as tanh or ReLU. s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = \text{softmax}(Vs_t)$.

RNNs have shown great success in many NLP tasks. At this point I should mention that

the most commonly used type of RNNs are LSTMs, which are much better at capturing long-term dependencies than vanilla RNNs are. But don't worry, LSTMs are essentially the same thing as the RNN we will develop in this tutorial, they just have a different way of computing the hidden state. Here are some example applications that RNN can apply [33]:

- Language Modeling and Generating Text
- Machine Translation
- Speech Recognition
- Generating Image Descriptions

3.3.4 Realization of Three Models

In this part, we will use Keras to build models, since Keras is a model-level library, providing high-level building blocks for developing deep learning models. It has the advantages:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

In addition, TensorFlow will be used in this part as well (TensorFlow is an open-source symbolic tensor manipulation framework developed by Google. By default, keras will use TensorFlow as its tensor manipulation library).

In this project, the steps for the models as follows:

1. Scaling data
2. Split training dataset to train and validate date
3. Import packages from library
4. Define the function of prediction using models included in the library
5. Convert matrix to get two classes

6. Train the model
7. Evaluate loss & accuracy
8. Get the prediction of the test data
9. Convert value to boolean
10. Compare the predictions with the observed values

```
def ann_pre(X_train, y_train, X_test, y_test):
    ann_model = shallow_net_A()
    ann_summary = ann_model.summary()
    # Convert the values
    X_train_ann = np.nan_to_num(X_train)
    y_train_ann = np.nan_to_num(y_train)
    X_test_ann = np.nan_to_num(X_test)
    y_test_ann = np.nan_to_num(y_test)
    # Conver the matrix, finally we have two classes (n_classes), the original one has oly one class
    n_classes = 2
    y_train_ann = keras.utils.to_categorical(y_train_ann, n_classes)
    y_test_ann = keras.utils.to_categorical(y_test_ann, n_classes)
    # Training the model
    ann_fit = ann_model.fit(X_train_ann, y_train_ann, batch_size=128, epochs=99, verbose=1, validation_data=(X_te

    # Evaluate: loss & accuracy -> Using Evaluation to get the accracy
    ann_evaluate = ann_model.evaluate(X_test_ann, y_test_ann)

    # Using prediction
    ann_pre = ann_model.predict(X_test)
    # Convert value to boolean value
    y_pre = (ann_pre > 0.5)
    # Counting the boolean value, counting the accuracy by using basic calculation
    from sklearn.metrics import confusion_matrix
    ann_output = confusion_matrix(y_test_ann.argmax(axis=1), y_pre.argmax(axis=1))
    ann_prediction_acc = ann_output[0][0]/(ann_output[0][0]+ann_output[1][0])

    return ann_summary, ann_fit, ann_evaluate, ann_prediction_acc, ann_model
```

fig 40. ANN Model Function

```
def mlp_pre(X_train, y_train, X_test, y_test):
    mlp_model = shallow_net_C()
    mlp_summary = mlp_model.summary()
    # Convert the values
    X_train_mlp = np.nan_to_num(X_train)
    y_train_mlp = np.nan_to_num(y_train)
    X_test_mlp = np.nan_to_num(X_test)
    y_test_mlp = np.nan_to_num(y_test)
    # Conver the matrix, finally we have two classes (n_classes)
    n_classes = 2
    y_train_mlp = keras.utils.to_categorical(y_train_mlp, n_classes)
    y_test_mlp = keras.utils.to_categorical(y_test_mlp, n_classes)
    # Training the model
    mlp_fit = mlp_model.fit(X_train_mlp, y_train_mlp, batch_size=128, epochs=99, verbose=1, validation_data=(X_test_

    # Evaluate: loss & accuracy -> Using Evaluation to get the accracy
    mlp_evaluate = mlp_model.evaluate(X_test_mlp, y_test_mlp)

    # Using prediction
    mlp_pre = mlp_model.predict(X_test)
    # Convert value to boolean value
    y_pre = (mlp_pre > 0.5)
    # Counting the boolean value, counting the accuracy by using basic calculation
    from sklearn.metrics import confusion_matrix
    mlp_output = confusion_matrix(y_test_mlp.argmax(axis=1), y_pre.argmax(axis=1))
    mlp_prediction_acc = mlp_output[0][0]/(mlp_output[0][0]+mlp_output[1][0])

    return mlp_summary, mlp_fit, mlp_evaluate, mlp_prediction_acc, mlp_model
```

fig 41. MLP Model Function

```
def rnn_pre(df_train, train_rate=0.75):
    # Set the dataset for train and test
    df_rnn_train = df_train.loc[:,train_cols + ['is_attributed']]
    df_rnn_test = df_test.loc[:,train_cols + ['is_attributed']]
    df_rnn = df_rnn_train.append(df_rnn_test)

    pre_col_index = list(df_rnn_train).index('is_attributed')
    dataset = df_rnn.values.astype('float32')

    # Normalize the dataset, set all the data of the dataset to be in the range between 0 and 1
    scaler = MinMaxScaler(feature_range=(0, 1))
    dataset = scaler.fit_transform(dataset)

    # Split into train and test sets
    train_size = int(len(dataset) * train_rate)
    test_size = len(dataset) - train_size
    train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]

    # Use this function to prepare the train and test datasets for modeling
    look_back = 1
    trainY = train[:, pre_col_index]
    trainX = np.delete(train, pre_col_index, axis = 1)
    testY = test[:, pre_col_index]
    testX = np.delete(test, pre_col_index, axis = 1)

    # Reshape input to be [samples, time steps, features], here it changes the dimension from 2D to 3D
    trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
    testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

    # Create and fit the LSTM network
    RNN_model = Sequential()
    RNN_model.add(LSTM(5, input_shape=(1, len(trainX[0][0]))))
    RNN_model.add(Dense(1))
    RNN_model.compile(loss='mean_squared_error', optimizer='adam')
    RNN_model.fit(trainX, trainY, epochs=10, batch_size=128, verbose=2)

    # Make predictions, trainPredict should be 1D array
    trainPredict = RNN_model.predict(trainX)
    testPredict = RNN_model.predict(testX)
```

```
  # Change the dimension from 3D to 2D
  trainX_2D = trainX.transpose([1,0,2]).reshape(len(trainX),len(trainX[0][0]))
  testX_2D = testX.transpose([1,0,2]).reshape(len(testX),len(testX[0][0]))

  # Append prediction back to the model
  trainPredict_6cols = np.append(trainX_2D, trainPredict, 1)
  testPredict_6cols = np.append(testX_2D, testPredict, 1)

  # Invert predictions back to normal values
  trainPredict_6cols = scaler.inverse_transform(trainPredict_6cols)
  testPredict_6cols = scaler.inverse_transform(testPredict_6cols)

  # Calculating the RMSE
  trainScore = math.sqrt(mean_squared_error(trainY, trainPredict_6cols[:, pre_col_index]))
  print('Train Score: %.2f RMSE' % (trainScore))
  testScore = math.sqrt(mean_squared_error(testY, testPredict_6cols[:, pre_col_index]))
  print('Test Score: %.2f RMSE' % (testScore))

  final_prediction_train = np.where(trainPredict_6cols[:, pre_col_index] > 0, 1, 0)
  final_prediction_test = np.where(testPredict_6cols[:, pre_col_index] > 0, 1, 0)

  # Change dimension from 2D to 1D
  final_prediction_train = np.reshape(final_prediction_train, (-1, 1))
  final_prediction_test = np.reshape(final_prediction_test, (-1, 1))

  # Counting the accuracy by using basic calculation
  rnn_acc_train = sklearn.metrics.accuracy_score(np.array(final_prediction_train)[:],
                                  np.array(trainY)[:])
  rnn_acc_test = sklearn.metrics.accuracy_score(np.array(final_prediction_test)[:],
                                  np.array(testY)[:])
  return rnn_acc_train
```

fig 42. RNN Model Function

In this project, we will build three new functions with ANN, MLP and RNN deep learning models to predict the accuracy of is_attributed, as the images shown above.

As mentioned above, we've built random forest model that defines a new Random Forest function using random forest model, then training the model, getting the prediction of the testing data, comparing it with the observed values to get the accuracy.

4 Conclusion

4.1 Random Forest

4.1.1 Comparison Between the Two Outputs

1. Random Forest

```
RF_fit, RF_pre, RF_acc, rf_model = randomForest_pre(X_train, y_train, X_test, y_test)
print('Random Forest accuracy: {}%'.format(RF_acc * 100))
```

```
Random Forest accuracy: 99.842%
```

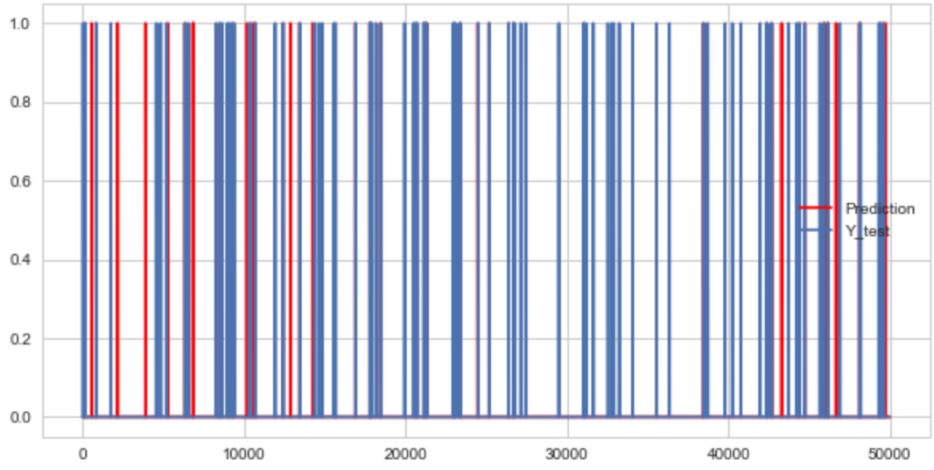


fig 43. Output for Random Forest Function for Original Data

1. Random Forest

```
RF_fit, RF_pre, RF_acc, rf_model = randomForest_pre(X_train, y_train, X_test, y_test)
print('Random Forest accuracy: {}%'.format(RF_acc * 100))
```

```
Random Forest accuracy: 99.92%
```

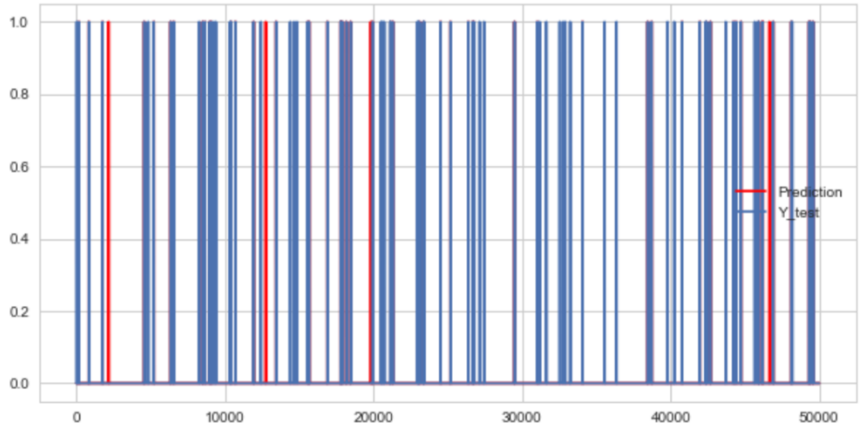


fig 44. Output for Random Forest Function for Creating Feature Data

We have conclusion that the new features are more essential to predict the accuracy, because it generated 99.92% of accuracy, which is much better than the original one (99.842%).

As mentioned above, we've built gradient boosting model that defines a new Gradient Boosting function, then training the model, getting the prediction of the testing data, comparing it with the observed values to get the accuracy.

4.2 Gradient Boosting

```
GB_fit, GB_predict, GB_acc, gb_model = gradientBoosting_pre(X_train, y_train, X_test, y_test)
print('Gradient Boosting accuracy: {}%'.format(GB_acc * 100))
```

```
Gradient Boosting accuracy: 99.57000000000001%
```

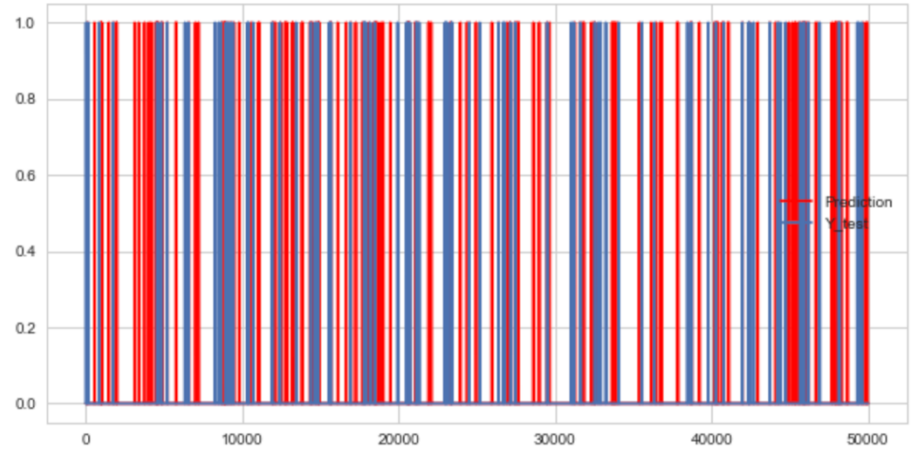


fig 45. Output for Gradient Boosting of Original Data

```
GB_fit, GB_predict, GB_acc, gb_model = gradientBoosting_pre(X_train, y_train, X_test, y_test)
print('Gradient Boosting accuracy: {}%'.format(GB_acc * 100))
```

```
Gradient Boosting accuracy: 99.634%
```

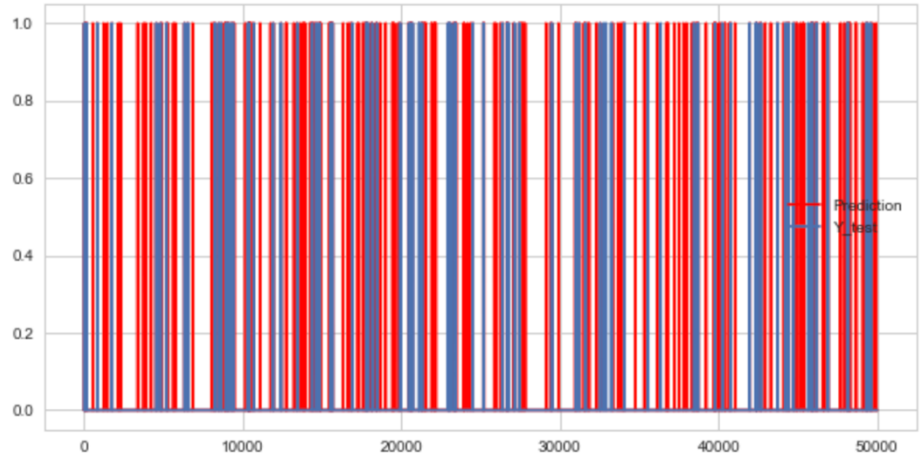


fig 46. Output for Gradient Boosting of Creating Features Data

4.3 Support Vector Machine

As mentioned above, we've built SVM model that defines a new SVM function, then training the model, getting the prediction of

the testing data, comparing it with the observed values to get the accuracy.

```
svm_fit, svm_predict, svm_acc, svm_model = svm_pre(X_train, y_train, X_test, y_test)
print('SVM accuracy: {}%'.format(svm_acc * 100))
```

```
SVM accuracy: 99.824%
```

fig 47. Output for Support Vector Machine of Original Data

```
svm_fit, svm_predict, svm_acc, svm_model = svm_pre(X_train, y_train, X_test, y_test)
print('SVM accuracy: {}%'.format(svm_acc * 100))
```

```
SVM accuracy: 99.824%
```

fig 48. Output for Support Vector Machine of Creating Features Data

We have conclusion that the new features are as important as the original one to predict the accuracy, because it generated 99.824% of accuracy, which is the same as the previous one.

4.4 Logistic Regression

As mentioned above, we've built logistic regression model that defines a new LG function, then training the model, getting the prediction of the testing data, comparing it with the observed values to get the accuracy.

```
LG_fit, LG_predict, LG_acc, lg_model = logisticRegression_pre(X_train, y_train, X_test, y_test)
print('Logistic Regression accuracy: {}%'.format(LG_acc * 100))
```

```
Logistic Regression accuracy: 99.824%
```

fig 49. Output for Logistic Regression of Original Data

```
LG_fit, LG_predict, LG_acc, lg_model = logisticRegression_pre(X_train, y_train, X_test, y_test)
print('Logistic Regression accuracy: {}%'.format(LG_acc * 100))
```

```
Logistic Regression accuracy: 99.824%
```

fig 50. Output for Logistic Regression of Creating Features Data

We have conclusion that the new features are as important as the original one to predict the accuracy, because it generated 99.824% of accuracy, which is the same as the previous one.

4.5 ANN

As mentioned above, we've built ANN model that defines a new ANN function, then training the model, getting the prediction of the testing data, comparing it with the observed values to get the accuracy.

```
ann_summary, ann_fit, ann_evaluate, ann_prediction_acc, ann_model = ann_pre(X_train, df_train, X_test, df_test)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 55)                330
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 112
=================================================================
Total params: 442
Trainable params: 442
Non-trainable params: 0
_________________________________________________________________
Train on 200000 samples, validate on 50000 samples
Epoch 1/99
200000/200000 [==============================] - 2s 12us/step - loss: 0.0035 - acc: 0.9983 - val_loss: 0.0020 - val_a
cc: 0.9982
Epoch 2/99
200000/200000 [==============================] - 2s 11us/step - loss: 0.0019 - acc: 0.9983 - val_loss: 0.0019 - val_a
cc: 0.9982
Epoch 3/99
```

```
print('ANN accuracy: {}%'.format(ann_prediction_acc * 100))
```

```
ANN accuracy: 99.824%
```

fig 51. Output for ANN of Original Data

```
ann_summary, ann_fit, ann_evaluate, ann_prediction_acc, ann_model = ann_pre(X_train, y_train, X_test, y_test)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 55)                2970
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 112
=================================================================
Total params: 3,082
Trainable params: 3,082
Non-trainable params: 0
_________________________________________________________________
Train on 200000 samples, validate on 50000 samples
Epoch 1/99
200000/200000 [==============================] - 3s 15us/step - loss: 0.0034 - acc: 0.9982 - val_loss: 0.0020 - val_a
cc: 0.9982
Epoch 2/99
200000/200000 [==============================] - 3s 14us/step - loss: 0.0019 - acc: 0.9983 - val_loss: 0.0018 - val_a
cc: 0.9982
Epoch 3/99
```

```
print('ANN accuracy: {}%'.format(ann_prediction_acc * 100))
```

```
ANN accuracy: 99.824%
```

fig 52. Output for ANN of Creating Features Data

Based on the images above, we have conclusion that the new features are as important as the original one to predict the accuracy, because it generated 99.824% of

accuracy, which is the same as the previous one.

4.6 MLP

```
mlp_summary, mlp_fit, mlp_evaluate, mlp_prediction_acc, mlp_model = mlp_pre(X_train, df_train, X_test, df_test)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 55)                330
_________________________________________________________________
dense_11 (Dense)             (None, 55)                3080
_________________________________________________________________
dense_12 (Dense)             (None, 55)                3080
_________________________________________________________________
dense_13 (Dense)             (None, 55)                3080
_________________________________________________________________
dense_14 (Dense)             (None, 2)                 112
=================================================================
Total params: 9,682
Trainable params: 9,682
Non-trainable params: 0
_________________________________________________________________
Train on 200000 samples, validate on 50000 samples
Epoch 1/99
```

```
print('MLP accuracy: {}%'.format(mlp_prediction_acc * 100))
```

```
MLP accuracy: 99.824%
```

fig 53. Output for MLP of Original Data

```
mlp_summary, mlp_fit, mlp_evaluate, mlp_prediction_acc, mlp_model = mlp_pre(X_train, y_train, X_test, y_test)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 55)                2970
_________________________________________________________________
dense_4 (Dense)              (None, 55)                3080
_________________________________________________________________
dense_5 (Dense)              (None, 55)                3080
_________________________________________________________________
dense_6 (Dense)              (None, 55)                3080
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 112
=================================================================
Total params: 12,322
Trainable params: 12,322
Non-trainable params: 0
_________________________________________________________________
Train on 200000 samples, validate on 50000 samples
Epoch 1/99
```

```
print('MLP accuracy: {}%'.format(mlp_prediction_acc * 100))
```

```
MLP accuracy: 99.824%
```

fig 54. Output for MLP of Creating Features Data

As mentioned above, we've built MLP model that defines a new MLP function, then training the model, getting the prediction of the testing data, comparing it with the observed values to get the accuracy.

Based on the images above, we have conclusion that the new features are as important as the original one to predict the accuracy, because it generated 99.824% of accuracy, which is the same as the previous one.

4.7 RNN

As mentioned above, we've built MLP model that defines a new MLP function, then training the model, getting the prediction of the testing data, comparing it with the observed values to get the accuracy.

```
rnn_acc = rnn_pre(df_train)
print('RNN accuracy: {}%'.format(rnn_acc * 100))
```

```
Epoch 1/10
 - 4s - loss: 0.0018
Epoch 2/10
 - 5s - loss: 0.0017
Epoch 3/10
 - 4s - loss: 0.0017
Epoch 4/10
 - 5s - loss: 0.0017
Epoch 5/10
 - 4s - loss: 0.0017
Epoch 6/10
 - 6s - loss: 0.0017
Epoch 7/10
 - 5s - loss: 0.0017
Epoch 8/10
 - 5s - loss: 0.0017
Epoch 9/10
 - 5s - loss: 0.0017
Epoch 10/10
 - 4s - loss: 0.0017
Train Score: 0.04 RMSE
Test Score: 0.04 RMSE
RNN accuracy: 14.674133333333334%
```

fig 55. Output for RNN of Original Features Data

```
rnn_acc = rnn_pre(df_train)
print('RNN accuracy: {}%'.format(rnn_acc * 100))
```

```
Epoch 1/10
 - 6s - loss: 0.0014
Epoch 2/10
 - 5s - loss: 0.0012
Epoch 3/10
 - 5s - loss: 0.0012
Epoch 4/10
 - 5s - loss: 0.0012
Epoch 5/10
 - 5s - loss: 0.0012
Epoch 6/10
 - 5s - loss: 0.0012
Epoch 7/10
 - 5s - loss: 0.0012
Epoch 8/10
 - 5s - loss: 0.0012
Epoch 9/10
 - 5s - loss: 0.0012
Epoch 10/10
 - 4s - loss: 0.0012
Train Score: 0.03 RMSE
Test Score: 0.03 RMSE
RNN accuracy: 92.27680000000001%
```

fig 56. Output for RNN of Creating Features Data

We have conclusion that the new features are more essential to predict the accuracy based on the images above, because it generated 99.277% of accuracy, which is much better than the original one (14.674%).

4.8 Comparison Between All Features & Original

4.8.1 Comparison Between Extracting Time Information & Original

```
df_predict = rf_model.predict(df)
# Compare the prediction with the known values
df_acc = sklearn.metrics.accuracy_score(np.array(df_predict)[:],
                                        np.array(sample_out)[:])
```

```
print('By using the best algorittm, the accuracy of the prediction: {}%'.format(df_acc * 100))
```

```
By using the best algorittm, the accuracy of the prediction: 99.9546%
```

fig 57. Prediction Accuracy for Best Algorithm

```
Random Forest accuracy: 99.832%
Gradient Boosting accuracy: 99.57000000000001%
Logistic Regression accuracy: 99.824%
SVM accuracy: 99.824%
ANN accuracy: 99.824%
MLP accuracy: 99.824%
RNN accuracy: 19.5808%
```

fig 58. Prediction Accuracy for All Algorithms of Extracting Time

It is obvious that it will improve accuracy for prediction by extracting time. We have improved best accuracy from 99.842% to 99.9546%, which is a huge improvement for the prediction.

4.8.2 Comparison Between Creating Confidence Rate for is_attributed & Original

```
df_predict = rf_model.predict(df)
# Compare the prediction with the known values
df_acc = sklearn.metrics.accuracy_score(np.array(df_predict)[:],
                                        np.array(sample_out)[:])
```

```
print('By using the best algorittm, the accuracy of the prediction: {}%'.format(df_acc * 100))
```

```
By using the best algorittm, the accuracy of the prediction: 100.0%
```

fig 59. Prediction Accuracy for Best Algorithm of Confidence Rate

```
Random Forest accuracy: 99.864%
Gradient Boosting accuracy: 99.634%
Logistic Regression accuracy: 99.824%
SVM accuracy: 99.824%
ANN accuracy: 99.824%
MLP accuracy: 99.824%
RNN accuracy: 41.820266666666667%
```

fig 60. Prediction Accuracy for All Algorithms of Confidence Rate

We have improved the prediction accuracy of best algorithm from 99.864% to 100%, based on the images shown above. The 100% accuracy is due to the fact that the data we tested was too long because the computer only tested 100W rows of data, which is not big enough, so the accuracy of the prediction was biased. However, it is clear that adding new feature like Confidence Rate for is_attributed is much more helpful when predicting the accuracy.

4.8.3 Comparison Between Group-By-Aggregation & Original

```
# Read the output of the test data
sample_out = pd.read_csv('data/sample_submission.csv', nrows=1000000)[['is_attributed']].astype('float64')
sample_out = np.nan_to_num(sample_out)
```

```
df_predict = rf_model.predict(df)
# Compare the prediction with the known values
df_acc = sklearn.metrics.accuracy_score(np.array(df_predict)[:],
                                        np.array(sample_out)[:])
```

```
print('By using the best algorittm, the accuracy of the prediction: {}%'.format(df_acc * 100))
```

```
By using the best algorittm, the accuracy of the prediction: 99.99090000000001%
```

fig 61. Prediction Accuracy for Best Algorithm of Group-By-Aggregation

```
Random Forest accuracy: 99.9%
Gradient Boosting accuracy: 99.816%
Logistic Regression accuracy: 99.824%
SVM accuracy: 99.824%
ANN accuracy: 99.824%
MLP accuracy: 99.824%
RNN accuracy: 44.299733333333336%
```

fig 62. Prediction Accuracy for All Algorithms of Group-By-Aggregation

It is obvious that it will improve accuracy for prediction by Group-by-aggregation. We have improved best accuracy from 99.9% to 99.9909%, which is an improvement for the prediction.

4.8.4 Comparison Between Creating Time till Next Click & Original

```
df_predict = rf_model.predict(df)
# Compare the prediction with the known values
df_acc = sklearn.metrics.accuracy_score(np.array(df_predict)[:],
                                        np.array(sample_out)[:])
```

```
print('By using the best algorittm, the accuracy of the prediction: {}%'.format(df_acc * 100))
```

```
By using the best algorittm, the accuracy of the prediction: 99.9302%
```

fig 63. Prediction Accuracy for Best Algorithm of Time till Next Click

```
Random Forest accuracy: 99.884%
Gradient Boosting accuracy: 99.57000000000001%
Logistic Regression accuracy: 99.824%
SVM accuracy: 99.824%
ANN accuracy: 99.824%
MLP accuracy: 99.824%
RNN accuracy: 23.906133333333333%
```

fig 64. Prediction Accuracy for All Algorithms of Time till Next Click

We have improved the prediction accuracy of best algorithm from 99.884% to 99.9302%, based on the images shown above. It is clear that adding new feature like Confidence Rate for is_attributed is much more helpful when predicting the accuracy.

4.8.5 Comparison Between Creating Click on App Ad Before and After & Original

```
df_predict = rf_model.predict(df)
# Compare the prediction with the known values
df_acc = sklearn.metrics.accuracy_score(np.array(df_predict)[:],
                                        np.array(sample_out)[:])
```

```
print('By using the best algorittm, the accuracy of the prediction: {}%'.format(df_acc * 100))
```

```
By using the best algorittm, the accuracy of the prediction: 99.9305%
```

fig 65. Prediction Accuracy for Best Algorithm of Click on App Ad Before and After

```
Random Forest accuracy: 99.85000000000001%
Gradient Boosting accuracy: 99.57000000000001%
Logistic Regression accuracy: 99.824%
SVM accuracy: 99.824%
ANN accuracy: 99.824%
MLP accuracy: 99.824%
RNN accuracy: 78.70186666666666%
```

fig 66. Prediction Accuracy for All Algorithms of Click on App Ad Before and After

It is obvious that it will improve accuracy for prediction by creating Click on App Ad Before and After this new feature. We have improved best accuracy from 99.85% to 99.9305%, which is a big improvement for the prediction.

4.8.6 All Features

```
Random Forest accuracy: 99.92%
Gradient Boosting accuracy: 99.634%
Logistic Regression accuracy: 99.824%
SVM accuracy: 99.824%
ANN accuracy: 99.824%
MLP accuracy: 99.824%
RNN accuracy: 92.276800000000001%
```

fig 67. Prediction Accuracy for All Algorithms of All Features

By creating new features, we could improve the prediction accuracy based on the original one. For each new feature, it will be more helpful when applying the algorithms, not only for the best one, but also for the others. Therefore, it is obvious that building up models by creating new features to make the prediction will improve the accuracy.

5 Discussion

5.1 Conclusion

It is demonstrated that a feature dataset is a collection of related feature classes that share a common coordinate system. Feature datasets are used to spatially or thematically integrate related feature classes. Their primary purpose is for organizing related feature classes into a common dataset for building a topology, a network dataset, a terrain dataset, or a geometric network. It has advantages as following:

- Organize thematically related feature classes
- Organize data access based on database privileges
- Organize feature classes for data sharing

From the result, we could conclude that:

- It is clear that the features could provide more information for the regression models, allowing them to make better predictions, as is the case of features app, os, channel. It is the

same to the prediction before the project.

- The result of using the original data to predict is different from using the data that adding new features. The most same accuracy of these two ways are Logistics Regression, SVM, ANN, and MLP, which is 99.824%. However, it is obvious that using the data that adding new features will generate more accuracy prediction than using the original one.
- With the implemented solution, it is clear that the best model is Random Forest, because it generated the highest accuracy, which is 99.92% using the data that adding new features.
- There is always a better predictive model because on one is perfect.

5.2 Evaluation & Limitations

The aim of this project it to predict the click ad fraud of TalkingData in China. However, it is impossible to consider all the factors in one research. There are still too many factors will have effect on the users' behavior, for example, a young kid playing the mobile devices or uncertainty of accidents. All of these could affect the result of prediction. Then, as one competitor who joined the competition on Kaggle said, there may be some error in the dataset. Furthermore, the training dataset of TalkingData is not completed as the expected, therefore, it could have effect on the result as well.

5.3 Future work

There are still various additional field is worthy to research in the future, without thinking about the factor of this project. For example, is there any other algorithm that can improve the prediction, is there any virtualization, the best parameters for the model, is there any shortcoming in the model,

is there any way to improve the performance of the model? Moreover, getting more data from TalkingData with different areas will also give a better prediction.

References:

[1] https://en.wikipedia.org/wiki/Receiver_operating_characteristic

[2]https://github.com/nikbearbrown/INFO_7390/blob/master/Week_5/NBB_Decision_Trees_Random_Forest.ipynb

[3] https://www.quora.com/What-are-the-advantages-and-disadvantages-for-a-random-forest-algorithm

[4]https://github.com/nikbearbrown/INFO_7390/blob/master/Week_4/NBB_Logistic_Regression.ipynb

[5] https://elitedatascience.com/machine-learning-algorithms

[6] https://towardsdatascience.com/ensemble-learning-in-machine-learning-getting-started-4ed85eb38e00

[7] https://www.analyticsvidhya.com/blog/2017/02/introduction-to-ensembling-along-with-implementation-in-r/

[8] https://www.kaggle.com/kailex/talkingdata-eda-and-class-imbalance

[9] https://www.kaggle.com/pranav84/talkingdata-eda-to-model-evaluation-lb-0-9683/notebook

[10] https://www.kaggle.com/nuhsikander/lgbm-new-features-corrected

[11] https://www.kaggle.com/rteja1113/lightgbm-with-count-features

[12] https://www.kaggle.com/aharless/swetha-s-xgboost-revised

[13] https://www.kaggle.com/bk0000/non-blending-lightgbm-model-lb-0-977

[14] Linfeng Zhang and Yong Guan, The 28th International Conference on Distributed Computing Systems: Detecting Click Fraud in Pay-Per-Click Streams of Online Advertising Networks

[15] Ahmed Metwally, Divyakant Agrawal Amr, El Abbadi, WWW 2007 / Track: E*-Applications, Session: E-Commerce and E-Content: DETECTIVES: Detecting Coalition hit Inflation attacks in advertising networks Streams

[16] Hamed Haddadi, Royal Veterinary College, University of London: Fighting Online Click-Fraud Using Bluff Ads

[17] Richard Oentaryo, Ee-Peng Lim, Michael Finegold, David Lo, Feida Zhu, Clifton Phua, Eng-Yeow Cheu, Ghim-Eng Yap, Kelvin Sim, Minh Nhut Nguyen, Bijay Neupane, Mustafa Faisal, Zeyar Aung, Wei Lee Woon, Wei Chen, Dhaval Patel, Journal of Machine Learning Research 15 (2014) 99-140 Submitted 3/13; Revised 11/13; Published 1/14: Detecting Click Fraud in Online Advertising: A Data Mining Approach

[18] Tom Fawcett, Foster Provost, Data Mining and Knowledge Discovery 1, 291–316 (1997) °c 1997 Kluwer Academic Publishers. Manufactured in The Netherlands: Adaptive Fraud Detection

[19] Yongguang Zhang, Wenke Lee: Intrusion Detection in Wireless Ad-Hoc Networks

[20] Feng Dong, Haoyu Wang, Li Li, Yao Guo, Tegawende F. Bissyande, Tianming Liu, Guoai Xu, Jacques Klein, arXiv:1709.01213v4 [cs.CR] 13 Jun 2018: FraudDroid: Automated Ad Fraud Detection for Android Apps

[21] How Marketers Can Detect and Fight Against Ad Fraud: https://thedma.org/blog/marketing-analytics/marketers-can-detect-fight-ad-fraud/#3-1

[22] https://github.com/andersy005/kaggle-talkingdata-adtracking-fraud-detection/blob/master/notebooks/data-preprocessing.ipynb

[23] https://en.wikipedia.org/wiki/Receiver_operating_characteristic

[24] http://dask.pydata.org/en/latest/docs.html

[25] https://keras.io/

[26] https://en.wikipedia.org/wiki/Scikit-learn

[27]https://github.com/nikbearbrown/INFO_7390/blob/master/Week_5/NBB_Deep_Learning_Shallow_Neural_Networks.ipynb

[28] https://www.linkedin.com/pulse/artificial-neural-networks-advantages-disadvantages-maad-m-mijwel/

[29]https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/

[30] https://arxiv.org/pdf/1309.0238.pdf

[31] https://www.sciencedirect.com/science/article/pii/S1026309811002136

[32] http://deeplearning.net/tutorial/mlp.html

[33] http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/