Machine Learning Engineer Nanodegree

Capstone Project

Keyun Shang

April 7st, 2018

TalkingData AdTracking Fraud Detection Challenge (from Kaggle[1] competition)

Project Overview

For companies that advertise online, an overwhelming volume of click fraud can lead to misleading click data and wasted money. Although companies keep trying to find the behaviour pattern of click fraud , the huge amount data makes it almost impossible to be analyzed manually. In recent years, the emerge of high computation power and machine learning has given us the ability to use computer for predicting the user behavior. [2] [3]

TalkingData is china's biggest independent data service platform and had suffered from huge volumes of fraudulent traffic. Through building an IP blacklist and device blacklist , they have successfully prevented click fraud for app developers. However, now they want to step ahead of fraudsters and hope to find an user behaviour pattern that predicts if a user will download an app after clicking a mobile app advertisement, which is the aim of the capstone project.

Problem Statement

Their problem is a binary classification problem. The input are a set of features including ip address, app, device, os ,channel, click time etc., the goal is to predict whether a user download an app after clicking a mobile app advertisement.

Metrics

As this is a Kaggle competition project, the best way for evaluation would be the Kaggle score for the test set. However, it is not appropriate in this study due to the following reasons:

---

[1] https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection

[2] https://pdfs.semanticscholar.org/1518/6bd13f3f5a33598be9930d2e093055bb7c93.pdf (A Novel Ensemble Learning-based Approach for Click Fraud Detection in Mobile Advertising)

[3] http://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=2989&context=sis_research (Detecting Click Fraud in Online Advertising: A Data Mining Approach)

1) The number of submissions per day is limited by Kaggle.
2) If the Kaggle project expires before we complete the study, we can no longer obtain Kaggle score.

Accuracy and F-beta score

Accuracy by itself is not a good metric for our case. As data exploration shows, there is a a highly skewed distribution is our data. In other words, we could simply say all of the data are negative and generally be right , without ever looking at the data.

Therefore, we will use F-beta score as the metric instead. F-beta score combines both precision and recall and should be a good metric for our case.

$$F_\beta = (1+\beta^2) \cdot ((precision \cdot recall) \div (\beta^2 \cdot precision + recall))$$

Precision is the ability of the classifier not to label as positive a sample that is negative. Recall is the ability of the classifier to find all the positive samples. In addition, when β = 0.5 , the precision is more emphasized.

II. Analysis

Data Exploration

Input data fields

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_attributed: the target that is to be predicted, indicating the app was downloaded

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

Exploratory Visualization

The plots below show the number of unique values per feature as well as how the target values are distributed. The data are 100,000 randomly-selected rows of training data.

Fig.1 A plot showing the number of unique values for features like ip,app, device, os, channel. Note that the number of unique values of ip is very high.

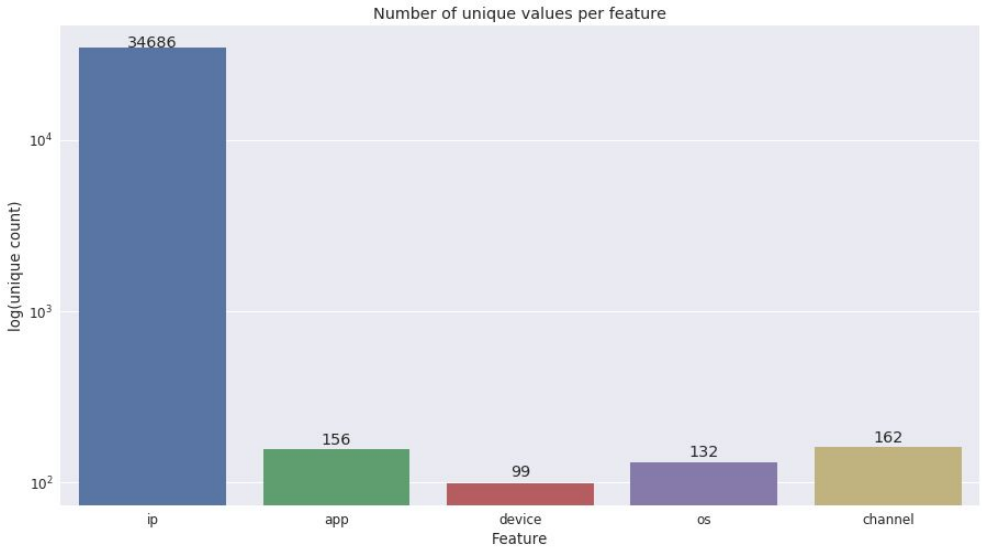


Fig.2 A plot showing the distribution of the probability of the target value. Note that this distribution presents a large class imbalance, as the probability of positive target values is

only 0.25%. We need to deal with the imbalance in data processing.

Algorithms and Techniques

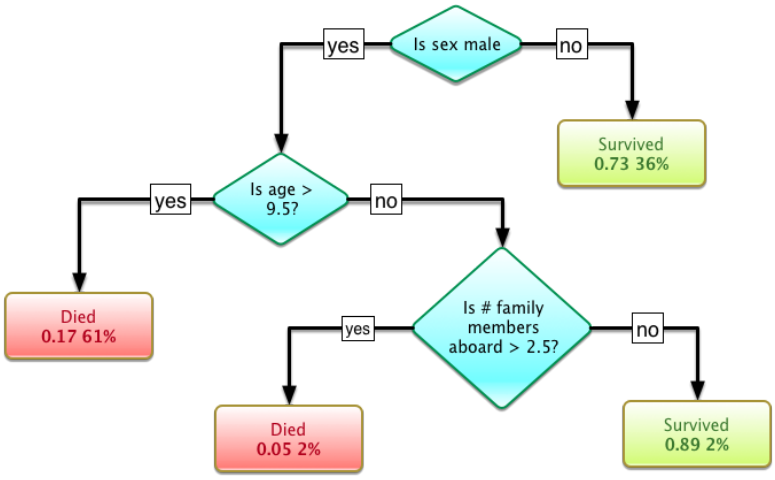
We will use the following common algorithms as below to train the model for comparison.

- Decision tree
- Naive Bayes
- Logistic Regression

Decision tree

(*Main article: Decision tree learning*)

Decision tree is a tree-like structure, where each non-leaf node represents a test on attribute, each branch represents the outcome of the test, each leaf represents the class label , or final outcome. Below is a tree example showing the survival of passengers on the Titanic titanic accident.



The goal of decision tree learning is to construct such a decision tree that predicts a value of target variable based on several input variables. To construct a decision tree, there are several algorithms such as ID3 (Iterative Dichotomiser 3) and CART(Classification And Regression Tree). Usually, constructing a decision tree works top-down, by at each step choosing a variable that best split the data. Different algorithms have different metrics for measuring the "best" by measuring the homogeneity of the target variable within the subsets. For instance, information gain is a metric applied to ID3 algorithm.

Information gain is based on the concept of entropy. Entropy is defined as below.

$$H(T) = I_E(p_1, p_2, \ldots, p_J) = -\sum_{i=1}^{J} p_i \log_2 p_i$$

Where p_1, p_2, p_3, ... represent the probability of each class in the children nodes. The calculation of information gain can be expressed as below.

$$\overbrace{IG(T,a)}^{\text{Information Gain}} = \overbrace{H(T)}^{\text{Entropy(parent)}} - \overbrace{H(T|a)}^{\text{Weighted Sum of Entropy(Children)}}$$

To decide a node that best splits the items, each candidate node is measured with information gain. The split with the highest information gain will be taken as the best. The process goes on until all children nodes have a zero of information gain.

Naive Bayes

(*Main article: Naive Bayes*)

Naive bayes are a set of supervised learning methods based on Bayes' theorem with an assumption of all features are independent of with each other.

Given a class variable y and a dependent feature vector x_1 through x_n, their relationship can be expressed using Bayes' theorem as follows:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)}$$

Above,

- $P(y \mid x_1, \dots, x_n)$ is a posterior probability of class (y , target) given predictor (x_1 through x_n, features)
- $P(y)$ is the prior probability of class
- $P(x_1, \dots x_n \mid y)$ is the likelihood which is the probability of predictor given class
- $P(x_1, \dots, x_n)$ is the evidence which is the prior probability of predictor

Using the naive independence assumption that,

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y),$$

The relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Naive Bayes classifier is a function that combines the naive probability model with maximum a posteriori or MAP decision rule as follows:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

Logistic regression

(*Main article: Logistic regression*)

Logistic regression is a linear model for classification problem. It measures the relationship between the categorical dependent variable (target or class) and the a set of independent variables(attributes, or features) by estimating probabilities using logistic function, which defined as below.

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

t can be expressed as below.

$$t = \beta_0 + \beta_1 x$$

Then, the logistic function can also be written as follows:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

The inverse of the logistic function, g, the logit (log odds) can be expressed as below:

$$g(p(x)) = \ln\left(\frac{p(x)}{1 - p(x)}\right) = \beta_0 + \beta_1 x$$

If there are multiple explanatory variables, the above expression $\beta_0 + \beta_1 x$ can be revised to

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m = \beta_0 + \sum_{i=1}^{m} \beta_i x_i$$

The goal of logistic regression learning is find the best fitting model to describe the relationship between dependent variable and independent variables by generating the coefficients of the above formula. Rather than minimizing the least square error in linear regression, logistic regression chooses parameters that maximizes the log-likelihood of a model.

Through an initial evaluation, we will pick up one best model and tune it with grid search method.

Benchmark

As there is no available benchmark model for this project at present, we take use the naive predictor as the benchmark model. Naive predictor is a simplest predictor that predicts the data with current values. Any new model should be better than or at least the same with it.

III. Methodology

Data Preprocessing

Target value data upsampling

As data exploration shows, there exists a large class imbalance in the training data. We will improve this situation by adding the positive target value data. The probability of positive target value in the dataset is about 0.25% , we increase it to 20% by upsampling the positive target data.

Fig.3 A plot showing the distribution of the probability of the target value after data upsampling.

One-hot Encoding

The features like app, device, os channel , can be seen as categorical data. Although they are numerical data , allowing the model to assumes a nature ordering between categories

could lead to a poor performance. Therefore, we applied one-hot encoding to these categorical data. Finally, we got a total of 6,574 features.

The ip address can also be seen as categorical data and theoretically we should include it into one-hot encoding. However, as the data exploration shows, ip address has a large amount of unique values even just in sample of training data, one-hot encoding to this feature could lead to a significant performance issue. Therefore, we removed ip address from our feature space in data processing.

Shuffle and split data

Although TalkingData has provided a dataset covering approximately 200 million clicks over 4 days[4], training with all of the data could be relatively time and memory consuming. Therefore, I decided to take use of a part of these data only. In fact, the amount of the data I used for training is as following.

- Total number of records: 25,000
- Records with positive label value: 5,000
- Records with negative label value: 20,000

Furthermore, I splitted the training data with into the training/testing sets a ratio of 80:20. For model evaluation, I splitted again the train set using K-fold for cross validation, where K = 3.

Implementation

The implement process can be divided into following steps :

1. Create learners for decision tree, GaussianNB, logistic regression.
2. For each learner,
    a. fit the learner to the train set
    b. Get the predictions on the training subset
    c. Get the predictions on the testing subset
    d. Compute the accuracy and F-beta score on the training subset
    e. Compute the accuracy and F-beta score on the test set
3. Print training time, prediction time, accuracy score and F-beta score for each model

[4] https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data

Main libraries

The main library I used for the implementation is sklearn.

For the learner algorithms, the libraries below are used.

- sklearn.tree.DecisionTreeClassifier
- sklearn.naive_bayes.GaussianNB
- sklearn.linear_model.LogisticRegression

For the metrics of accurrary and F score, the libraries below are used.

- Sklearn.metrics.fbeta_score
- Sklearn.metrics.accuracy_score

For shuffling and splitting data, the library below are used.

- Sklearn.model_selection.train_test_split
- Sklearn.model_selection.KFold

Hyperparameters for model evaluation

The hyperparameters for model evaluation are as follows.

Model	**Hyper parameters**	**Meaning**	**Value tested**
DecisionTreeClassifier	criterion	The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.	'entropy'
	splitter	The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.	'best'
	min_samples_split	The minimum number of samples required to split an internal node	2
	min_samples_leaf	The minimum number of samples required to be at a leaf node	1
GaussianNB	priors	Prior probabilities of the classes.	None

LogisticRegression	penalty	Used to specify the norm used in the penalization.	'l2'
	C	Inverse of regularization strength	1.0
	multi_class	Multiclass option can be either 'ovr' or 'multinomial'. If the option chosen is 'ovr', then a binary problem is fit for each label.	'ovr'

The code snippet for the models' training and evaluation is shown below.

```
# Split and shuffle data with K-Fold for cross validation
kf = KFold(n_splits=3, random_state=None, shuffle=True)

# Create models' instances
clf_A = tree.DecisionTreeClassifier(criterion='entropy', splitter='best', min_samples_split=2,
min_samples_leaf=1)
clf_B = GaussianNB(priors=None)
clf_C = LogisticRegression(penalty='l2',  C=1.0, multi_class='ovr')

# results dictionary
results = {}

for clf in [clf_A, clf_B, clf_C]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    i = 0
    for train_index, test_index in kf.split(X_train):
        results[clf_name][i] = {}

        # Fit the learner to the training data
        start = time();
        learner = clf.fit(X_train.iloc[train_index], y_train.iloc[train_index]);
        results[clf_name][i]['train_time'] = time()  - start

        # get predictions on test set and training data
        start = time();
        predictions_train = learner.predict(X_train.iloc[train_index]);
        predictions_test = learner.predict(X_train.iloc[test_index]);
        results[clf_name][i]['pred_time'] = time() - start

        # get accuracy and f-beta score on training data
        results[clf_name][i]['acc_train' ] = accuracy_score(y_train.iloc[train_index],
predictions_train)
        results[clf_name][i]['f_train'] = fbeta_score(y_train.iloc[train_index],
predictions_train,beta=0.5)
```

```
        # get accuracy and f-beta score on test data
        results[clf_name][i]['acc_test'] = accuracy_score(y_train.iloc[test_index], predictions_test)
        results[clf_name][i]['f_test'] = fbeta_score(y_train.iloc[test_index],
predictions_test,beta=0.5)
        i = i + 1
```

Refinement

Based on the result of the initial evaluation, I chose the logistic regression as the best candidate. For refining the model, I used the grid search. The search dictionary is as follows:

parameters = {'penalty': ['l1','l2'],'C':np.logspace(-3,3,7)}

Here, **penalty** is used to specify the norm used in the penalization.

C is the Inverse of regularization strength, smaller values specify stronger regularization. **np.logspace(-3,3,7)** corresponds to an array of [0.001, 0.01, 0.1 , 1 ,10, 100, 1000].

The code snippet for the grid search is shown below.

```
# choose cross-validation iterator
cv = ShuffleSplit(X_train.shape[0], n_iter=10, test_size=0.2, random_state=0)

# Initialize the classifier
clf = LogisticRegression(penalty='l2',  C=1.0, multi_class='ovr')

# Create the parameters list you wish to tune, using a dictionary if needed.
parameters = {'penalty': ['l1','l2'],'C':np.logspace(-3,3,7)}

# Make an fbeta_score scoring object using make_scorer()
scorer = make_scorer(fbeta_score,beta=0.5)

# Perform grid search on the classifier using 'scorer' as the scoring method using GridSearchCV()
grid_obj = GridSearchCV(estimator=clf, cv=cv,param_grid=parameters,scoring=scorer)

# Fit the grid search object to the training data and find the optimal parameters using fit()
grid_fit = grid_obj.fit(X_train, y_train)

# Get the estimator
best_clf = grid_fit.best_estimator_
```

IV.Results

Model Evaluation and Validation

The evaluations to the three models are as the following tables. The metrics for evaluation include model training time, model predicting time, accuray score and F score on training and sub testing subset.

Since I used K-fold for the cross validation, each table includes 3 folds' results.

Decision tree

No.	Training time(sec.)	Prediction time(sec.)	Accuracy of training subset	F-beta score of training subset	Accuracy of test subset	F-beta score of test subset
1	15.89	1.72	0.9999	0.9999	0.9330	0.8458
2	15.98	1.67	1.0000	1.0000	0.9415	0.8512
3	14.22	1.68	1.0000	1.0000	0.9331	0.8432

Naive Bayes

No.	Training time(sec.)	Prediction time(sec.)	Accuracy of training subset	F-beta score of training subset	Accuracy of test subset	F-beta score of test subset
1	3.93	6.19	0.9104	0.8480	0.9079	0.8318
2	3.92	5.99	0.9018	0.8305	0.8958	0.8081
3	4.00	6.10	0.9220	0.8685	0.9131	0.8584

Logistic regression

No.	Training time(sec.)	Prediction time(sec.)	Accuracy of training subset	F-beta score of training subset	Accuracy of test subset	F-beta score of test subset
1	1.00	1.41	0.9570	0.9220	0.9552	0.9179
2	1.07	1.35	0.9572	0.9238	0.9565	0.9191
3	1.13	1.38	0.9603	0.9288	0.9524	0.9103

Based on the evaluation of the training time , predicting time, accuray score and F-beta score, the logistic regression can be considered as the best of the three.

Final model

The result of grid search is as follows.

No.	Mean	std	params
1	0.00000	0.00000	{'C': 0.001, 'penalty': 'l1'}
2	0.63566	0.01009	{'C': 0.001, 'penalty': 'l2'}
3	0.86720	0.00477	{'C': 0.01, 'penalty': 'l1'}
4	0.87498	0.00542	{'C': 0.01, 'penalty': 'l2'}
5	0.90638	0.00754	{'C': 0.1, 'penalty': 'l1'}
6	0.91252	0.00646	{'C': 0.1, 'penalty': 'l2'}
7	0.91458	0.00413	{'C': 1.0, 'penalty': 'l1'}
8	0.91500	0.00479	{'C': 1.0, 'penalty': 'l2'}
9	0.91425	0.00431	{'C': 10.0, 'penalty': 'l1'}
10	0.91446	0.00372	{'C': 10.0, 'penalty': 'l2'}
11	0.91324	0.00451	{'C': 100.0, 'penalty': 'l1'}
12	0.91422	0.00364	{'C': 100.0, 'penalty': 'l2'}
13	0.91348	0.00434	{'C': 1000.0, 'penalty': 'l1'}
14	0.91389	0.00405	{'C': 1000.0, 'penalty': 'l2'}

Mean is the average of F-beta score, std is the standard deviation.

Based on the observation to the result of grid search, the below hyperparameters can be thought as the best.

{'C': 1.0, 'penalty': 'l2'}

Justification

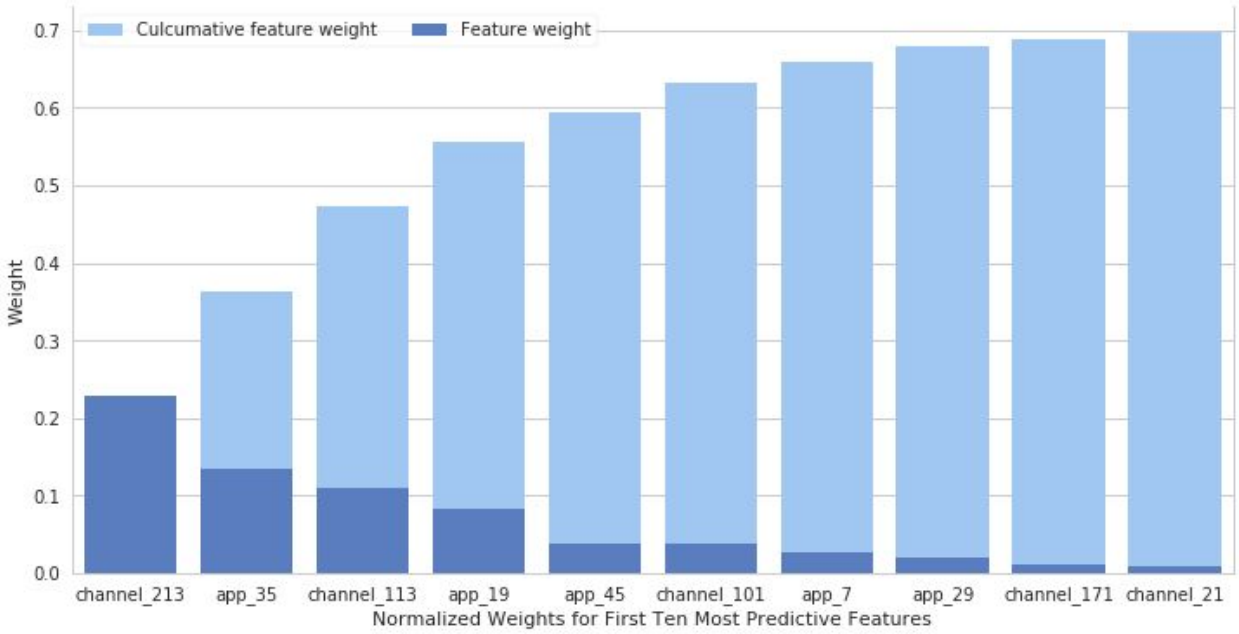
Compared to the benchmark, final model has a much better accuracy score and F-beta score. Therefore, the final tuned model can be deemed a satisfactory solution.

	Benchmark	**Final model**
Accuracy score	0.2000	0.9570
F-beta score	0.2381	0.9179

V.Conclusion

Free-Form Visualization

Fig.4 Normalized weights for first ten most predictive features.



As **Figure 9** shows, we can see that the cumulative feature weight of the first ten most predictive features is near to 0.7. In other words, the top ten important features contribute more than half of the importance of all features present in the data. This suggests that we can reduce the feature space and simplify the information required for model learning, if the training time and predicting time are concerns. The table below shows a comparison of accuracy and F-beta score between the model of full features and the model with top ten important features.

	Full-feature model	Reduced-feature model
Accuracy score	0.9570	0.9498
F-beta score	0.9179	0.9056

Reflection

Eventually I successfully built a model with a F-beta score over 0.90. Meanwhile, I also analyzed the top ten most important features which contribute more than half of the importance of all features. The importance analysis could help company better understand user behaviour.

Additionally, the process used for this project can be summarized as the following steps:

1. The data are explorated and observed and pre-processed
2. A benchmark was created
3. Based on the pre-processed data, three models are trained and evaluated with specified metrics
4. The best model was selected and optimized

I found the step 1 is the most difficult. Since the large amount of categorical features could lead to a large memory consuming in data processing, I had to drop some features like ip address, and implement a customized one-hot encoding method instead of using third-party library to ensure that data processing wouldn't run out of the memory.

Improvement

Although TalkingData provides with a huge data for training, I didn't use all of the data due to the hardware limitation. With better hardware such as larger memory and faster CPU/GPU, we can include much more data into the training process , which may produce better performance than now.

Another thing we can do is to observe and try to understand more about some of the features. For instance , in this project, we dropped ip address because it had a large amount of unique values. However, we could analyze the count of user clicks by ip and include it into feature space.

For academic purpose, in this project, I evaluated decision tree , naive bayes and logistic regression. Actually, there are other algorithms like random forest , that work quite well in real-world situations and deserve a try.