Machine Learning Engineer Nanodegree

Capstone Project

Hrvoje Jerkovic

November 29th, 2017

Predicting mobile users demographics based on applications data and mobile phone specifications

Contents

Project Overview

Mobile networks daily log tremendously large amounts of data about mobile devices that are connected on mobile network. In same way mobile, operating systems on users' devices can collect data on daily usage of mobile phone operations, locations and usage of applications.

Decisions we make every day are creating a picture of who we are and what we value. Those decisions are mainly reflected on our usage of our personal mobile device which holds most important data.

This information is highly relevant for sales and marketing industry. We use product from our preferred brands and they seek better ways of personalizing our experience tailoring it to our individual needs.

Idea of the project is to predict user's demographic data based on available mobile data from mobile network, mobile devices and usage of mobile applications. Although many demographic parameters could be used we are going to focus on predicting users gender and age based on their application download and usage behaviors.

Domain Background

Mobile data traffic and usage worldwide will reach 100 exabytes in 2016 and exceed 450 exabytes in 2021 when mobile subscriptions will exceed nine billion. Telecom data meets the 3Vs criteria of big data: velocity, variety, and volume (some say 6Vs by including volatility, veracity, and value), and should be supported with a big data infrastructure (processing, storage, and analytics) for both real-time and offline analysis [1].

Advances in technology lead to real-time (and near real-time) analytics, which are important for fraud management, network optimization, sales and marketing. This collection of data feeds the predictive analytics engines of machine learning (ML) and deep learning. Telecom big data from CDRs, real-time streaming analytics, and closed-loop optimization of network resources are the means to squeeze every bit of capacity and maximize profits from mobile networks. Telecom companies are like other businesses, and the use cases for big data and machine learning are similar with the most attention given to fraud management and revenue assurance, followed closely by marketing and sales [1].

Problem Statement

Our task is to build a model predicting users' demographic characteristics based on their app usage, geolocation, and mobile device properties.

Doing so will help millions of developers and brand advertisers around the world pursue data-driven marketing efforts which are relevant to their users and catered to their preferences.

Data analysis and exploration

During this project, we'll be working with dataset provided by TalkingData, China's largest third-party mobile data platform. Currently, TalkingData is seeking to leverage behavioral data from more than 70% of the 500 million mobile devices active daily in China to help its clients better understand and interact with their audiences. Since dataset is too large to analyze on ordinary computer we 'we used 16 GB Memory virtual machine for AI operations from Digital Ocean Inc.

is collected from TalkingData SDK integrated within mobile apps TalkingData serves under the service term between TalkingData and mobile app developers. Full recognition and consent from individual user of those apps have been obtained, and appropriate anonymization have been performed to protect privacy [5].

Data description

Description of datasets used for this project:

- gender_age_train.csv, gender_age_test.csv - the training and test set
    - group: this is the target variable we are going to predict

12 classes to predict from this group are: 'F23-', 'F24-26','F27-28','F29-32', 'F33-42', 'F43+','M22-', 'M23-26', 'M27-28', 'M29-31', 'M32-38', 'M39+' . F stands for female age group and M stands for male age group. Final result will be CVS file that contains predictions for each mobile device ID with probabilities that device belongs to each of the demographic groups in following form:

device_id	F23-	F24-26	F27-28	F29-32	F33-42	F43+
1.00208E+18	0.001424	0.005998	0.013605	0.013286	0.025313	0.046103
-1.54786E+18	0.007414	0.013299	0.031228	0.058677	0.072686	0.151391
7.37458E+18	0.023158	0.036713	0.036233	0.158343	0.162774	0.079852

Figure 1. Portion of final CSV file (only female groups shown)

- events.csv, app_events.csv - when a user uses TalkingData SDK, the event gets logged in this data. Each event has an event id, location (lat/long), and the event corresponds to a list of apps in app_events.
    - timestamp: when the user is using an app with TalkingData SDK
- app_labels.csv - apps and their labels, the label_id's can be used to join with label_categories
- label_categories.csv - apps' labels and their categories in text
- phone_brand_device_model.csv - device ids, brand, and models
    - phone_brand: Samsung, Ktouch, hisense, Lenovo, obi, ipair, nubia, youmi, dowe, heymi, hammer, koobee, meitu, nibilu, oneplus, yougo, nokia, candy, ccmc, ,yuxin, kiwu, greeno, asus, panosonic, weitu, aiyouni, moto, xiangmi, micky, bigcola, wpf, hasse, mole, fs, mige, fks, desci, mengmi, lshi, smallt, newman, banghua, epai, pner, ouxin, ximi, haier, bodao, nuomi, weimi, kupo, google, ada, lingyun

Exploration and visualization

Detailed analysis and visualizations were done on dataset, here we will show only most important ones. Following charts will gave is important insights into data. Graphs below shows number of application events throughout whole day and density of application events day by day. Graph below show count of demographic groups.

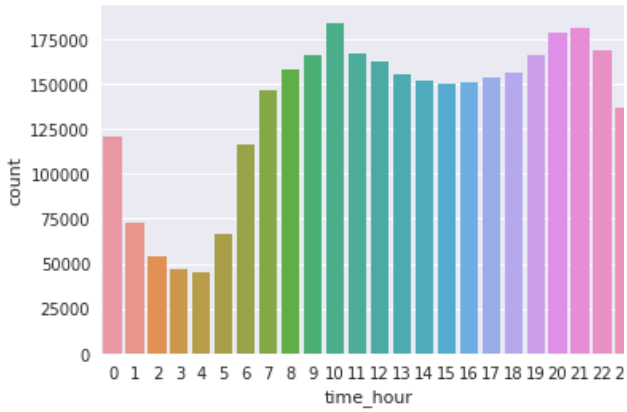


Figure 2. AppEvents density by hour in a day

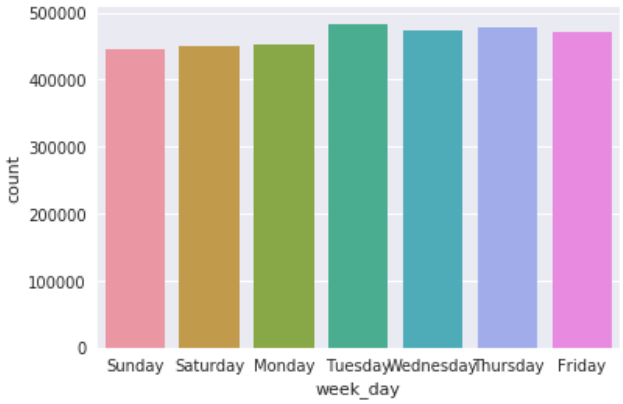


Figure 3. AppEvents through week

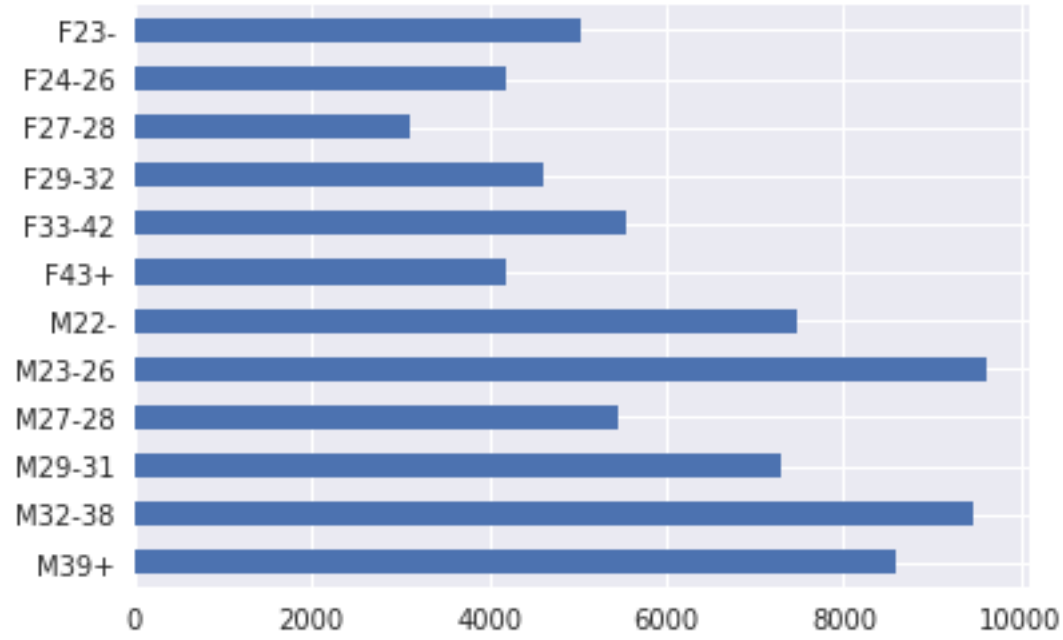


Figure 4. Demographic groups (F - female, M -male)

Events location visualization

Application events location is important factor in data exploration and visualization analysis. It lets us show where is event happening. Images below shows application events across China and specifically for Shanghai region.

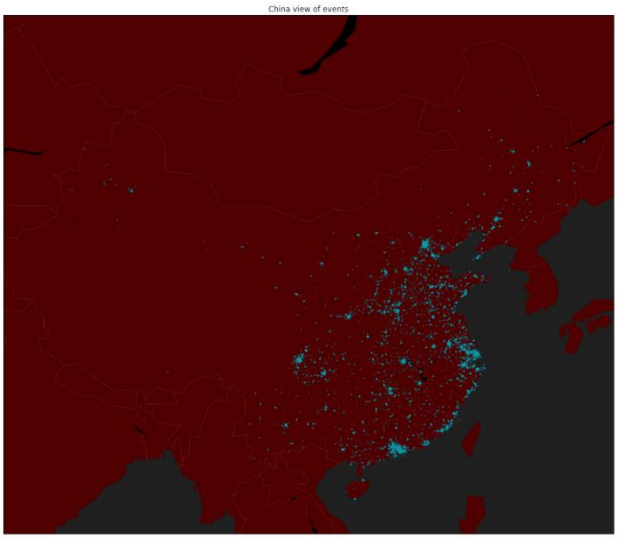


Figure 5. Application events locations - China region

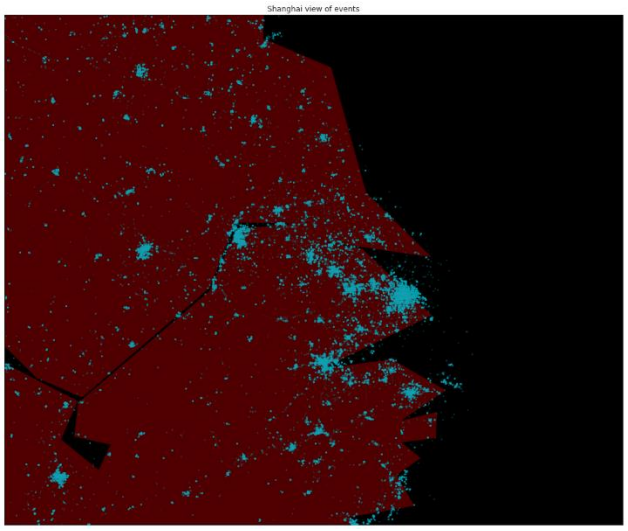


Figure 6. Application events locations - Shanghai region

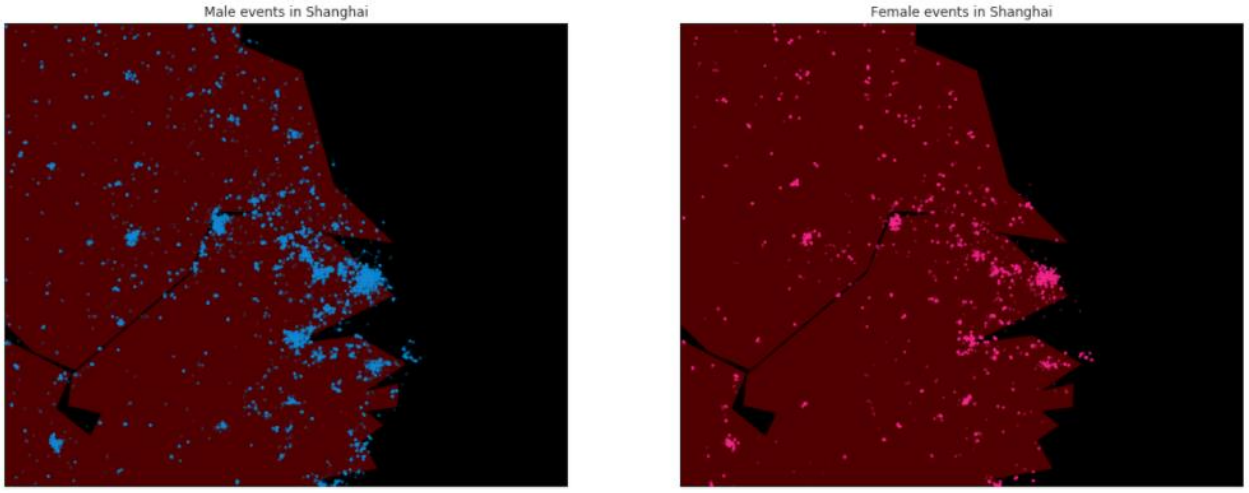


Figure 7. Male and female application events in Shanghai region

Analysis and solution statements

Our task is to build a model predicting users' demographic characteristics based on their app usage, geolocation, and mobile device properties. We need to solve multiclass classification problem This is case where one label needs to be predicted based on several others. This is classical situation where we could apply supervised learning algorithms.

Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations [13].

Many algorithms could be used to approach this problem. Sklearn documentation page on Supervised learning lists all approaches [14].

Algorithm choice - logistic regression

Logistic regression algorithm could be proper choice for problem described. In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme [2].

The reason why we used logistic regression is because solvers implemented in the class LogisticRegression like "newton-cg", "lbfgs", "sag" and "saga" are found to converge faster for some high dimensional data. Setting multi_class to "multinomial" with these solvers learns a true multinomial logistic regression model [15], which means that its probability estimates should be better calibrated than the default "one-vs-rest" setting with additional tuning.

The "sag" solver uses a Stochastic Average Gradient descent [6]. It is faster than other solvers for large datasets, when both the number of samples and the number of features are large.

The "saga" solver [7] is a variant of "sag" that also supports the non-smooth penalty="l1" option. This is therefore the solver of choice for sparse multinomial logistic regression.

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function [2].

A logistic function or logistic curve is a common "S" shape (sigmoid curve), with equation:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

where:

- e = the natural logarithm base (also known as Euler's number),
- x_0 = the x-value of the sigmoid's midpoint,
- L = the curve's maximum value, and
- k = the steepness of the curve [3].

The implementation of logistic regression in scikit-learn can be accessed from class *LogisticRegression*. This implementation can fit binary, One-vs- Rest, or multinomial logistic regression with optional L2 or L1 regularization.

As an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Similarly, L1 regularized logistic regression solves the following optimization problem

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Metric - Multi Class Log Loss

Log loss is the loss function used in (multinomial) logistic regression and extensions of it such as neural networks, defined as the negative log-likelihood of the true labels given a probabilistic classifier's predictions. The log loss is only defined for two or more labels. For a single sample with true label yt in {0,1} and estimated probability yp that yt = 1, the log loss is

-log P(yt|yp) = -(yt log(yp) + (1 - yt) log(1 - yp)) [6]

We'll use this metric for defining our multi-class logarithmic loss. Since each mobile phone device has been labeled with one true class. For each device we must define set of predicted probabilities (one for each class). The formula is then:

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

where N is the number of devices in the test set, M is the number of class labels, log is the natural logarithm, yijyij is 1 if device ii belongs to class jj and 0 otherwise, and pijpij is the predicted probability that observation ii belongs to class jj [5].

12 demographic classes that we need to predict based on available data are: 'F23-', 'F24-26','F27-28','F29-32', 'F33-42', 'F43+', 'M22-', 'M23-26', 'M27-28', 'M29-31', 'M32-38', 'M39+'

As final solution predictions .csv file will be generated that will predict probability of *deviceID* belonging to certain demographic group. Sum of all 12 class predictions should sum up to approximately to 1.

Data processing

Feature selection

In main feature selection we'll excluded following features: phone brand, device model, installed apps, app labels.

One-hot encoding

We are going to one-hot encode every feature and sparse matrices will help deal with a very large number of features we are going to have after one-hot encoding. We need to encode labels with value between 0 and n_classes-1 using LabelEncoder [7].

Sparse matrix of features

In numerical analysis and computer science, a sparse matrix or sparse array is a matrix in which most of the elements are zero. By contrast, if most of the elements are nonzero, then the matrix is considered

dense[8]. We will use scipy.sparse to import csr_matrix function and conduct creation of sparse matrix with that library[9].

A sparse matrix of features can be constructed in various ways. We have used this constructor:

csr_matrix ((data, (row_ind, col_ind)), [shape=(M, N)]) row_ind and col_ind satisfy the relationship [row_ind[k], col_ind[k]] = data[k]

This let's us specify which values to put into which places in a sparse matrix. For phone brand data the data array will be all ones, *row_ind* will be the row number of a device and *col_ind* will be the number of brand. Following that principle we ended up with following feature shapes:

Brand features: train shape (74645, 131), test shape (112071, 131)

Device model features: train shape (74645, 1667), test shape (112071, 1667)

Building feature matrix from app features

Installed apps features: for each device we could have information about applications that device have installed. So we need as many features columns as there are apps and all apps are linked to devices through events. We'll merge device_id column from events table to app_events csv file. Resulting dataframe will have device_id from application, application ID and it's size and we'll than aggregate that with trainrow (trainr) and testrow (testr) columns to know at which row to put each device in the features matrix.

	device_id	app	size	trainr	testr
0	-9222956879900151005	548	18	21594.0	NaN
1	-9222956879900151005	1096	18	21594.0	NaN
2	-9222956879900151005	1248	26	21594.0	NaN
3	-9222956879900151005	1545	12	21594.0	NaN
4	-9222956879900151005	1664	18	21594.0	NaN

Figure 8. First 5 records in Device-Apps dataset

Next step is to build a feature matrix. Row_ind comes from trainr or testr and col_ind is the label-encoded app_id. App labels feature is created by merging app_labels with the deviceapps dataframe as shown on image below.

	device_id	label	size	trainr	testr
0	-9222956879900151005	117	1	21594.0	NaN
1	-9222956879900151005	120	1	21594.0	NaN
2	-9222956879900151005	126	1	21594.0	NaN
3	-9222956879900151005	138	2	21594.0	NaN
4	-9222956879900151005	147	2	21594.0	NaN

Figure 9. Device-Labels dataset

Now we'll preform concatenation of all features which gave us total features shape: train shape (74645, 21527), test shape (112071, 21527).

Concatenation was done using function numpy.hstack which takes a sequence of arrays and stack them horizontally to make a single array.

Performing cross-validation

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a test set X_test, y_test. Note that the word "experiment" is not intended to denote academic use only, because even in commercial settings machine learning usually starts out experimentally.

When evaluating different settings ("hyperparameters") for estimators, such as the C setting that must be manually set, there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. This way, knowledge about the test set can "leak" into the model and evaluation metrics no longer report on generalization performance. To solve this problem, yet another part of the dataset can be held out as a so-called "validation set": training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set.

However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a random choice for the pair of (train, validation) sets.

A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets. The following procedure is followed for each of the k "folds": A model is trained using k-1 of the folds as training data; the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy) [12].

We have used Stratified K-Folds cross-validator which provides train/test indices to split data in train/test sets. This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class [11].

After doing cross-validation we have applied log-loss function.

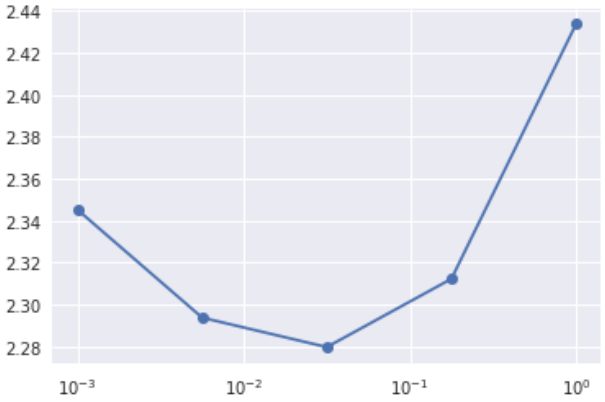
Evaluation metrics - LogisticRegression solvers implementation analysis

We need to find best solver setup for achieving best result against our dataset. LogisticRegression classifier solves multiclass classification problem in form of one versus rest fashion which is what we need so we can perform basic implementation and check results.

But we can also fit a multinomial model that optimizes the multiclass logloss which is exactly our case. So this modification of default setup could improve results since this is our setup.

Benchmark – parameter tuning

C-constant parameter



C is constant - by default: 1.0. Inverse of regularization strength; must be a positive float. Smaller values specify stronger regularization [4]. We've tested values for regularization constant C. Since there is probably a lot of columns which are not so important (rare apps or models of brands) we are probably going to get better score with stronger regularization which means that C value will probably going to be below 1.

Default setup that we could use in our case is following: LogisticRegression(C=0.02) which yilded

2.2797068236722908

```
In [63]: score(LogisticRegression(C=0.01))
Out[63]: 2.2848755470140127

In [55]: score(LogisticRegression(C=0.02))
Out[55]: 2.2797068236722908

In [64]: score(LogisticRegression(C=0.03))
Out[64]: 2.2796060828323981

In [65]: score(LogisticRegression(C=0.04))
Out[65]: 2.2809556715503021

In [66]: score(LogisticRegression(C=0.05))
Out[66]: 2.2828903616369471
```

Figure 10. Logistic Regression algorithm results with only C constant variations

Solver parameter

With best constant C we could try to use different solver parameters. As stated in sklearn documentation:

For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones. For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss; 'liblinear' is limited to one-versus-rest schemes. 'newton-cg', 'lbfgs' and 'sag' only handle L2 penalty, whereas 'liblinear' and 'saga' handle L1 penalty. Note that 'sag' and 'saga' fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing [4].

So based on available algorithms and taken into consideration model of our data; we can conclude that in our example we can only use following solver parameters: newton-cg', 'sag', 'saga' and 'lbfgs'.

```
In [62]: score(LogisticRegression(C=0.02, multi_class='multinomial',solver='saga'))
/root/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/sag.py:326: ConvergenceWarning: T
he max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
Out[62]: 2.2733450166849916

In [67]: score(LogisticRegression(C=0.02, multi_class='multinomial',solver='lbfgs'))
Out[67]: 2.273326572493398

In [68]: score(LogisticRegression(C=0.02, multi_class='multinomial',solver='newton-cg'))
Out[68]: 2.2731559680466482

In [69]: score(LogisticRegression(C=0.02, multi_class='multinomial',solver='sag'))
/root/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/sag.py:326: ConvergenceWarning: T
he max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
Out[69]: 2.273158162504858
```

Figure 11. Solver parameters in Logistic Regression

As we have argued in chapter "Algorithm choice - logistic regression" that solvers like "newton-cg", "lbfgs", "sag" and "saga" are found to converge faster for some high dimensional data.

In our case "sag" and "saga" solvers did not converge but "Newton-cg" in our case with constant C = 0.02 gave best *logloss* result of 2.2731559680466482.

Conclusion

Work done in this project shows that Linear Regression is good algorithm choice for multiclass case classification problem with solvers that can learn a true multinomial logistic regression model. Further testing of other algorithm choices could help find even better solution.

We'll see if gradient boosted decision trees (XGBoost) can help us in our case . Probably with some dimensionality reduction techniques we could make the features more appropriate for tree-based models.

We'll also focus on finding better ways to filter features and to come up with new features that will help algorithm perform better. For example we could find out correlation between events, time of events and location where application appears on each active device.

We can also check *is_active* field on each app in regard with each application event. Also, we need further analysis of locations and check to see if near 0,0 coordinates influence significantly predictions and in what way.

We can also vectorize weights using weighting for applications events or labels and analyze possible feature interactions.

References

[1] Big Data & Machine Learning in the Telecom Network, 3Q 2016 | Technology Analysis Report | AN-2302.

[2] Logistic regression, http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model

[3] Logistic function, https://en.wikipedia.org/wiki/Logistic_function

[4] Sklearn class, sklearn.linear_model.LogisticRegression, http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

[5] TalkingData Mobile User Demographics, Evaluation, https://www.kaggle.com/c/talkingdata-mobile-user-demographics#evaluation

[6] Sklearn, Log loss, logistic loss or cross-entropy loss, http://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html

[7] Label Encoder, http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[8] Sparse matrix, https://en.wikipedia.org/wiki/Sparse_matrix

[9] Scipy - Sparse matrix, https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html

[10] Numpy.hstack, https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.hstack.html

[11] sklearn.model_selection.StratifiedKFold, http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

[12] Cross-validation: evaluating estimator performance, http://scikit-learn.org/stable/modules/cross_validation.html

[13] Supervised Learning, Wikipedia page, https://en.wikipedia.org/wiki/Supervised_learning

[14] SciKit Learn, Supervised learning, http://scikit-learn.org/stable/supervised_learning.html

[15] Christopher M. Bishop: Pattern Recognition and Machine Learning, Chapter 4.3.4