Capstone Proposal

Hrvoje Jerković
December 1st, 2017

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

Domain Background

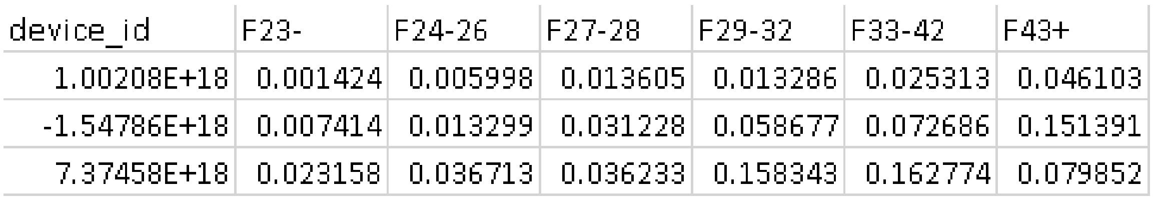
Mobile data traffic and usage worldwide will reach 100 exabytes in 2016 and exceed 450 exabytes in 2021 when mobile subscriptions will exceed nine billion. Telecom data meets the 3Vs criteria of big data: velocity, variety, and volume (some say 6Vs by including volatility, veracity, and value), and should 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. For each demographic group, we need to calculate probability that device ID belongs to that group.

The 12 classes to predict 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)

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.

Same procedures could be used by other mobile data and big data companies to analyze users app and device events.

Datasets and Inputs

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].

Solution Statement

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].

Choice of algorithm

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.

Evaluation Metrics

As evaluation metric of choice we will use Logloss function. Logloss 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].

Project Design

Data Exploration and visualization

Detailed analysis and visualizations will be done on dataset. Various graphs could be generated that could show: application events through days and hours of the days, density of usage of applications by female and male users per each demographic group, count of each demographic group app events and other relevant explorations.

Events location visualization

Application events location is important factor in data exploration and visualization analysis. It lets us show where is event happening. Charts could be generated that will show various regions of the country, usage of apps per individual demographic groups, map of app usage in local areas and in city areas.

Data processing

Feature selection

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

One-hot encoding and sparse matrix of features

After choosing features 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].

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].

At that stage we'll have brand features and device models features data set created.

Building feature matrix from app features and concatenation

Installed apps features: for each device we could have information about applications that device have installed. So next step will be to build a feature matrix. After that we'll preform concatenation of all features.

Concatenation will be 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 [12].

We will 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].

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 tuning

C is constant - by default: 1.0. Smaller values specify stronger regularization [4]. We'll test values for regularization constant C. Since there is probably a lot of data 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. So we will preform testing for all C parameters in rang from 0.1 and 0.01. After that we'll use best C constant to use it with solver parameter for further tuning.

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].

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