Machine Learning Engineer Nanodegree

Capstone Project: Predicting mobile users' demographics based on applications data and mobile phone specifications

Hrvoje Jerkovic

December 7th, 2017

I. Definition

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.

This is case of multiclass or multinomial classification. In machine learning, multiclass or multinomial classification is the problem of classifying instances into one of three or more classes. (Classifying instances into one of the two classes is called binary classification.) [16]. We need to predict demographic information of users mobile based on multiclass information. Classes in our case would be: app usage, geolocation, and mobile device properties.

Strategy for solving the problem

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

Logistic regression algorithm could be proper choice for problem described. Logistic regression, despite its name, is a linear model for classification rather than regression. In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme [2].

We'll follow this strategy for finding solution to our problem:

- **Preprocessing steps**

- Feature selection: we'll use all features in our dataset so this step will be relatively simple.
- One-hot encoding: this is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. We are going to one-hot encode every feature chosen in step 1.
- Sparse matrix of features: sparse matrices will help deal with a very large number of features we are going to have after one-hot encoding. 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].
- Building feature matrix: this final preprocessing step which will help us to build matrix of features. This is a term used in machine learning to describe the list of columns that contains independent variables which should be processed including all lines in the dataset. We'll have to do concatenation of all features.
- Performing cross-validation: Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it. In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples [12].

- **Classifiers**
  - Linear regression will be our classifier of choice because of following benefits specific to our case: we don't have to worry that much about features being correlated, like it's the case with Naive Bayes classifier. We also have good probabilistic interpretation, unlike Decision Trees or SVMs, and we can easily update model to take in new data (using an online gradient descent method), unlike decision trees or SVMs. In our case this classifier is also good because if we expect to receive more training data in the future (which is the case with mobile data) than we will be able to quickly incorporate data into our model.
- **Logloss function**
  - 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.
    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.

As result we'll get probability for each of the classes that needs to be predicted. 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+'
F stands for female age group and M stands for male age group.
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.

Metrics

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.

II. Analysis

Data 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, in brackets we'll define total shape of data. First number represents umber of rows and second number of columns.

- gender_age_train.csv (74645, 4), gender_age_test.csv (112071, 1) - the training and test set

device_id	gender	age	group
-8076087639492060000	M	35	M32-38
-2897161552818060000	M	35	M32-38
-8260683887967670000	M	35	M32-38
-4938849341048080000	M	30	M29-31

Figure 1. gender_age_test.csv sample

device_id
1002079943728930000
-1547860181818780000
7374582448058470000
-6220210354783420000

Figure 2. gender_age_train.csv sample

  - 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 3. Portion of final CSV file (only female groups shown)

- events.csv (3252950, 5), 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.

event_id	device_id	timestamp	longitude	latitude
1	29182687948017100	5/1/2016 0:55	121.38	31.24
2	-6401643145415150000	5/1/2016 0:54	103.65	30.97
3	-4833982096941400000	5/1/2016 0:08	106.6	29.7
4	-6815121365017310000	5/1/2016 0:06	104.27	23.28
5	-5373797595892510000	5/1/2016 0:07	115.88	28.66
6	1476664663289710000	5/1/2016 0:27	0	0
7	5990807147117720000	5/1/2016 0:15	113.73	23

Figure 4. events.csv data sample

event_id	app_id	is_installe	is_active
2	5927333115845830000	1	1
2	-5720078949152200000	1	0
2	-1633887856876570000	1	0
2	-653184325010919000	1	1
2	8693964245073640000	1	1

Figure 5. app_events.csv sample data

- timestamp: when the user is using an app with TalkingData SDK

- app_labels.csv (459943, 2)- apps and their labels. app_id has been anonymised via function which uses also negative numbers. This will not be a problem as long as they are unique for each device. the label_id's can be used to join with label_categories

app_id	label_id
7324884708820020000	251
-4494216993218550000	251
6058196446775230000	406

Figure 6. app_labels data sample

- label_categories.csv (930, 2) - apps' labels and their categories in text. This is list off all most popular application categories, 930 of them, that are most frequently used on mobile phones.

label_id	category
1	
2	game-game type
3	game-Game themes
4	game-Art Style
5	game-Leisure time
6	game-Cutting things

Figure 7. label_categories.csv sample data

- phone_brand_device_model.csv (187245, 3)- device ids, brand, and models

  - Key phone_brands are following: 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

device_id	phone_brand	device_model
-8890648629457970000	Í°ŹçŁł	çş¨çŁł
1277779817574750000	Í°ŹçŁł	MI 2
5137427614288100000	ä¸‰ćź	Galaxy S4
3669464369358930000	SUGAR	ć—¶Í°šć‰‹ćšş
-5019277647504310000	ä¸‰ćź	Galaxy Note 2
3238009352149730000	ÍŤŽä¸ş	Mate

Figure 8. phone_brand_device_model.csv sample data

Important insights

We can see that based on application event (event.csv) we can get information like device_id, timestamp, longitude and latitude and for same event_id we can extract from app_events.csv all applications associated with certain device and weather specific application is installed and active on certain device. Based on app_id we can also extract category of the app from app_labels.csv.

event.csv also holds valuable device_id information which will help us find out device brand and model from phone_brand_device_model.csv.

All those datasets will be useful for building feature matrix which will be trained on training set gender_age_train.csv (74645, 4) which for specific device_id already holds demographic data (sex, age and age group)

And after training we'll check our model on test set - gender_age_test.csv (112071, 1) which only holds device_id.

Data abnormalities

As we can see some events have (lat,lon) = (0,0) which probably means that location couldn't be determined. We can find all event on that location and all events that have longitude and latitude less than 1 which means they are close to that location. That is location is not probable since it's points to sea area close to African cost where there is no land nor island.

Exploratory 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.

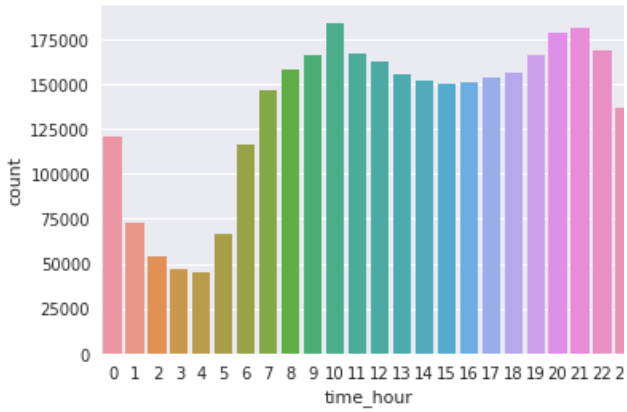

Figure 9. AppEvents density by hour in a day

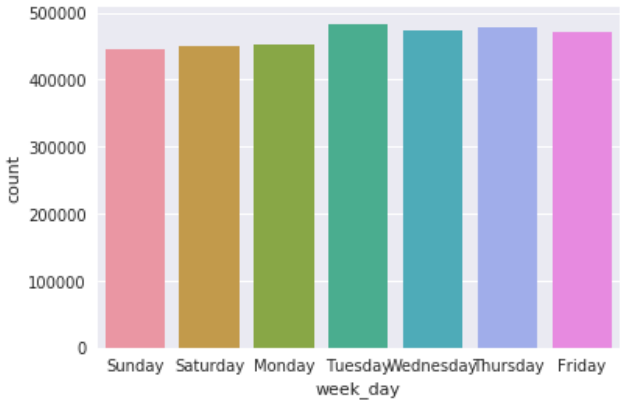

Figure 10. AppEvents through week

As we can see from Figure 9. most intensive mobile activity y is recorded in times between 09:00-11:00 when probably working hours are most intensive but also 19:00-22:00 are hours which are almost equally intensive. We can assume that in those hours people dominantly use mobile applications for social activities and maybe for work as well. Only hours 01:00-05:00 in the morning have significantly lower number of app_events. Activity rapidly falls after 00:00 and after 05:00.

Figure 10. shows slight variations in applications usage from day to day. Lowest activity is recored on Sunday althought Saturday and Monday are close to same level as well and highest activity is on Tuesday. It would be interesting to see break-down analysis of which apps are used most during each day in the week.

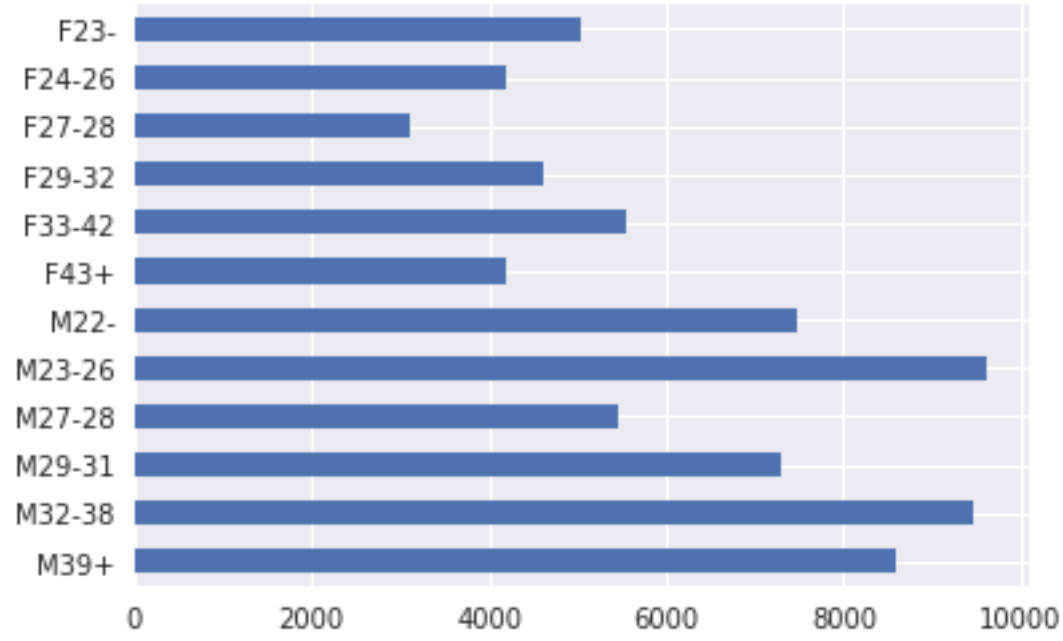

Figure 11. Demographic groups (F - female, M -male)

Figure 11. shows that generally there is more male than female in dataset present. Calculation gave us following result: male count: 47 904, female count: 26 741. There is almost two times more men in dataset than women. We can see that dominating female groups represented in dataset are F23-, F33-42, M22- and in groups with largest number of app_events : M23-26 and M32-38.

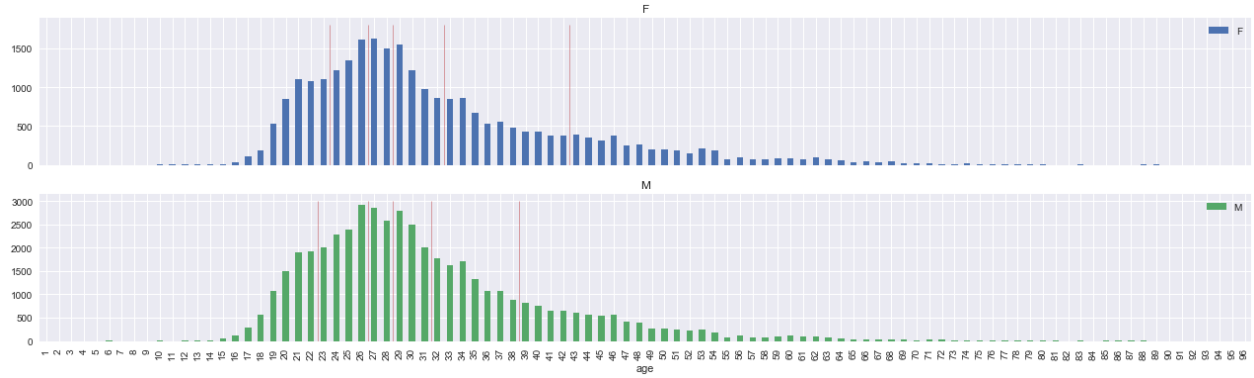

Figure 12. App_events by age and sex.

Since these are just grouped results more detailed analysis could be done to break in down by age on year to year base for male and female individually. This was done in Figure 12. which shows that most app_events for male and female are accounted for ages 26-28.

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.

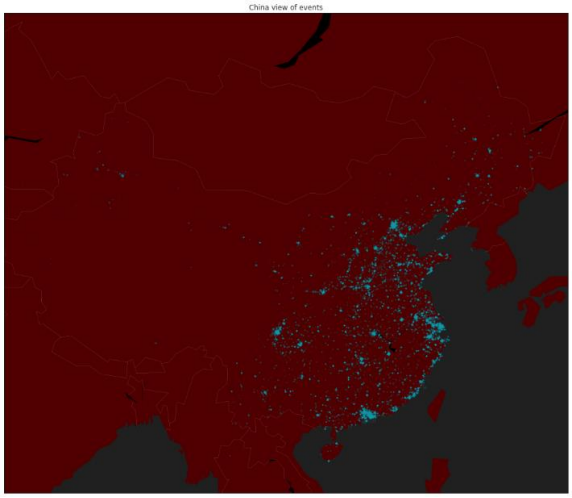

Figure 13. Application events locations - China region

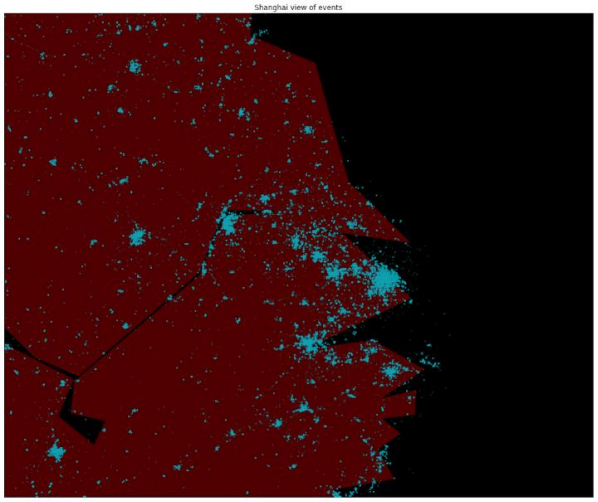

Figure 14. Application events locations - Shanghai region

This map nicely shows population density of China since we can see that majority of app_events are happening in the most densely populated regions of China which is around costal areas of China.

We have also analyzed one city region; in this case we've taken wider area of city of Shanghai. Zooming was done with following parameters for longitude range from115 to 125 and latitude range from 28 to 35. Female and male application events were separated so we can find possible patters.

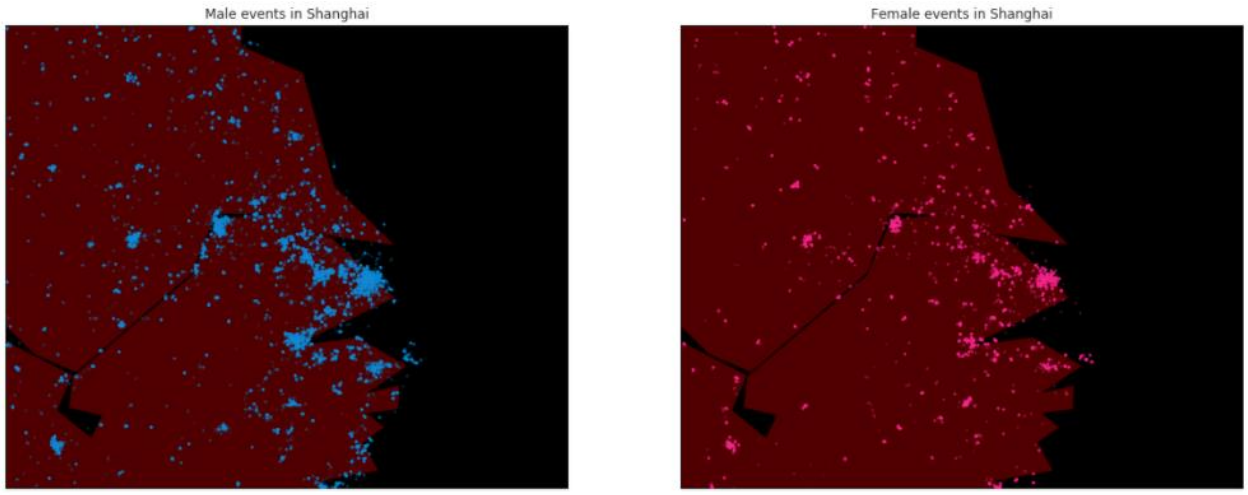

Figure 15. Male and female application events in Shanghai region

We can see that city area is quite dispersed which corresponds to demographic / geographic map of the region. Female and male events are almost equally dispersed in different regions. Further analysis could be done which will show specific male and female app events in various regions and in various times of specific weekdays. This as well could be interesting data for advertisers.

Algorithms and Techniques

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 - Figure 16.), 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].

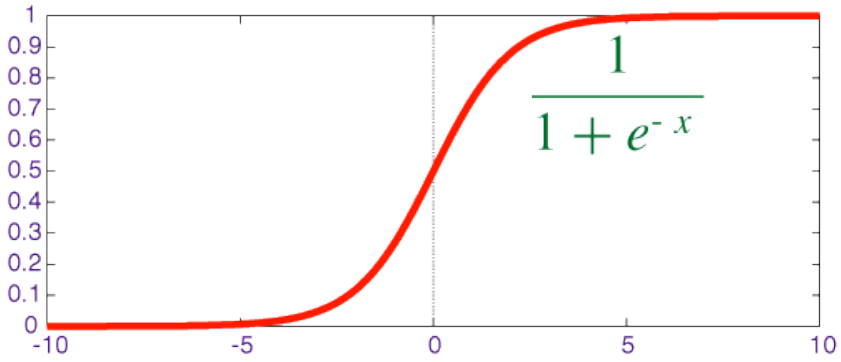

Figure 16. Sigmoid curve

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

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

Benchmark

Comparison with XGBoost algorithm

We'll try to compare efficiency of our chosen Linear Regression model with XGBoost which is an open-source software library which provides the gradient boosting framework. We'we taken starter script and applied it to our case [18] so we can benchmark against our chosen model. Final, tuned, log-loss score with XGBoost algorithm was 2.39667.

Linear Regression benchmark

We need to find best C constant parameter and solver setup for achieving best result against our dataset. Logistic Regression 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 log-loss which is exactly our case. So this modification of default setup could improve results since this is our setup.

Benchmark – parameter tuning

C-constant benchmark

```
cvalue = np.logspace(-3,0,6)
res = []
for C in cvalue:
    res.append(score(LogisticRegression(C = C)))
plt.semilogx(cvalue, res,'-o');
```

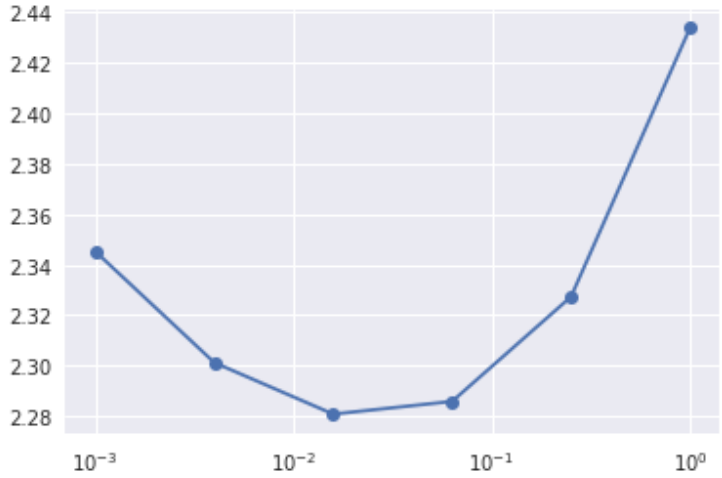

Figure 17. C-constant Benchmark in logspace

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 18. Logistic Regression algorithm results with only C constant variations

Solver parameter benchmark

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 19. 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 *log loss* result of 2.2731559680466482.

III. Methodology

Data Preprocessing

Feature selection

In main feature selection we'll excluded following features: phone brand, device model, installed apps, app labels. Feature space would be way too large if we would include timestamp, longitude and latitude of app_events. Also these features are less relevant in deciding about demographics of mobile users.

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 20. 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 21. 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.

Implementation

Following steps were taken for implementing all procedures explained above.

1. Loading of CSV files
2. Performing various visualizations and inspections explained previously as part of data exploration analysis.
3. Loading of training, testing data and phone data (device brand and model) in its dataframes (g_a_train, g_a_test and phone)
4. Removing duplicate ID in phone dataframe
5. Parsing events.csv with timestamp
6. Loading of appevents (app_events.csv) and applabels (app_labels.csv)
7. Use LabelEncoder to encode labels with value between 0 and n_classes-1.
8. Constructing sparse matrix of features for brand and phone model. We'll construct sparse matrix in following way:

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

This allows us to define what values to put into certain places in a sparse matrix. For example, 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.

Here is implementation examples that shows how "Brand features" were created following procedures defined in steps 7. and 8.

Brand features implementation

```
brand_encoder = LabelEncoder().fit(phone.phone_brand)
phone['brand'] = brand_encoder.transform(phone['phone_brand'])
g_a_train['brand'] = phone['brand']
g_a_test['brand'] = phone['brand']
Xtr_brand = csr_matrix((np.ones(g_a_train.shape[0]),(g_a_train.trainr,
g_a_train.brand)))
Xte_brand = csr_matrix((np.ones(g_a_test.shape[0]),
(g_a_test.testr,g_a_test.brand)))
```

Explanation of procedure:

- fit linear data (learn parameter phone.phone_brand)
- apply transform method (.transform) which applies this transformation model to unseen data
- join two structures with the same index gatrain['brand'] = phone['brand'] and gatest['brand'] = phone['brand']
- apply sparse matrix procedure explained above

In similar way Device model is implemented.

9. Installed apps features implementation
   For each device, we want to have list of installed applications. We'll have as many feature columns as there are distinct apps. Apps are linked to devices through events.

Implementation procedure for "Installed apps" features

Merge device_id column from events table to app_events group the resulting dataframe by device_id and app_id and aggregate merge in trainrow and testrow columns to know at which row to put each device in the features matrix.

```
apps_encoder = LabelEncoder().fit(appevents.app_id)
appevents['app'] = apps_encoder.transform(appevents.app_id)
napps = len(apps_encoder.classes_)
deviceapps = (appevents.merge(events[['device_id']],
how='left',left_on='event_id',right_index=True)

.groupby(['device_id','app'])['app'].agg(['size'])
                        .merge(g_a_train[['trainr']], how='left',
left_index=True, right_index=True)
                        .merge(g_a_test[['testr']], how='left',
left_index=True, right_index=True)
                        .reset_index())
```

10. Next step is to build a feature matrix. Data will be all ones, row_ind comes from trainr or testr and col_ind is the label-encoded app_id.

11. Implement “App labels” features in same way as described in step 10.
12. Concatenate all features in Xtrain and Xtest dataset which will be used for alghoritm implementation
13. Fit and transform data for algorithm implementation
    a. Define targ_encoder, y and n_class variable (categories to predict)
    b. Apply .fit and .transform functions
14. Defining loss- score function.

Implementing loss-score function

```
def score(clf, random_state = 0):
    kf = StratifiedKFold(y, n_folds=5, shuffle=True,
random_state=random_state)
    pred = np.zeros((y.shape[0],nclasses))
    for itrain, itest in kf:
        Xtr, Xte = Xtrain[itrain, :], Xtrain[itest, :]
        ytr, yte = y[itrain], y[itest]
        clf.fit(Xtr, ytr)
        pred[itest,:] = clf.predict_proba(Xte)…
...
```

Loss- score function is based on Logistic Regression classifier and cross validation implementation by using StratifiedKFold cross validation (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 which is exactly what we need in our case – percentage for each demographic group).

Refinement

As explained in *Benchmark – parameter tuning* and *C-constant benchmark* we’ve undertook series of steps to tune-up default Logistic Regression parameters.

Implementation procedure were following

1. Implementation of benchmark for regularization constant C
2. Comparison of default and multinomial attribute usage with chosen C constant
3. Comparison of results for different solvers

IV. Results

Model Evaluation and Validation

During implementation, a validation set was used to evaluate the models in both cases – Linear Regression and XGBoost algorithm.

The final model and hyperparameters chosen in Benchmark chapter were chosen because they performed the best among the tried combinations.

Final model is reasonable and nicely aligned with expected solutions and performs better than most likely possible alternative which is XGBoost algorithm.

Mobile data dataset given here is universal and it can be trusted since it takes log data from standardized mobile applications and devices so we can be confident that this model is applicable in other real-life situations as well and that it will generalize well to unseen data.

Small changes in training data doesn't significantly affect final score. Small changes in app_events.csv from 3.2 million records to something 2-3% smaller number of records doesn't significantly alter final result.

Justification

In comparison with XGBoost algorithm Linear Regression scored better and therefore is justified choice of algorithm which was tested in Benchmark chapter.

Results obtained by Linear Regression are giving reasonable and industry-relevant results that can significantly help marketing experts in concluding final demographic group of tested audience.

All benchmark procedures were executed in Amazon AWS instance in cloud with 8 core processor and 16GB RAM. Final Linear Regresson model should work fine even on up to 100% lager dataset.

V. Conclusion

Free-Form Visualization

We applied t-SNE (Distributed Stochastic Neighbor Embedding) to visualize our dataset. t-SNE is a (prize-winning) technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets just like ones that we have with mobile data. The technique can be implemented via Barnes-Hut approximations, allowing it to be applied on large real-world datasets (up to 30 milion samples) [19] .

t-SNE visualization was used on 10% of training data obtained by stratified sampling. There are 3 dimensions in modified data, and each plot shows two of them. We showed it like that since the data points are not well separable based on 5 features. Features that were used were: ['active', 'counts', 'device_model', 'installed', 'phone_brand']

Neither correlation from all 3 possible combinations didn't show any strongly linearly separable data. Figure 22. show Dimension 1 and Dimension 2 correlation graph.

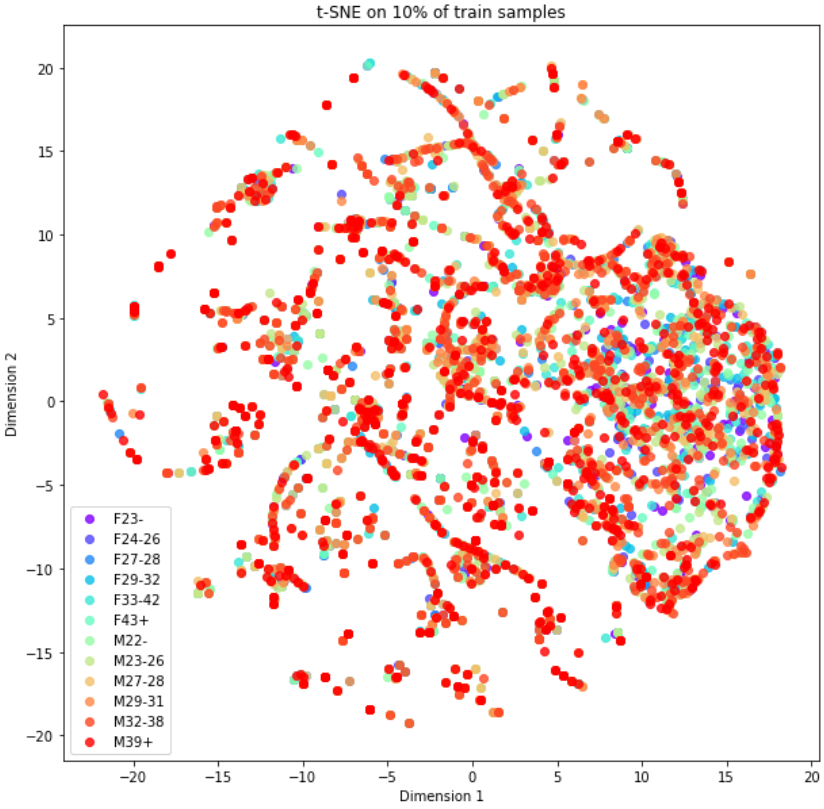

Figure 22. Training data (10%) visualization based on t-SNE with 3 dimensions. Graph is showing correlation between Dimension 1 and 2.

Reflection

This project shows how to do a logistic regression on application events, application labels and phone data for mobile data available with an efficient way of building sparse feature matrices which is one of the best ways for properly constructing dataset for testing and training in our specific case.

Project brings very interesting aspect in regarded with demographic analysis. Considering that in some time in most of active population in developed or even undeveloped countries will almost certainly carry mobile phone with themselves – we can create very accurate model dynamic real-time models that can predict various demographic parameters like: places and times occupied by certain demographic group, anticipated application activity, demographic movements by seasons, etc.

First possible problem with choosing this project was probability of getting larger class imbalance regarding demographic labels that needs to be predicted on dataset. Research done in exploratory analysis showed that this imbalance was not that large so but remains worry whether this model should be a good fit in case there is similar dataset that has greater imbalance training dataset. After some research, we've concluded that this will not be problem. Research in this field found out that for some methods of model construction sample imbalance was not an issue at all – not even a tiny amount. For logistic regression, there was absolutely no benefit to creating a balanced sample [18].

Pandas (data structures and data analysis tools we used) seems to allow duplicate keys in indices. So usual possible point of error in final results might lie here. We need to check for duplicates in fields that are supposed to be unique, that is procedure which must be done while working with such dataset like mobile data. If this step is omitted that extra data rows appearing after merges and can greatly influence result.

If we take into account all said here we can conclude that this model can be good fit for any type of similar mobile data.

Our first choice might have been random forest algorithm or an extra trees model. But large number of sparse features are a not a good fit for decision trees based algorithm. On the other hand, a linear model can be trained fast on this kind of data and turns out to perform quite well.

Improvement

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 see that gradient boosted decision trees (XGBoost) can't help in our case . Probably with some dimensionality reduction techniques we could make the features more appropriate for tree-based models and try with them.

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

[16] Multiclass Classification, https://en.wikipedia.org/wiki/Multiclass_classification

[17] Handling Imbalanced data when building regression models, https://www.analyticbridge.datasciencecentral.com/forum/topics/handling-imbalanced-data-when-building-regression-models

[18] XGBoost simple starter, https://www.kaggle.com/zfturbo/xgboost-simple-starter

[19] t-SNE, https://lvdmaaten.github.io/tsne/