Capstone Project

Machine Learning Engineer Nanodegree

Ludovica Gonella

November 20th, 2016

I. Definition

Project Overview

Mobile data companies have access to a large number of data from their users. These information can include the geographical position, applications installed and used and details on the device. TalkingData, China's largest third-party mobile data platform, is interested in better understanding users' behaviour in order to provide their clients with information that would help them design and target their product more efficiently. However, in order to maintain user's anonymity, some personal data are concealed: of these age and sex are the most important.

The objective of this project is to define a classifier able to identify the sex and age group of TalkingData users. The project featured as a Kaggle competition and can be found at this link:

https://www.kaggle.com/c/talkingdata-mobile-user-demographics

Problem Statement

The goal was to train a classifier to return one among 12 categories between sex and age group. The step I undertook to solve this problem were:

1. Analyzed classes distribution to check if data were skewed
2. Performed data analysis to identify the features most effective in splitting the data
3. Defined a baseline model
4. Engineered the features identified in point 2
5. Tuned and trained a logistic regression and a random forest model and compared their performance
6. Selected the best performing model and further tuned its hyperparameters
7. Thoroughly validated the model performance using log-loss, F1-score and confusion matrix

Metrics

For validating the performance of the models I used three metrics: log-loss, F1-score and confusion matrix.

Log-loss

Logarithmic-loss, or log-loss, is the cross entropy between the probability distribution of true values and the predicted ones. It can be expressed as:

$$L = -\sum_{i}^{N}\sum_{j}^{M} y_{ij} log(\hat{y})$$

Where N is the number of inputs, M is the number of classes, $y_{ij} \in \{0,1\}$ is a binary indicator that states if input *i* belongs to class *j* and $\hat{y}$ is the predicted value.
Log-loss is a soft method for measuring accuracy as it does not punish the misclassification of an instance directly, but the confidence with which such decision was taken. If the classifier was highly confident that an instance of class A belonged to class B instead it would be penalized much more than a classifier that took the same decision but with less confidence. The bigger the score, the worst the model performs (the best score is 0). I used this metric as score since it was a requirement of the Kaggle competition, also it gives a general idea of the performance of the model.

F1-score

F1 score is the weighted average of precision (p) and recall (r). Precision is the number of correct positive results (true positives, tp) divided by the sum of true positives and false positives. Recall is the ratio of true positives to the sum of true positives and false negatives which is the totality of all the actual positives values.

$$F1 = 2\,\frac{p*r}{p+r}, \quad p = \frac{tp}{tp+fp}, \quad r = \frac{tp}{tp+fn}$$

F1 score can vary between 0 and 1, where 0 is the worst score and 1 is the best. In sklearn the multi-class implementation the score is the weighted average of the F1 score of each class. By averaging precision and recall, the F1 score gives a more in depth evaluation of the performance of the model.

F1 is particularly recommended when, like in this case, the distribution between classes is skewed. Accuracy simply measures the ratio between all the instances correctly classified and the total number of inputs. If class 1 is more represented than class 0 it is possible that the model learns to better recognize 1s over 0s. However, in a situation like this accuracy can still be pretty high because class 1 represents the majority of data and thus has a much higher weight than class 0. In these cases F1 scores returns more conservatives estimate of the model's behaviour as it takes into the account its performance for each class and then performs a weighted average.

The *average* parameter was set to *weighted* because it calculates metrics for each label, and finds their average weighted by support (the number of true instances for each label). [1]

[1] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

Confusion matrix

Confusion or error matrix, is a contingency table that allows the visualisation of the performance of a supervised classifier. The rows of the table represent the instances in an actual class while the columns are the instances of a predicted one. Thanks to this representation it is easy to recognize where the model confused the classes and classified them incorrectly: the diagonal elements are those that the classifier predicted correctly while the off-diagonal ones are those mislabeled. In the case of two classes the confusion matrix appears as the example below.

True Positives	False Negatives
False Positives	True Negatives

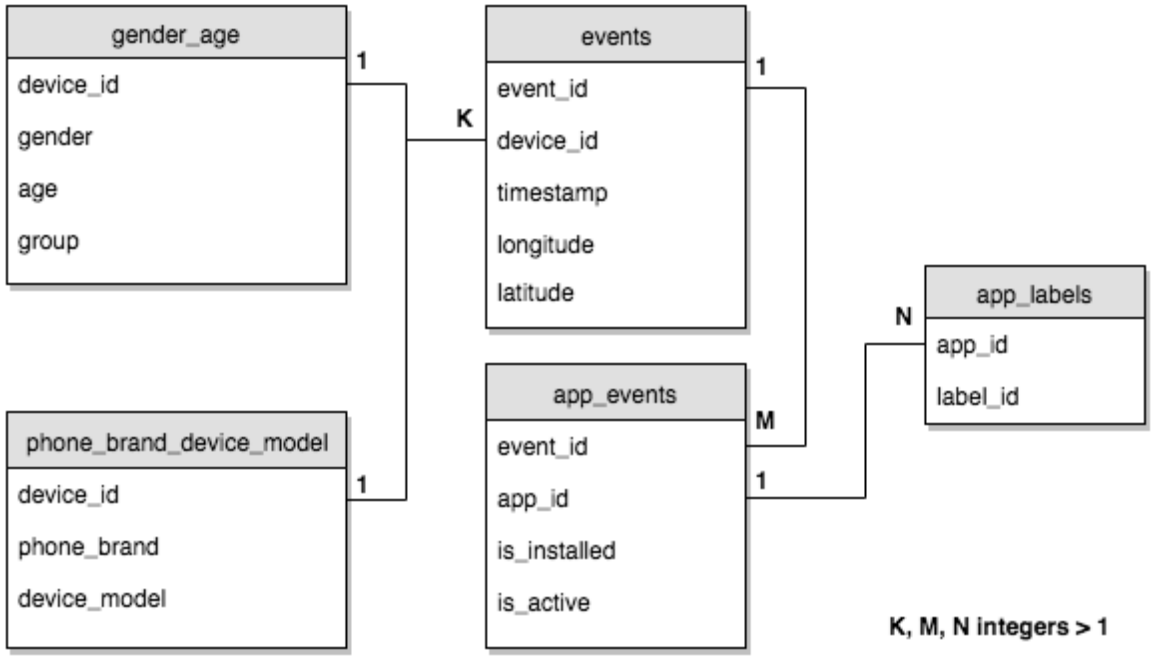
II. Analysis

Data Exploration

Data for this project 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 users of those apps have been obtained by the company and appropriate anonymization has been performed in order to protect users' privacy. Due to privacy reasons, details on how gender and age data were obtained were not provided but they are guaranteed to be accurate for prediction[2]. Data were provided as nine different files, seven of which represent the features organized in different types while two of them contains the ids to select the training and test set. Below there is a representation of the features files and a description of each label.

- *Gender:* M/F string.
- *Age:* integer number, range 1-96.
- *Group:* is the label to predict.
- *Device_id:* unique long integer that identifies each user.
- *Event_id:* integer that identifies events which are logged every time a user uses TalkingData SDK.
- *Timestamp*: instant of when the event was recorded. All the data were collected in one week period.
- *longitude/latitude*: position of the event recorded.
- *App_id*: long integer that identifies each app.
- *Label_id*: integer that represents an app category.
- *is_installed/is_active*: two binary columns that identify which app are installed or active on a device for each event.
- *phone_brand/device_model:* strings with brand/model of each device.

[2] https://www.kaggle.com/c/talkingdata-mobile-user-demographics/data



Gender and *age* are values used only for exploring the dataset and not in training. *Phone_brand* and *device_model* need to be factorized for the classifier to be able to deal with them. All the training and test data need to be created by joining each dataset with *gender_age_train* and *gender_age_test* on *device_id*.
Some device ids were linked to two phones, of these only the first instance was kept and the other one was dropped. Also, when joining *events* and *app_events* on *event_id* it turned out that some ids from *events* were missing in *app_events* resulting in some rows with several *NaN* values. At the end of the analysis I decided not to drop those rows because this problem affected training and test set alike. I was afraid that by removing these examples from the training set the performance on the test set would be affected. I could not remove the rows from the test set because of the Kaggle's requirement of a defined number of rows in the final submission.

The majority of the variables are categorical, for example the *app_ids* use in an *event*, and therefore the outlier analysis does not apply here. Given time I would have analyzed the frequency of each values but I think that ignoring this analysis for now will not dramatically negatively affect the performance of the model.

The training set with the final features was very large and in order for the algorithm to run on my pc I had to use sparse data to train the logistic regressor and a feature transformation algorithm on the app features to train the RF.

The final set of features used is too big to report an example here but the head of the training set is displayed in *notebooks/data.ipynb.*

Exploratory Visualization

Labels

The first thing I checked was if the classes were equally represented or if data were skewed. As can be seen from Figures 1 and 2 data are not uniformly distributed among groups within each sex. Moreover the overall number of female instances is lower than males (Figure 3), more precisely the ratio between the genders is M65% - F35%. In Figure 4 can be seen how in this dataset millennials are largely more numerous than other generations.

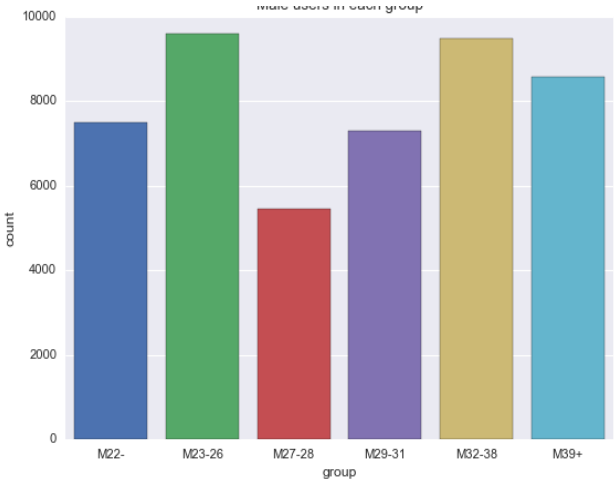


Figure 1: Male groups

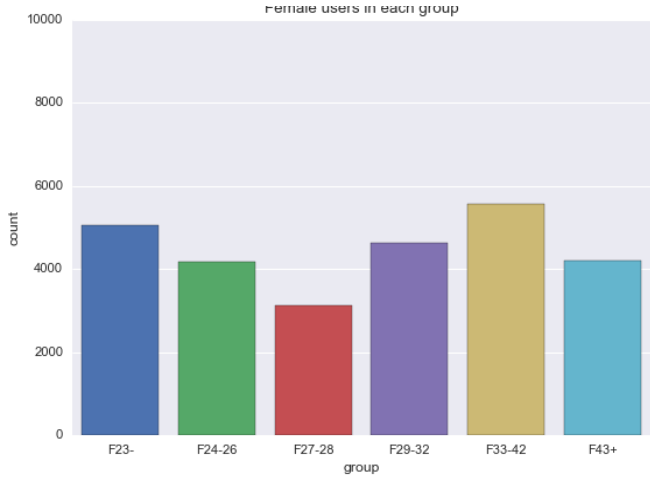


Figure 2: female groups

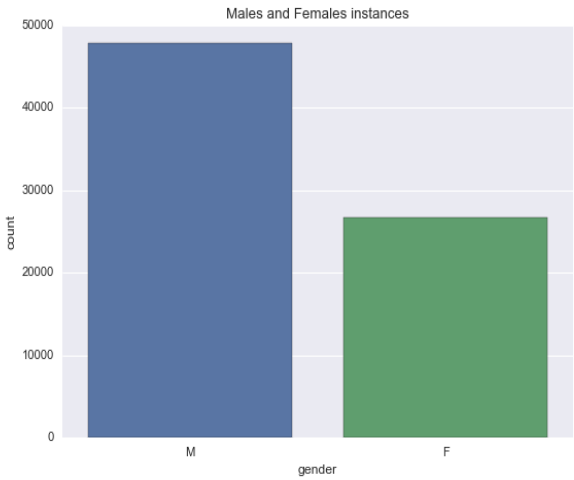


Figure 3: Males and females instances

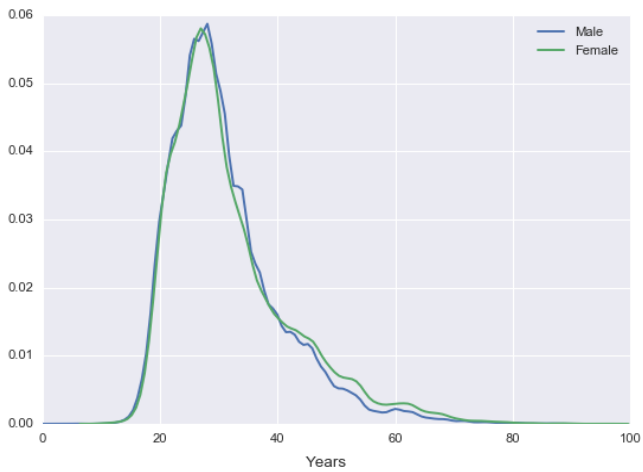


Figure 4: Age percentage distribution divided by gender

Time

Figure 5 represents male and female activity per day of the week. The ratio between genders is circa equal to M74%-F26% for each day. The overall activity per day is relatively stable. Figure 6 represents the activity per hour of the day divided by gender. There is a natural drop in activity during night but the ratio between male and female users is M72%-F28%.

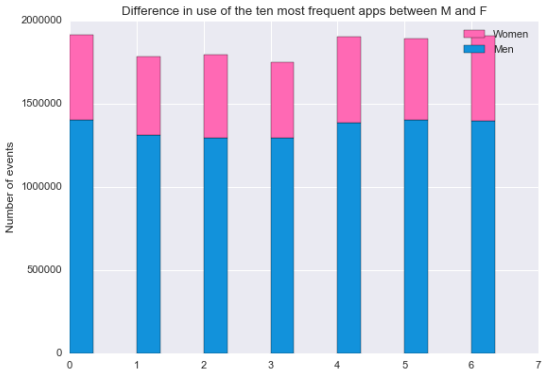


Figure 5: Male and female activity per day of the week

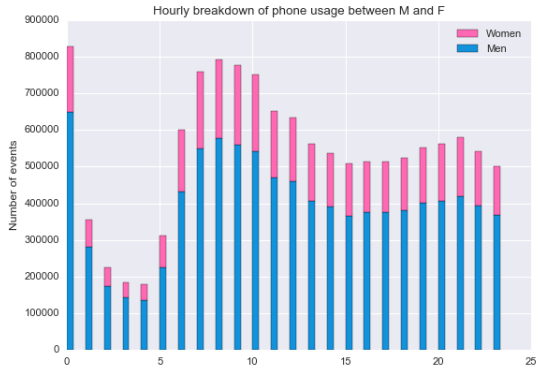


Figure 6: Male and female activity per hour of the day

Position

In order to understand if position could be a good feature to separate the classes I plotted the location of all events divided by gender on the world map. As can be seen from Figure 7 the majority of events take place in China. From the closeup in Figure 8a, it seems that position could indeed be a good indicator of the sex of the users. Figure 8b represents how the age of the users is distributed. Figure 8c represents the number of events per grid area, as can be seen the peaks of events are concentrated in the bigger urban areas. By overlapping Figure 8b and 8c, an interesting trend pops out: as already proven the majority of people in this dataset is young, and Figure 8b shows that they live mainly in cities while older people occupy the countryside. This means that the number of events per area can be used to have an idea of the age and sex of the user.

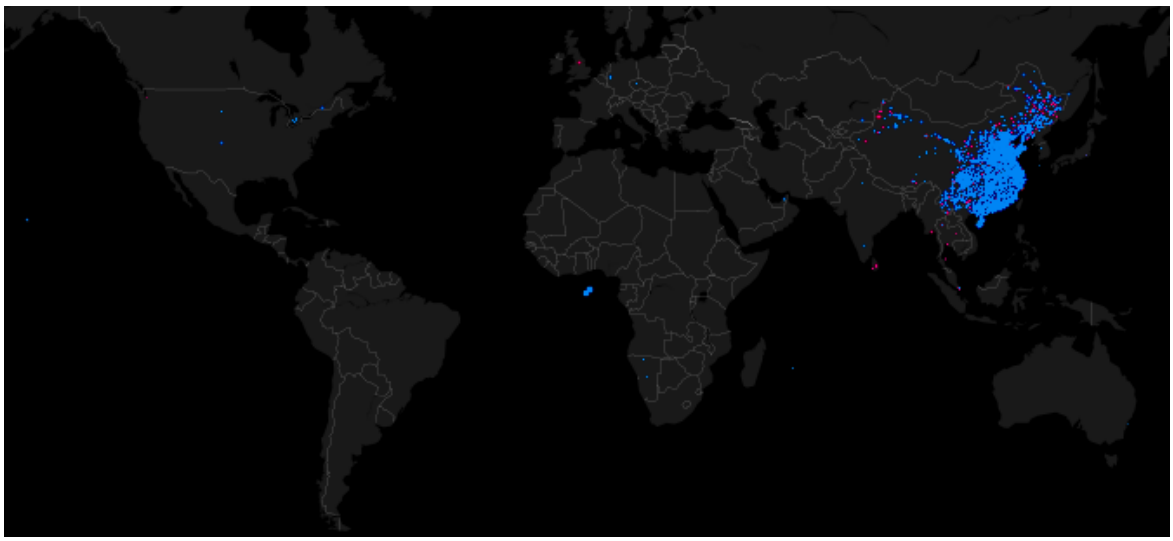
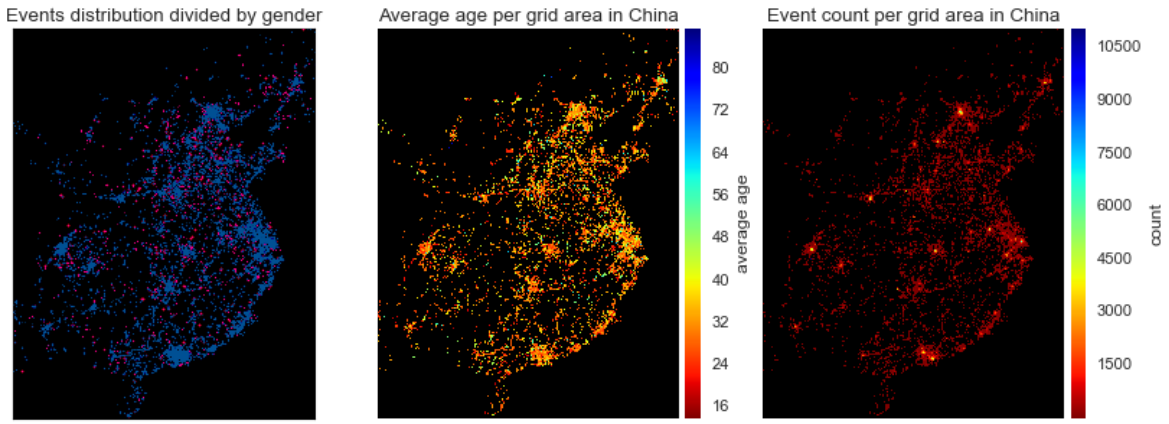


Figure 7: worldwide distribution of events.



(a) *(b)* *(c)*

Figure 8: Location of events plotted according to gender, age and number of events

Apps and labels

Figures 9 and 10 show the ten most used apps among females and males users respectively. For each app both the number of male and female users was plotted in order to have an idea of the percentage of use for one sex and the other. This analysis shows that the ratio of females and males for each app is circa of 30%-70%, which is the same distribution of sexes in the dataset. What is interesting however is that these apps are ordered differently. This could be used as an indicator in the features.
Figures 11 and 12 represents the distributions of apps labels installed and active per device. As can be seen there is a clear shift of the distributions of the number of installed apps for males and females. Also in active apps it appears that more women use less the phone than men (initial peak) and that there is a clear threshold of activity that separates male users from females.

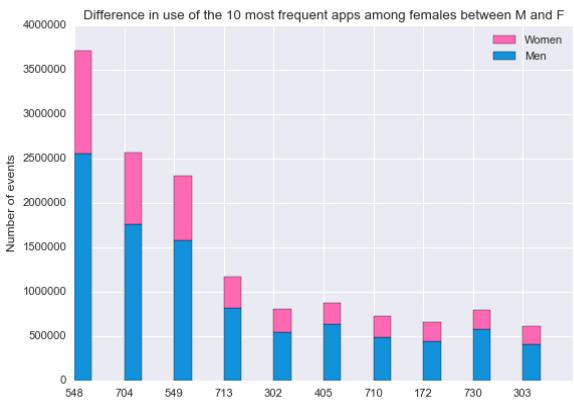


Figure 9: Most used apps among female users

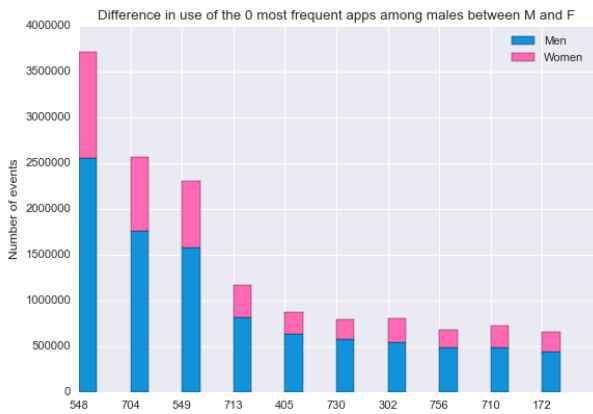


Figure 10: Most used apps among male users

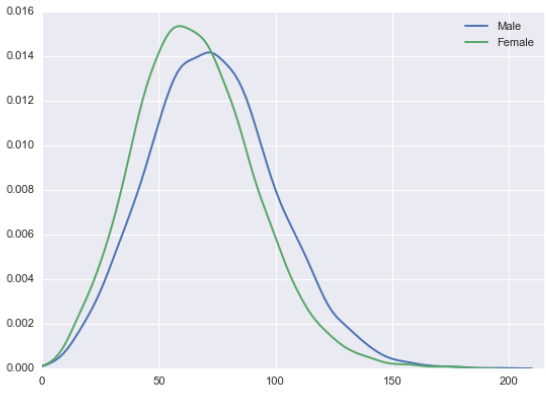


Figure 11: Distribution of how many apps are installed on each device given the gender

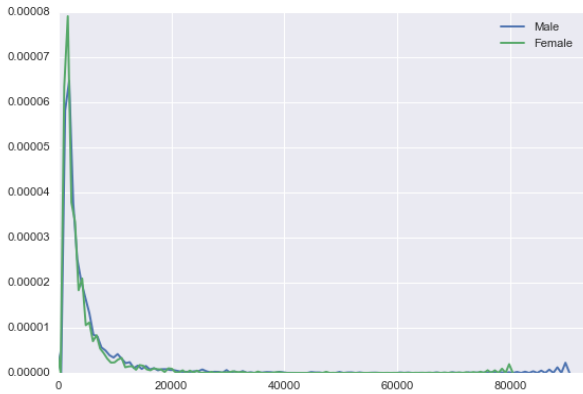


Figure 12: Distribution of how many apps are active on each device given the gender

This same analysis was performed on the installed *app_ids* and active *app_labels*. Since the results did not produce any different insight from the ones presented I decided to not include those graphs in this report. The full analysis can be found in the folder *notebooks.*

Phone specs

Figures 13 and 14 represent the ten most popular brand and models, respectively, among genders. The proportion of males and females users for most of the brand is roughly M60%-F40%. OPPO and Vivo (fourth and fifth brand), however, are more balanced: OPPO M55%-F45%, Vivo M53%-47%. Meizu, the sixth, is more biased toward males: M73%-F27%. Similarly, the majority of models are divided as 65% males and 35% females, close to the training set gender distribution. Galaxy Note 3 and 2 are more balanced: Galaxy Note 3 M58%-F42%, Galaxy Note 2 M51%-48%.
In order to find new ways to discriminate among the classes I created new features using phones' technical specs. I collected data on price (eur), screen size (in), ram (Gb), camera (Mpx) release month and year for 80% of the phones listed. Figure 15 represents the distribution of the price of the phones given the gender. As can be seen the two curves are very similar, however, the three central peaks are slightly horizontally offsetted and around 300€ the females' curve is bigger than the males' one. Overall the trend seems to suggest that women tend to spend more than men. Figure 16 is the distribution of phones' camera specs divided by gender. Here the two curves are the same minus a small vertical offset. All the remaining specs when plotted gave a result similar to Figure 14. Given that for all of these features there is little difference between males and females users I decided not to include them in the final model as I was afraid that they would introduce very little information and possibly some noise.

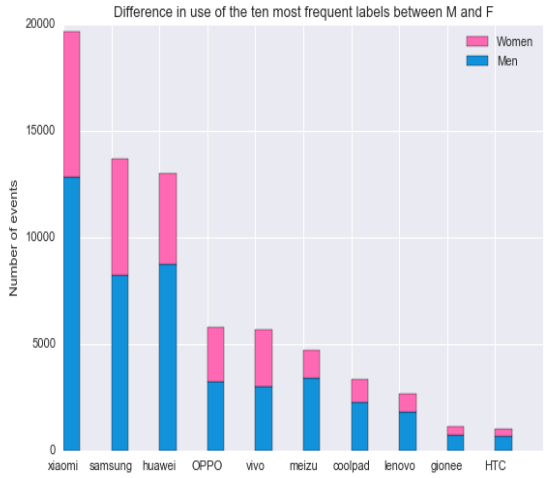


Figure 13: The overall ten most popular brands

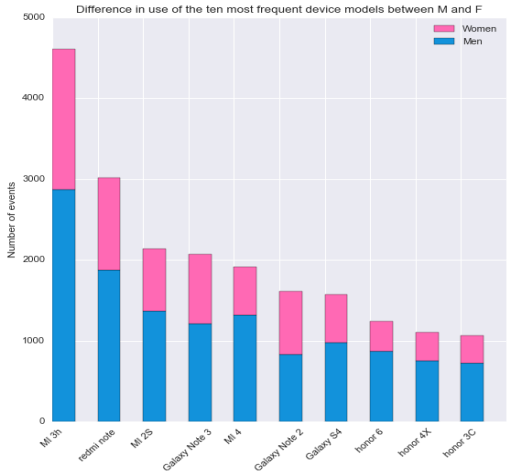


Figure 14: The overall ten most popular models

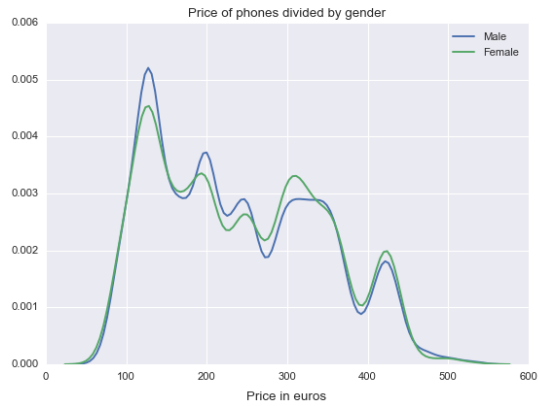


Figure 15: Distribution of phones' price divided per gender

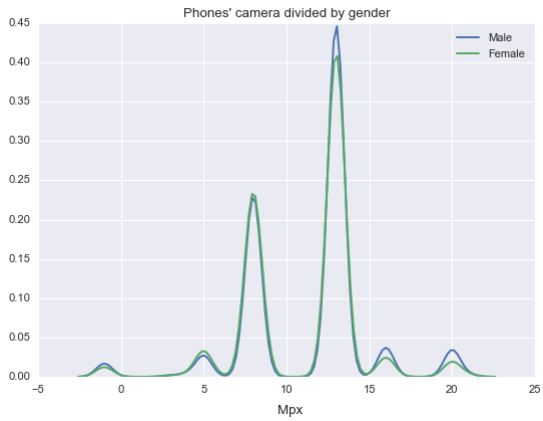


Figure 16: Distribution of phones' camera specs divided per gender

Algorithms and Techniques

The classifiers trained are a Logistic Classifier (LC) and a Random Forest (RF). LC was chosen as it is a classifier widely used in industry and the output was immediately ready for submission on Kaggle. RF was picked because of its versatility in adapting to data with large feature spaces. Both classifiers have the possibility to train on multiple cores which makes them convenient timewise.

Logistic Classifier

The logistic classifier uses data to train a logistic function in order to output a binary response.

$$\text{Logistic function: } y = \frac{1}{a + be^{-cx}}$$

Sklearn provides a multi-class that uses the cross-entropy loss to fit the data. By selecting "balanced" for the parameter "class_weight" the target labels are used to automatically adjust weights inversely proportional to class frequencies in the input data[3] . The hyperparameters that I chose to tune for this model were:

[3] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

- *C:* Inverse of regularization strength, positive number. Like in support vector machines, smaller values specify stronger regularization.
- *Class_weight:* can be *None* or *balanced.* If *balanced* the classifier uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as $n_{samples} / (n_{classes} * size_{class})$.
- *Solver:* Algorithm to use in the optimization problem. The default one is *liblinear* while the one that performed best was *newton-cg* (nonlinear conjugate gradient method, it is generally used to find the local minimum of a nonlinear function using its gradient $\nabla_x f$ alone[4]).

Random Forest

Random Forests are an ensemble learning method, they combine the predictions of different learning models in order in order to improve generalizability and robustness over a single estimator. In particular they fit a user specified number of tree classifiers on different subsets of the training data. It then use average to improve accuracy and control over-fitting. This procedure improve model performance because it decreases the variance of the model with little or no increase to the bias. The size of the sub-samples is the same of the original input, what changes are the features that are taken into account: at each split a different subset of features is picked up by the algorithm and one of them is selected. The split is then performed on that feature. This procedure is done in order to limit the development of correlation among different trees: this would likely happen as usually there are few features that are strong predictors and all the trees would end up doing the same split on those.
In this project the parameters I chose to tune are:

- *Max_depth*: The maximum depth of the tree.
- *min_samples_split* : The minimum number of samples required to split an internal node.

Similarly to LC, RF also have the option to automatically generates weights that are inversely proportional to the frequency of each class.

Stratified K-Fold and Randomized Search Cross Validation

KFold divides the dataset in k groups of samples, called folds, of equal sizes (if possible). The prediction function is learned using k - 1 folds, and the fold left out is used for test. Sklearn's *StratifiedKFold* is a variation of *k-fold* which returns stratified folds: each set contains approximately the same percentage of samples of each target class as the complete set[5]. This method is important when dealing with heavily skewed datasets because it ensures that the model is trained on samples that have the same population distribution as the whole training set.

Grid search trains the classifier on all the possible combinations of these values and selects the best according to a default or user specified metric. With randomized search, conversely, the user selects a maximum number of combinations to be considered and for each parameter the distribution from which to pick its value[6]. Although this method does not

[4] https://en.wikipedia.org/wiki/Nonlinear_conjugate_gradient_method
[5] http://scikit-learn.org/stable/modules/cross_validation.html
[6] http://scikit-learn.org/stable/modules/grid_search.html

guarantee a search as precise are grid search, it is much quicker and is suggested when the number of hyperparameters to tune is large.

Single Value Decomposition

Single Value Decomposition (SVD) is an alternative way to compute Principal Component Analysis (PCA). Normally PCA requires to obtain the eigenvalues and eigenvectors of the covariance matrix XX^T where X is the data. The covariance matrix is symmetric, thus diagonalizable, and the eigenvectors can be normalized in order to be orthonormal:

$$XX^T = WDW^T$$

Where W is a matrix of eigenvectors and D is a diagonal matrix with eigenvalues λ_i in decreasing order on the diagonal.
SVD applied on X gives:

$$1) \quad X = U\Sigma V^T$$

By using the decomposed expression of X to write the covariance matrix we obtain:

$$2) \quad XX^T = (U\Sigma V^T)(V\Sigma U^T)$$

Since V is orthogonal $V^TV = 1$, which means:

$$3) \quad XX^T = U\Sigma^2 U^T$$

The correspondence between *1* and *3* can be easily seen[7][8].
In this project I decided to use SVD instead of PCA because sklearn's implementation of SVD is compatible with scipy.sparse matrices.

Scaling

The features for this problem have all very different ranges and distributions and this could negatively affect the work of the logistic classifier. If in the features space there is was one feature with range 200-10000 and another one with range 1-2, the classifier wouldn't be able to weight a 10% change in both of them the same. The solution is to scale all features so that they have similar variance and can be directly compared. Random Forests are more robust to this problem because they split the data by taking into account one feature at the time so they don't directly compare different ranges. To scale data I used sklearn's *StandardScaler* which subtracts the mean by each feature and then scales them to unit variance.

Calibration

As stated before log-loss uses the confidence with which a model assigns an instance to a class to measure the cross entropy, however some classifiers, like random forests, don't provide this type of information. The solution is to use a calibration method that fits the output of the classifier to a logistic regression (Plat-Scaling). The probability estimates are produced via this equation:

$$P(y = 1|x) = \frac{1}{1 + exp(Af(x) + B)}$$

Where *A* and *B* are two scalar parameters that are learned by the algorithm.

---

[7] http://math.stackexchange.com/a/3871
[8] http://stats.stackexchange.com/a/134283

Benchmark

Kaggle provided a sample submission which consisted in a matrix with equal probability to each class for each instance. For this submission the log loss is 2.48. However, this is a very loose benchmark because the distribution of instances is not uniform among classes. The baseline model I built to compare my result can be found in “src/models/baseline.py” and consists in a dataset in which for each class the probability level is the frequency of that class. The log loss for this model is 2.43.

III. Methodology

Data Preprocessing

The data preprocessing can be found in the “features” folder and is divided in five files: dense and sparse feature generator scripts for both *phone* and *position* and one with both implementation for *app* and *labels*.

All these files had this steps in common:

1. Both *gender_age_train* and *gender_age_test* were loaded and a new column with the row value was added. *trainrows* and *testrows* become an encoded version of *device_id.*
2. Data were processed according to the specification of each dataset.
3. Data were scaled.
4. Data were divided in train and test set by using *gender_age_train* and *gender_age_test* rows as factorized values of *device_id.*
5. Trained both models selecting the best hyperparameters
6. Further improved the model that best performed on Kaggle’s leaderboard (logistic regression)

gender_age_train.csv and *gender_age_test.csv* hold the list of the *device_id*s belonging to the test and training set. In *make_training_set.py* and *make_test_set.py* I joined them with my features and obtained the two datasets to use with my model.

Implementation

Phone specifications

The original dataset (*phone_brand_device_model*) maps for each *device_id* the brand and model in the form of strings. By using *brand* and *model* as indices I merged the data with the prices of the majority of the devices.

Given that some brands have models with the same name I concatenated brand and model in a new *model* column. All three of the features (*brand, model* and *price*) were then scaled and saved in their dense form.

In order to generate the sparse data I factorized both *brand* and *model* and used these values to create two sparse matrices. The rows are the same of *gender_age_train* (*trainrow* is a unique key that map to *device_id* in *gender_age_train*). Each row (*device_id*) is 1 only in the column corresponding to the brand or model of the device. The same approach was used to create the test set.

Location

The data provide information on the position of devices, in terms of latitude and longitude, for the majority of events. As showed in the data analysis the majority is located in China. In order to flatten the features to one location per *device_id* I decided to select the most common location for each user. The coordinates were then scaled to make the dense features.

The sparse data were built using the same method as for the phone features: factorizing latitude and longitude and building two matrices with a 1 in the most common coordinate for each user.

App ids and labels

The raw data on app ids and labels consist on the list of apps installed and used on each device and for each app the list of categories it falls into. As seen from the data analysis the most used and installed apps change from males to females. In order to account for that I counted how many times each user used/had installed each app and then divided it for the total personal number of app used/installed. This way each value is proportional to the amount of time one person spent on the app. This information was stored in two sparse matrices (one for installed apps and one for the active ones) using factorized app ids as columns.

I then used an SVD with 500 columns to create a denser version of these data to use with random forests.

Make training and test set

The training and test set were obtained by joining the training and test set datasets of the three features categories. After this operation dense data presented some *NaN* values because not all the users had the events logged correctly. The solution I implemented was to fill all the *NaNs* with the average value of that feature that, being all of them normalized to a distribution with mean 0 and unitarian variance, is 0.

Training

Logistic regression was trained using the sparse features. Cross validation was performed using *StratifiedKFold* which as previously specified is particularly indicated for skewed datasets. The metric used to evaluate the model performance was log-loss. The

hyperparameter to choose was C, I tried seven different values ranging from 10^-5 to 1. After this model turned out to be the one performing better I performed some more training controlling both the *multi_class* and *class_weight* parameters.

Random forest was trained on the dense training data. The parameters investigated were *max_depth,* ranging between 3 and 11, and *min_samples_split*, ranging between 50 and 1000. The results of this classifier needed to be calibrated using *CalibratedClassifierCV.* As per documentation of the function the calibration set needs to be disjointed from the data on which the classifier has been fitted. I used *train_test_split* to divide the training data into train and calibration set. After the training the model was calibrated and saved.

For both classifiers the classes were encoded to integers between 1 and 12 prior to training.

Submission

Since Kaggle doesn't provide the labels for the test set the only way to check the score is to submit the predictions to the leatherboard. In *predict_model* both models were loaded and run to produce the predictions. In order for the submission file to be accepted by Kaggle I had to add the *device_id* value for each entry, de-encode the classes and put them as the name of the columns.

Refinement

As previously stated the benchmark defined for this task was a model that as probability for each class gave its frequency in the training set. The log loss of this model was 2.43.

I trained both models with the basic hyperparameter selection described above, after the best configuration was obtained for both of them I run the test set in order to produce a submission. Of the two models the best performing one turned out to be logistic regression with a log-loss score on Kaggle of 2.36979 while random forest scored 2.38610.

I further refined the logistic regression model by tuning the *multi_class* and *solver* parameters. The combination that resulted in the best result was with *multinomial* class and *newton-cg* solver. Once submitted, the output predicted with this model scored 2.36923, marginally better than the basic model.

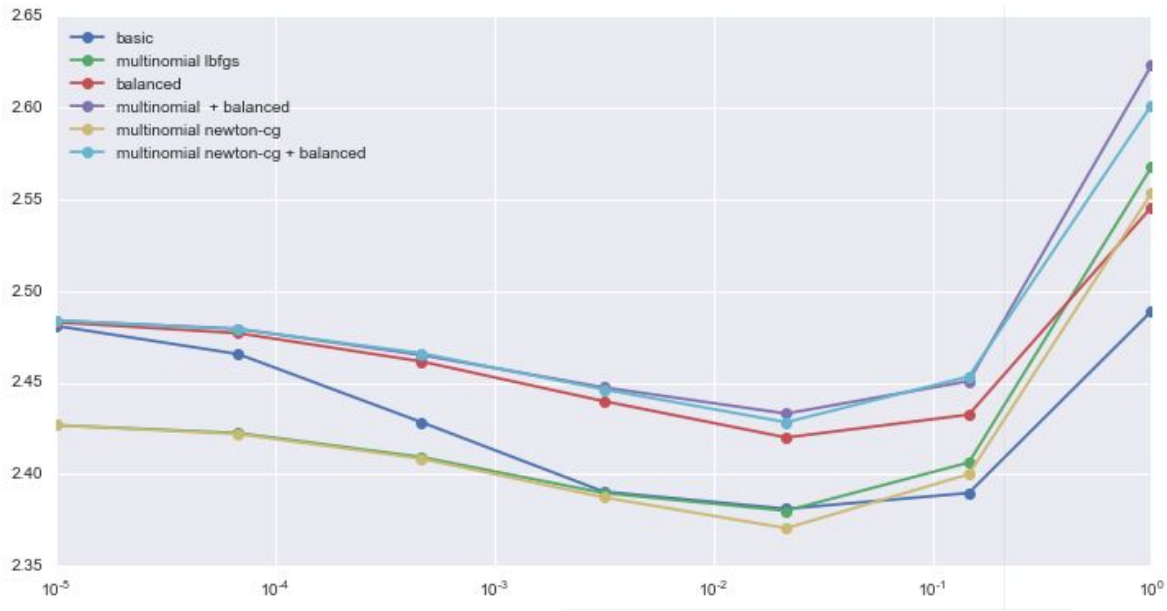


Figure 17: log loss of logistic regression trained on different values of multi_cllass and solver.

IV. Results

Model Evaluation and Validation

The model selected performs consistently better than both Kaggle's benchmark and the one I defined. Below there is the summary of the hyperparameters that define the logistic classifier:

- Inverse of regularization strength, *C* = 0.03
- *Multi_class = multinomial*
- *Solver = newton-cg*

In order to prove the strength of the model, I analyzed its performance with both F1-score and confusion matrix and compared it against my baseline. In *src/visualization/validation.py* I first divided the labelled data in training and development set, trained the best classifier on the training data alone and then performed the validation on the development set.

F1 Score

The output of logistic regression is a vector of probabilities so it cannot immediately be used to measure the f1 score. After predicting the output probabilities for the development set I selected the column with the highest probability for each row. This was then used as the predicted class for that entry. The same procedure was applied to the baseline model. The logistic model scored ≈0.12 while the benchmark score ≈0.03.

Confusion matrix

To further investigate where the mistakes were made I plotted the confusion matrix for my model. As can be seen from Figure 18 the large part of the mistakes is made because the classifier assumes that the majority of data belong to the group M23-26.

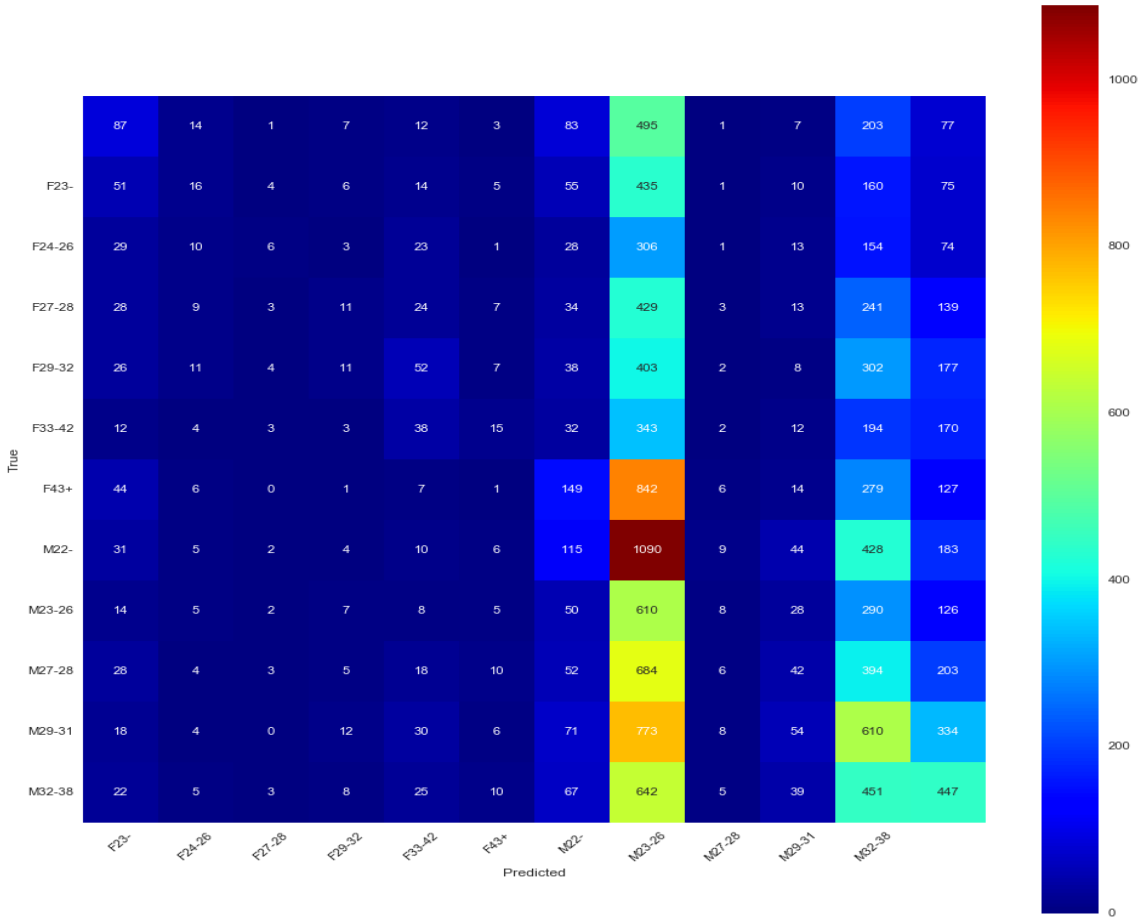


Figure 18: confusion matrix

Justification

The insights and statistical tests provided in the previous section show that although the model performs better than the benchmark it is not able to solve the problem given. As can be seen in Figure 18 the model learnt to predict more often the group that is more represented than anyone else: males between 23 and 26 years old. This is a predictable behaviour and unfortunately a very difficult one to correct. The option of resampling the population in order to have classes more balanced between each other is a serious mistake that would create a model not able of generalize the test set.

V. Conclusion

Free-Form Visualization

Please refer to Figure 18 in section *Model Evaluation and Validation.*

Reflection

The overall process used to solve this problem can be summarized as follows:

1. The problem and data were found on Kaggle
2. Data were downloaded and analyzed to extract useful insights
3. Created new features
4. The benchmark was defined
5. Two models were trained: logistic classifier and random forests
6. Given the better performance logistic classifier was chosen as final model, performed further tuning of its parameters
7. Final evaluation of the model

I found points 2 and 3 to be the most interesting and the most challenging at the same time. I enjoyed exploring the dataset and using the results to infer information to help me decide which models and metrics to use and what features to include. Features engineering has been very challenging because of the need to express the features in a way that was meaningful both in the sparse and dense form. Given the strong skewness of the data and the large number of classes I think logistic regression is not doing a bad job, however I was expecting a much better result from random forest. Overall I think the models I chose were not the best to generalize this type of problems. I think that the reasons of the poor performance of both models can be found in the unbalance of representation between classes and that features are linked by non-linear relationships that neither logistic classifier or random forests can pick up.

Improvement

The bigger limitation of my model is that it is not able to pick up non-linear relationships between the features. An example is that latitude and longitude are coordinates of a polar system which means that their information cannot be represented by a linear space. A solution to this problem would be to implement a classifier using neural networks(NN). NN consist of multiple layers of neural units highly interconnected. Each neuron can have an enforcing or inhibitory effect on the signal passed. The nonlinear structure of NN makes them ideal to approximate the same type of problems.

The reason why I didn't implement this solution in the first place is that I do not have any theoretical knowledge or experience with NN. Plus, for the capstone project I wanted to use the techniques that I learned in the Nanodegree.