Home assignment

Q1: The solution workflow:

TL;DR

In this short research, I have tried to predict whether users will subscribe, given their first 24h usage data from the application installation. I have built 14 features to describe the user's behavior (and more that were not used). This problem is of imbalance classification - the minority class is about 0.76% of the total users (after filtering). Moreover, we have a small sample of the minority class (45 samples), which means that the model doesn't have access to the full distribution of the minority population. In addition, for the subscriber population, I observed a big variation in the time of the subscription, where more than 50% subscribed after 20 days. My approach to the imbalance problem was adding weight to the loss function of an XGboost model, to balance the minority class and calibrate the classification threshold according to the geometric mean of Sensitivity and Specificity (TPR and TNR), but keeping in mind that in a real-life situation, **the threshold will be set according to the acceptable FPR from a business point of view.** I have evaluated the model via F1, precision and recall scores, confusion matrix, ROC curve, FPR vs FNR curves, and precision-recall curve. From the ROC and the precision-recall curves, I could learn that the model classification is slightly better than random. However, the results are very poor and not stable, including high recall with very low precision. **The threshold did not balance between the TNR and the TPR on the test set**. The precision and recall scores were obtained again, by adding a factor of 100 to the threshold, for better precision and for an argument's sake.

The data :

This is an imbalance classification problem with a low amount of data on the subscribers:

When we look at the data, we see an imbalance between the subscribers and the nonsubscribers, where only about 0.76% of the users are subscribers. The data includes 5929 users (after some filtering), and a total of 45 subscribers. For the subscribers, it seems that we might not have enough data to describe their distribution, which might impair the ability of the model to learn their patterns.

Further complications with the subscriber's population :

Below is the statistic description table, for the time between the installation of the app and the time the user subscribes, in days.

We can see that more than 50% of the subscribers, subscribed after more **than 20 days and the std is 17 days.**

However, we are trying to classify them during the first 24h of their usage. During this time, it is reasonable to assume that users who subscribe after 2 days or a week i.e **early subscribers** will have different behavior than users who subscribes after a month i.e **late subscribers**. Moreover, late subscribers, during their first 24h of usage, might have very similar patterns as the population of the non-subscribers, which in turn might affect the ability of the model to classify them.

	time_from_install_to_subscribe
count	45.000000
mean	23.097886
std	17.303693
min	2.019444
25%	7.800000
50%	20.815972
75%	34.334722
max	58.738889

Filtering:

I have conducted the following basic filtering:

- Removed data that include the usage of more than 24 hours after installation.
- Removed data for users that their installation time was after their time stamp usage - clearly some bug, probably due to an application update.

How I have divided the data for test and validation:

30 % of the data was saved for final testing and reporting results -we call this the test set, and the remaining are referred to as the training set.

From the training set, 30 % were taken for validation - feature selection, hyper-parameter tuning etc. This set is named the validation set, and the left 70% is the validation training.

Here we can see again the problem of the small sample arising, we have a total of 45 subscribers, yet for the training set we are left with 32, and for validation training, **we have only 22.**

Features:

Methodology in selecting features:

The main goal was to create an aggregate data table, where each row, represents a user.

While designing a feature, I had the following in mind: how can this feature distinguish the subscriber population from the none subscribers? how can I describe a re-current user?

Therefore for each feature I have designed, I have checked its statistics on the two populations (subscribers vs nonsubscribers) and searched for a significant difference in the statistics. This method was applied to most of the features I have designed.

I observed the overall correlation of the features, the statistics descriptions, and the boxplots of the two class populations and the overall population.

Example of how to choose a feature:

In the table below, we can see a statistics comparison between the two populations, for the feature - total usage time of the user, in hours. We can see that the median and average values of the subscriber's population are higher than those of the none

subscribers, which means that this feature might help the model to distinguish between subscribers and non-subscribers.

	total_usage_time_subscriber	total_usage_time_no_subscriber
count	45.000000	5884.000000
mean	0.318082	0.193101
std	0.653162	0.685121
min	0.001263	0.000416
25%	0.059133	0.027872
50%	0.107620	0.077588
75%	0.346918	0.177299
max	4.180064	21.386602

The features :

Here are the features I have used and their description:

- **total_usage_time** - the sum over the column ‘**usage_duration’**
- **num_unique_features** - the number of unique features the user has used. This was derived from the column ‘**feature_name’**
- **num_of_unique_sessions** - the number of unique sessions the user had. This was derived from the column ‘**app_session_id’.**
- **accepted_mean** - the mean over the ‘**accepted’** column
- **actions_count** - count the total actions of the user, this was derived from the ‘**accepted’** column.
- **gdp_value -** the GDP per capita of the user’s country. This was derived by mapping the country of the user and GDP data I have downloaded from here. The values were binned between 0-3. Higher value- higher GDP.
- **delta_s_between_sessions_mean** - the mean time between the sessions of the user. This was derived from the column ‘**app_session_id’**

- **delta_s_between_sessions_median** - the median time between the sessions of the user. This was derived from the column ‘**app_session_id’**
- **delta_s_between_sessions_std** - the std time between the sessions of the user. This was derived from the column ‘**app_session_id’**
- The features: **'device_ipad', 'device_iphone', ‘device_ipod' -** the user’s device. Derived from ‘**device**’ column.
- **action_per_session** - the mean number of actions per session. This was derived from the columns: ‘**app_session_id**’ and ‘**accepted**’.
- **normalize_usage_time :** this is a feature I have engineered from total_usage_time, num_of_unique_sessions, and action_per_session, and it is a multiplication of the above features:
  - **normalize_usage_time =**
    - →total_usage_timex X num_of_unique_sessions X action_per_session

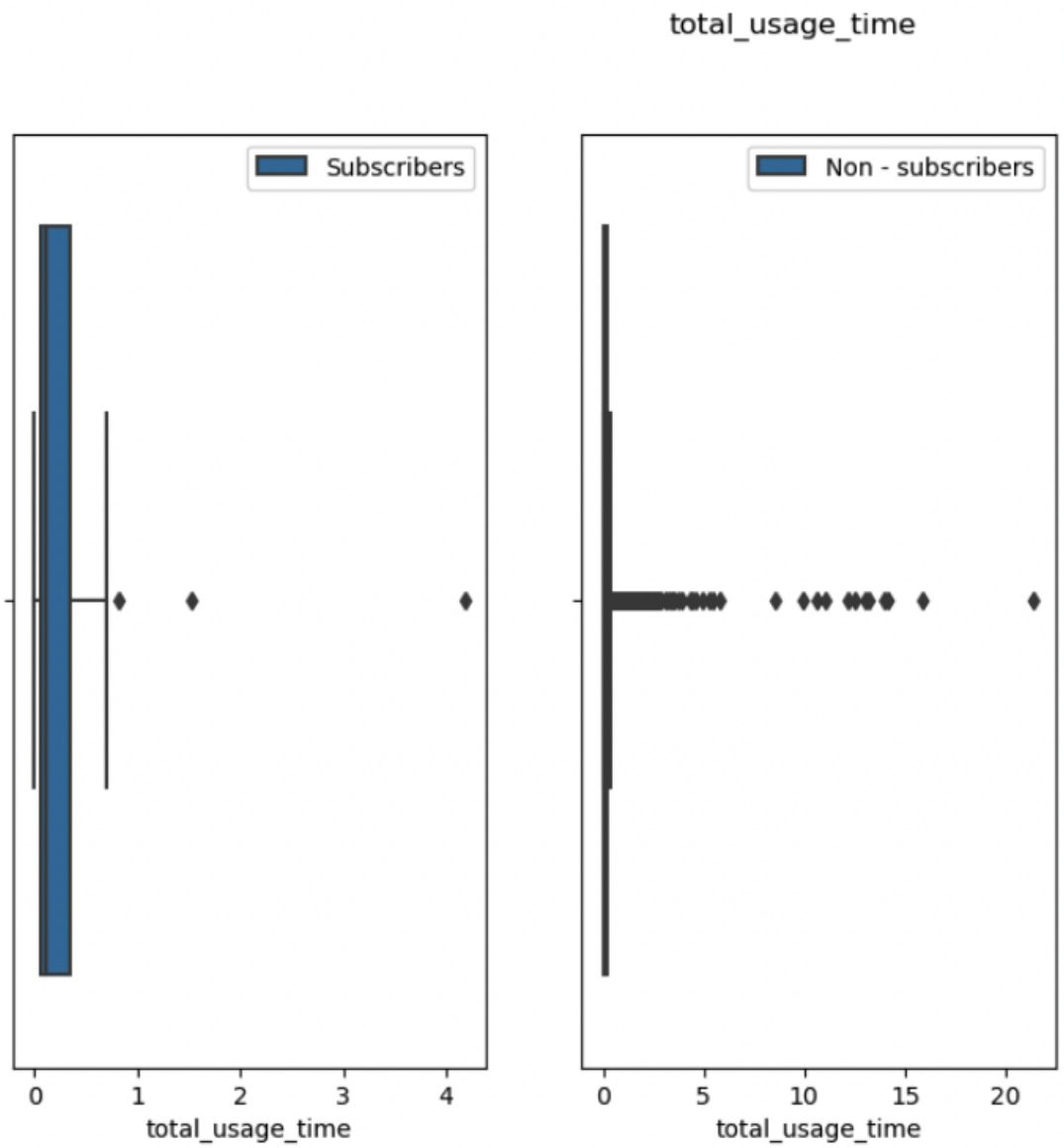
  The idea was to give more meaning to the usage time. I had in mind that some users might have left the app open without using it, therefore I multiplied the usage time with the features that indicate engagement with the app.

Why i did not clean outliers from the features?

1. Because I have tried cleaning some of the outliers, but it results in poor results.
2. I think that in this case, some of the outliers, actually help the model to classify correctly.

   For example, let's look at the boxplot of the subscribes and non-subscriber population, for the total usage time in hours :

   For the none subscribers, we have outliers - users who used the app for a total of 20 hours !? those users probably forgot the app open. However, this behavior is not observed for the subscriber's population, meaning that the model can learn that very high values, are indicated on a nonsubscriber.



The model :

For this problem, I have used my favorite machine learning model - XGBoost.

My approach to handling the imbalance in the data was as follows:

- Add weight to the loss function of the model, i.e to increase the cost of a wrong classification for the minority class.
- Calibrate the threshold for positive classification.

Adding weights :

The parameter that controls the weight of the minority class is scale_pos_weight. Using the training data and Stratified KFold, I have chosen the parameter that gave the best F1. I choose F1 to balance between precision and recall.

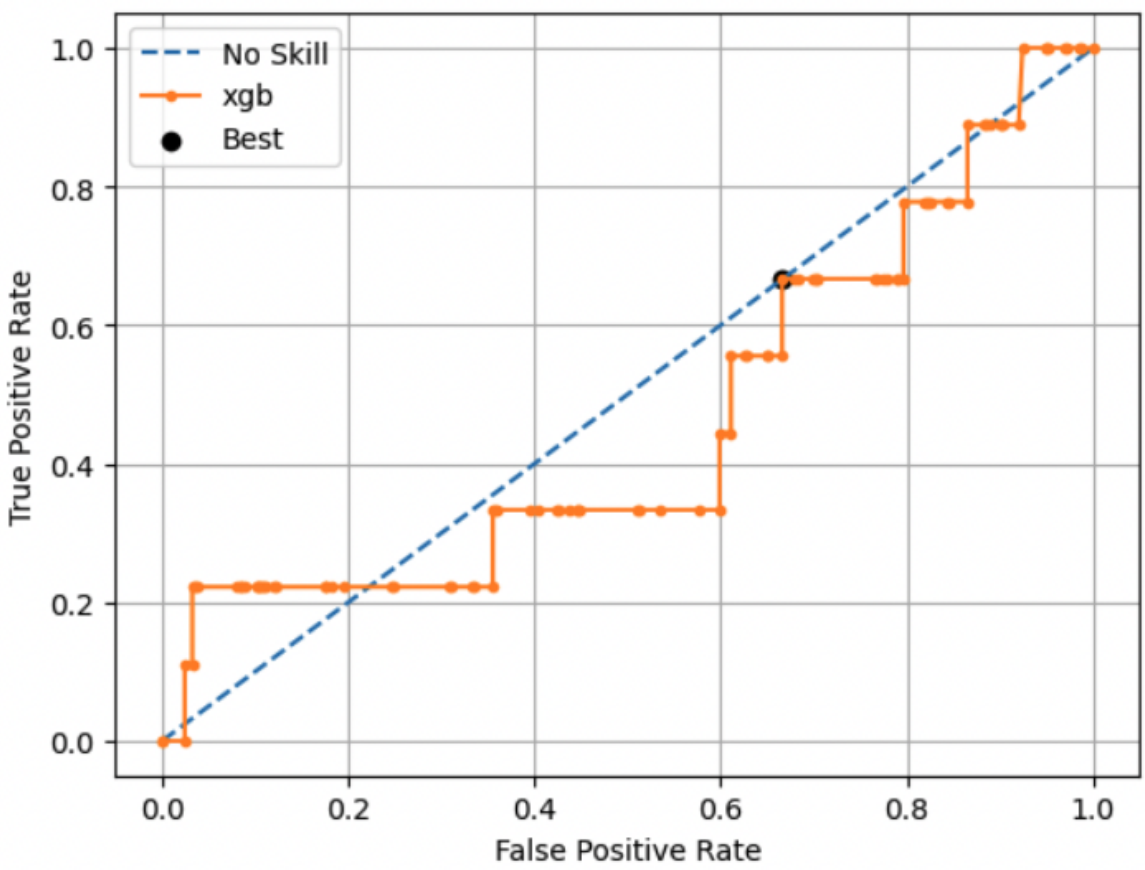
Finding the threshold :

Using the data of the training for validation, I have trained a model (using the scale_pos_weight I found earlier). Then, I created a set of probability predictions, from the validation set, for generating a ROC curve and FPR vs FNR plot.

From the ROC curve, I could learn the following :

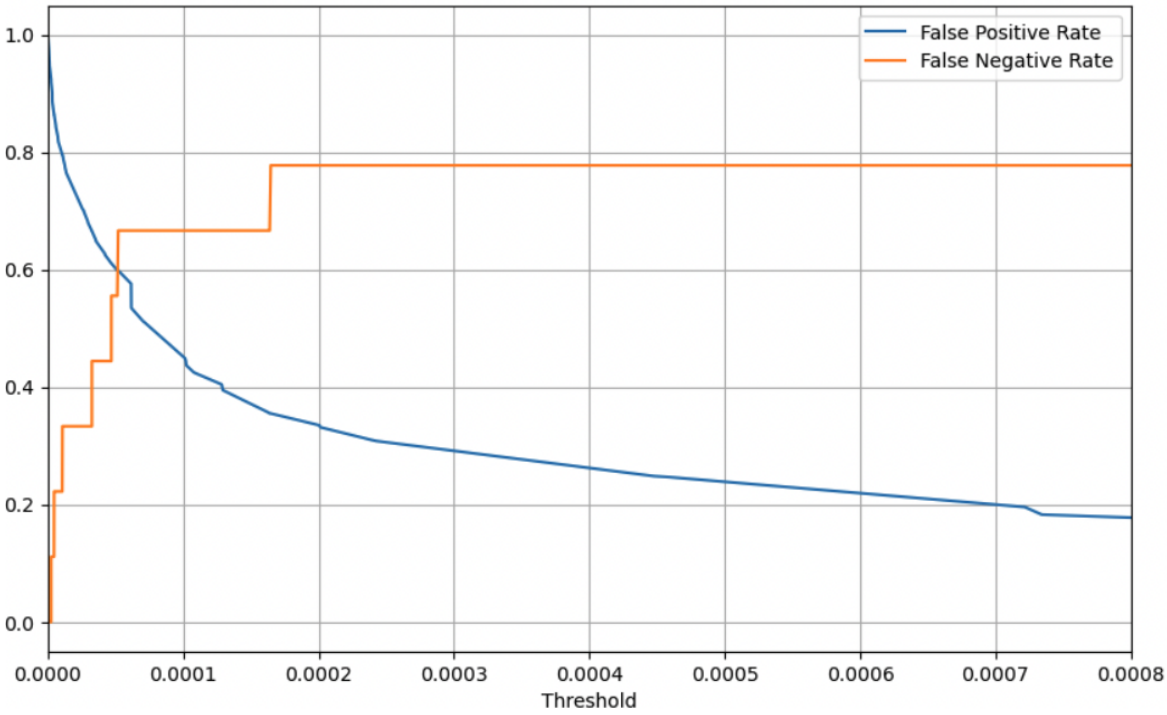
1. What is the best threshold for max the geometric mean of the sensitivity and specificity,
2. Is my current model better than a random one?

In the figure below we can see the ROC curve. In orange are the results of a model train on the validation set, in blue - is the behavior of a random model, and the black dot - is where the threshold gave the max geometric mean of the sensitivity and the specificity (will be referred to as g-mean from now on). The therhold was `0.000033.` Overall, it seems that we did not do quite well, the model curve is very close to a random model. However we should bare in mind, that probably, in a real-life situation, the relevant curve area is only **for low values of FPR**. **If we zoom, where the FPR< 0.2, our model is better than random.**



I have decided to take the threshold for max g-mean, to balance between the recall and the specificity, **but in real life, the threshold should be derived from a business point of view**, according to the FPR - FNR tradeoff, as described in the below figure - in the x-axis is the threshold, the orange curve is the FNR and the blue curve is the FPR.

For example, if it is not very costly to approach many users, with a high value of FPR, then we can allow a lower threshold.



Evaluation

When evaluating the model, either for validation or for the final test, I used the following metrics:

- ROC curve - From this curve (**when limited to low FPR**) we can learn if our model classifies better than a random model. The metric in this plot is the tradeoff between TPR and FPR.
- Precision recall curve - From this curve, we can learn about our model performance, and we can compare it to a random one. This curve is complementary to the ROC curve since in this curve, the focus is on the tradeoff between **precision and recall.**

- Precision, Recall, and F1 score - these scores allow us the understand the model's classification abilities in one number.
- Confusion matrix - this matrix summarizes the above.

Results

Here are the final results, for the test set:

Summarize table of F1, precision and recall scores for train, test, and random model :

We can see that our recall score for the test (’val’) is very high - 78%, but accordingly, the precision is very low 0.8%.

Notice that I added the results of a no-skill model for comparison. Also, Notice that these results are dependent on the threshold we set. I set the threshold to be quite low, for maximizing the g mean score. From the low precision and high recall we can learn that given the threshold we set, **the model is biased to classify a user as a subscriber.**

	precision_score	recall_score	f1_score
train	0.009755	1.000000	0.019321
val	0.008240	0.785714	0.016308
train_no_skill	0.007470	0.500000	0.014720
val_no_skill	0.007870	0.500000	0.015495

The confusion matrix for the test set :

We can see that we have a very high FPR, which allows us to have a high TPR. Of course, a 75% FPR is not practical, and for real-life situations, I would choose a higher threshold (according to the validation set). For this model, if the threshold will rise, **the TPR will decrease dramatically. We can learn that the model performance is very poor, given an FPR of 75%, we would expect a higher TPR.**

Moreover, according to the validation set, we have chosen a threshold, that was supposed to balance between the TNR and the TPR, yet we can clearly

see that that is not the case, the TPR is equal to 78% whereas the TNR is equal to 25 %.

	Pred 0(preds as N0 Subscriber)_val	Pred 1(pred as Subscriber)_val
True 0(No -Subscriber)	TN = 441 (TNR = 24.99%)	FP = 1324 (FPR = 75.01%)
True 1(Subscriber)	FN = 3 (FNR = 21.43%)	TP = 11 (TPR = 78.57%)

The above results, given a higher threshold :

Assuming that we do not wish to pay such a high price for the high recall, we can raise the threshold (of course this process of defining a new threshold, should be done via the validation set). Here are the results for rising the threshold by a factor of 100:

We can see, that now the model is not biased to classify a user as a subscriber. Also, now, the precision is at 2%, rather than 0.8 %. Of course, there are no free lunches, the recall has dropped to 35%.

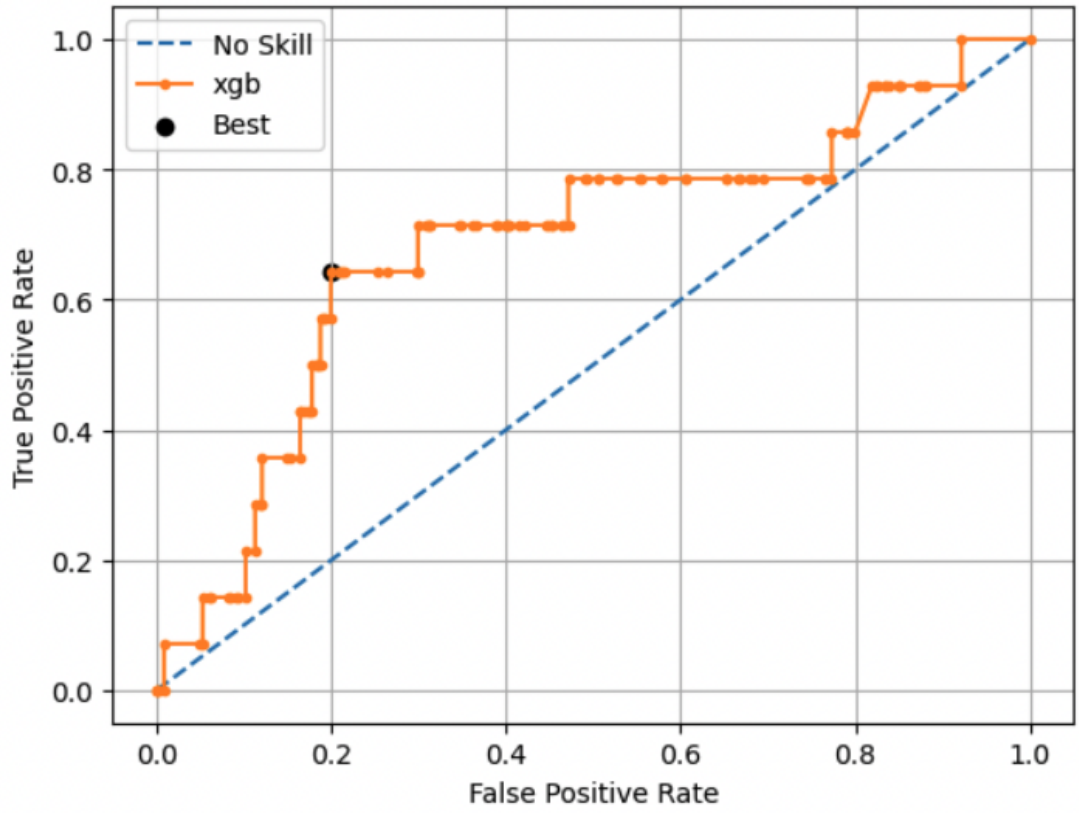
	Pred 0(preds as N0 Subscriber)_val	Pred 1(pred as Subscriber)_val
True 0(No -Subscriber)	TN = 1536 (TNR = 87.03%)	FP = 229 (FPR = 12.97%)
True 1(Subscriber)	FN = 9 (FNR = 64.29%)	TP = 5 (TPR = 35.71%)

	precision_score	recall_score	f1_score
train	0.062124	1.000000	0.116981
val	0.021368	0.357143	0.040323
train_no_skill	0.007470	0.500000	0.014720
val_no_skill	0.007870	0.500000	0.015495

The above results are only to be making a point - the precision-recall scores and the confusion matrix results are dependent on the threshold, **which in turn should be defined under the business constraints**. Given non, I have chosen to try to max the g mean, and since we got such a weak model, in order to do that, the model became biased for the minority class, resulting in high recall + very high FPR.

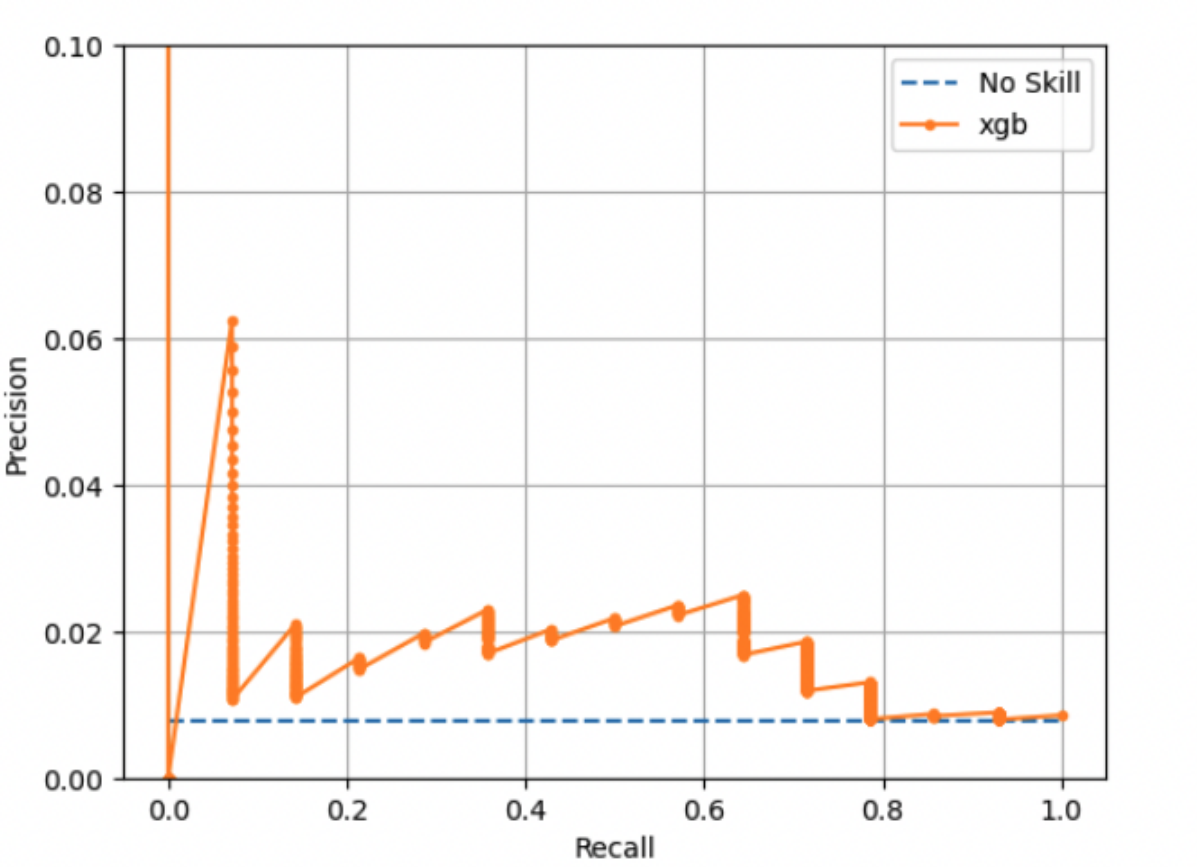
The ROC curve:

This result, and the precision-recall curve as well, are not dependent on a specific threshold, and therefore they can teach us something more general about our model. We can learn that our model is better than random, **but not significantly**. Also, as I have mentioned earlier, in the relevant area - where we zoom for low values of FPR, our model performance is better than random. The point where the g mean is max is marked in black.



The precision-recall curve:

We can learn that our model is better than random. Yet we can see that our precision is limited to very small values for various thresholds.



Conclusions

From the above test results, we can see that we have managed to train a model that is better than a random one, but not very far from that. The model precision is very limited and is not practical for business use.

In addition, the threshold we have set did not balance between the TPR and TNR in the test set.

There are serval reasons that I believe cause the model to have such poor results:

- **small data set for the minority class** - I had a total of 45 subscribers, this kind of sample number is likely not enough to catch the distribution behavior. Moreover, since I used a validation set and test, the model had access to an even lower number of subscribers. This results in significant instability. For example, I have noticed that changing the ratio of the train validation and test set, changes the overall results. **Also, I believe that the small sample data of the subscriber population in the validation data resulted in a poor and not generalized choice of a threshold for balancing between TPR and TNR.**
- High Variance of the time of subscription within the subscriber's group - we have late subscribers - users that subscribe after more than 20 days and we have

early subscribers - users who subscribe within a week. It is reasonable to assume that this population will have different patterns. It is very probable that the late subscribers, or at least a significant part of them, behave very similarly to the non-subscriber population, during the first 24h from the application installment.

- Time 😝: if I had more time I would :
  - Create a pipeline for feature selection. Maybe a small set of features will improve the model.
  - Create a pipeline for hyper-parameter tuning for XGB.
  - Try more models (some simpler ones)
  - More features: I would try to create more features, focusing on trying to describe user engagement and recurrent usages. Also, I would design features relative to the overall median.
  - Try cleaning the data again with another method, and search for problems - for example, I found users whose timestep was smaller than the installation time.
  - Try downsampling the majority class - although I don't believe this is a good approach, it is worth trying.
  - Try to use 3 classes(Long shot): early subscribers, late subscribers, and none subscribers. This will probably be beneficial given more data
  - I would ask what is the acceptable FPR and would create an AUC scoring for the ROC curve**, but for a small segment of FPR - [0, acceptable_FPR], and find a threshold and weight for the loss function according to this value.**
  - Ask for more data, if possible.

Q2 :

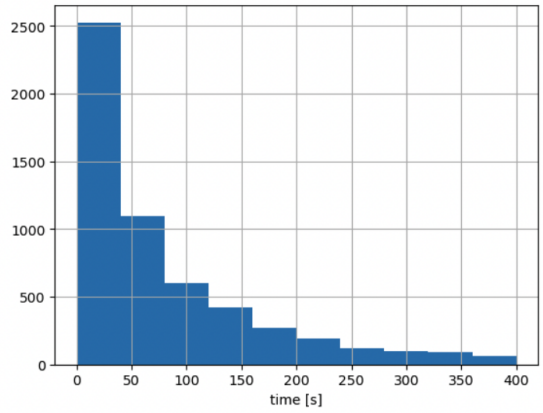
Let's look at the distribution of the first session time and the relevant statistics.

We will limit ourselves to looking up to 400s.

From the below histograms and statistical summary, we can learn, that if you are future to subscribe, you will use the application more time in the first session than a nonsubscriber.

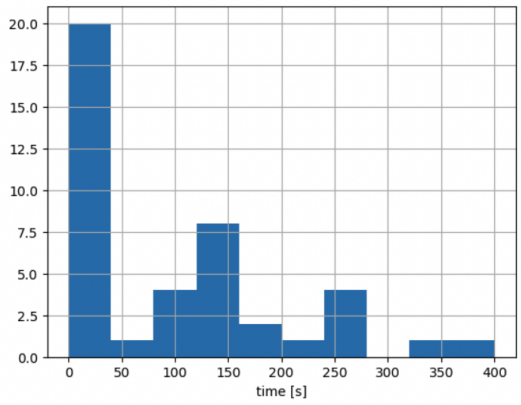
If we want to offer an incentive, we have a clear tradeoff - if we offer a discount early in the session - i.e targeting most of the users that are most likely to not subscribe, we would also target most of the user's that will subscribe at any case.

nonsubscribers:



	time_firtst_session
count	5884.000000
mean	285.462847
std	2293.708258
min	0.956793
25%	17.420675
50%	52.477400
75%	137.055975
max	76731.679000

subscribers:



	time_firtst_session
count	45.000000
mean	324.179499
std	1403.619827
min	4.547060
25%	28.217700
50%	88.440900
75%	164.120280
max	9500.660400

Machine learning approach :

We can do the following - train a classification model, based on features from the start of the session to time X, during the session and features that we got on the user, before the session. We can update the model every x seconds, during the session.

Given this data, we will build a classification model, where its output **is the probability score that the user will subscribe.**

Why not use the probabilities :

Assuming we will have a similar case such as in question 1, of imbalance data, it is likely that the output probability of the model will not be calibrated, i.e it will have a strong overconfident in predicting a nonsubscriber.

Given the scores/probabilities, we can rank the users into percentile such that the lower the percentile, the higher the probability that the user will subscribe.

Important side note - we can try to calibrate the probability of the model, i.e using past data and model past prediction and to build a calibration function and apply it to new predictions.

Now we can use the model, to do a/b testing - from which percentile/probability (if we manage to calibrate) should we offer incentive. For example, maybe offering to a percentile higher than the top 30 will give the most conversion.

Also, keep in mind, there will probably be different percentile thresholds for different times. For example, as the time of the session grows, the probability for the user to be a subscriber increased, therefore, we will require a higher percentile threshold. This should be decided empirically.

Additional features :

- D**ata on past subscriptions on other apps: for example, if the user has downloaded recently similar apps - maybe the user is comparing competing apps.**
- What the user is doing during the session - if he shared a figure / downloaded it.
- Age, work, and gender.
- Which apps were uninstalled recently?

Evaluate the model and implementation in production :

Offline:

- Evaluate the model, similarly as I have described in question 1.
- Try to avoid ranking - see if we can get a stable calibration model for probability.

Online:

I would try a/b testing for different user's percentiles (reminder, higher percentile means lower probability to be a subscriber) threshold and different percentages of discount.

The hypothesis is - to offer a discount to percentile > X, and after time y, can rise the conversion by z, where z will be deduced from the experiment.

Given the conversion rate, we will check the main KPI - how much profit was increased, and what is the confidence interval, and how much increase in profit is worth the change in our approach.

Some more elaboration for the a/b testing:

- Define a main variable we want to alter - i.e the percentile threshold of when we offer incentive/discount.
- Define our goal: increase conversion rate +monthly profit etc.
- Create controls and challengers -
    - Given that a user is classified above the percentile threshold, we randomly decide if to offer him/her the incentive.
    - The controls are those that were classified but we did not offer them the incentive.
    - The challengers - are those that were classified and we offer them the incentive
- We will need to let the test run long enough in order to get enough sample data for the comparison, in order to have statistically significant results.
- Calculate the confidence interval of our results. The desired value should be determined from a business point of view.

Q3:

1. Yes. Definitely, worth the try - anything that can help the model to better distinguish between the subscriber population and the nonsubscriber population is welcome. Some clear advantages:
    a. Access to features not that are not directly related to the app
    b. automatic - creating a model, that does the embedding automatically might have an advantage over human-made features, or at least it could be complementary to it.
    c. Help the model by taking a higher dimensional vector into a low dimension.

    We should take into account that we might lose explainability.
2. I would try as a start to use a very basic embedding technique and see if the model improves. If it does, I will implement state-of-the-art embedding techniques such as those described in the article. This is, of course not a full answer, I need to do thorough research on embedding techniques.