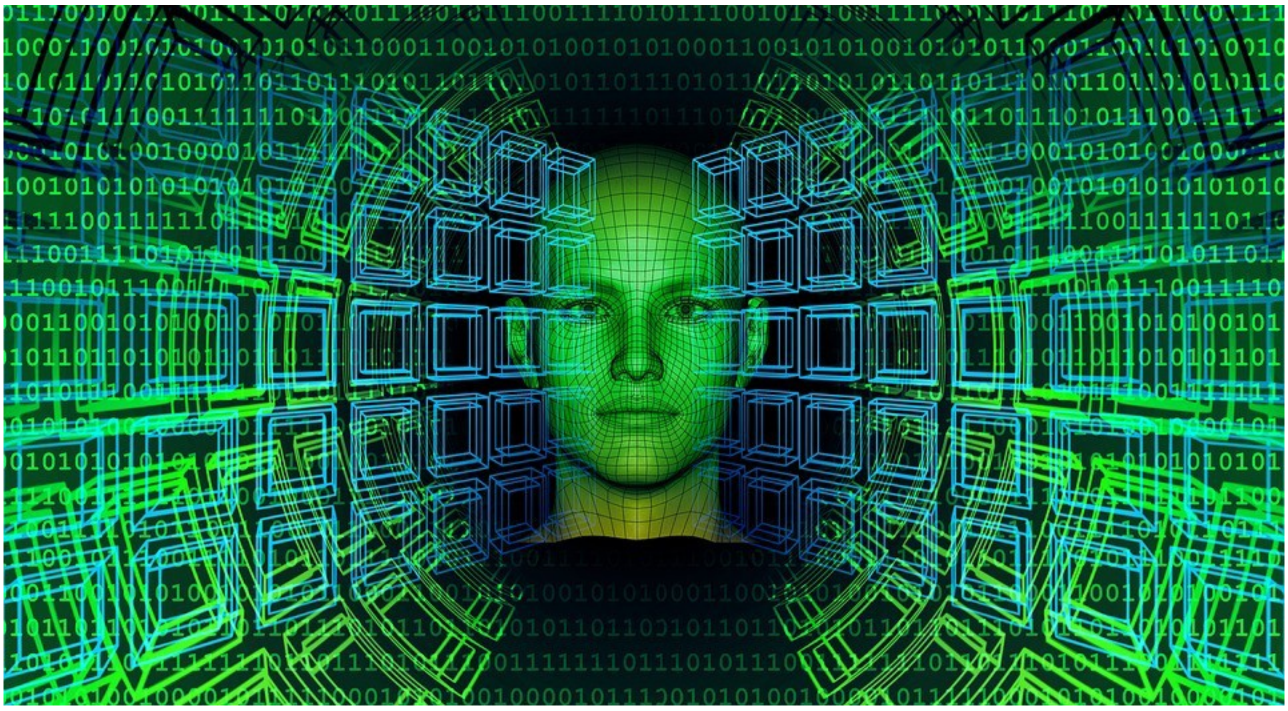
How to “farm” Kaggle in the right way

This article describes the advice and approaches on how to effectively use Kaggle as a competition platform to improve practitioner skills in Data Science with maximum efficiency and profitability.

Alex Kruegger
Follow
Feb 12 · 33 min read

farm (farming) — gaming tactic where a player performs repetitive actions to gain experience, points or some form of in-game currency.



Description

These methods helped me with getting a Kaggle Competition Master title in six months just taking three competitions in solo mode. They also helped me while I climbed to the top-200 world ranking on Kaggle (at the time of writing the original article 2018–10–18). I hope this answers the question of why I took the liberty to write an article of this kind.

Introduction

Recently (February 11th, 2019) a new session of an amazing course on Machine Learning has been launched. This course is in English, it was created with input from a bunch of Kaggle Grandmasters and I highly recommend it as a starting course to anyone who wants to start his/her career in Data Science. As per usual, after the completion of this course (or any course for that matter) the question among students arose: where and how can we obtain the practical experience to consolidate newly acquired theoretical knowledge. If you ask this question on any relevant forum, the answer is likely to be “Kaggle.” OK, Kaggle is nice, but where to start and how to use this platform for the improvement of practical skills most efficiently? In this article, I will try to give answers to these questions based on my own experience, and describe how to avoid general mistakes in the field of competitive Data Science and speed up the process of boosting your skills while having fun in the process.

A few words about the course from its creators:

mlcourse.ai is one of the major activities of the OpenDataScience community. Yury Kashnitskiy (yorko) *& Co. (~60 more people)* show that in principle you don’t need a University to acquire modern technical skills; moreover, you can do so entirely for free. *The main idea of the course is an optimal combination of theory and practice. On the one hand, the presentation of fundamental concepts involves some math, on the other hand — a lot of assignments, Inclass Kaggle competitions and projects will, with some effort from your side, improve your machine learning skills. Not to mention the competitive nature of the course — interactive student rating makes it fun to participate and motivates to endure till the end. Also, the course takes place in a truly vibrant community.*

There are a couple of Kaggle In Class competitions held during the course. Both are very interesting and required feature engineering. The first one is on user identification through the sequence of visited sites. The second one tries to predict the popularity of Medium articles. In a few assignments, you need to do your best to beat baselines in these competitions.

Paying tribute to the course and its creators, we continue our story…

I remember myself a year and a half ago, when I passed the first version of ML course from Andrew Ng, completed MIPT specialization, read a mountain of books… Needless to say, I had a lot of theoretical knowledge, but when I tried to solve a basic task from real life, I hit the wall. I didn’t know where to start. I understood the way how to solve the problem, understood which algorithms to use, but it was very hard to write the code, having to google tips, looking at sklearn help, etc constantly. Why? Because I didn’t have the working pipelines for such a task as well as practical experience.

It’s not going to work that way, I thought and went on to Kaggle. Starting out from the ranking competition was scary, and the first goal was Getting Started competition “House Prices: Advanced Regression Techniques”. While working on this competition the approach of efficient kaggle farming was formed, which is described in this article.

The approaches described in this article are not unique, there is no “know-how”, all the methods and techniques are well-known and straightforward, but this does not lessen their effectiveness. These methods helped me with getting a Kaggle Competition Master title in six months just taking three competitions in solo mode. They also helped me while I climbed to the top-200 world ranking on Kaggle (at the time of writing the original article 2018–10–18). I hope this answers the question of why I took the liberty to write an article of this kind.

In a nutshell, what is Kaggle?

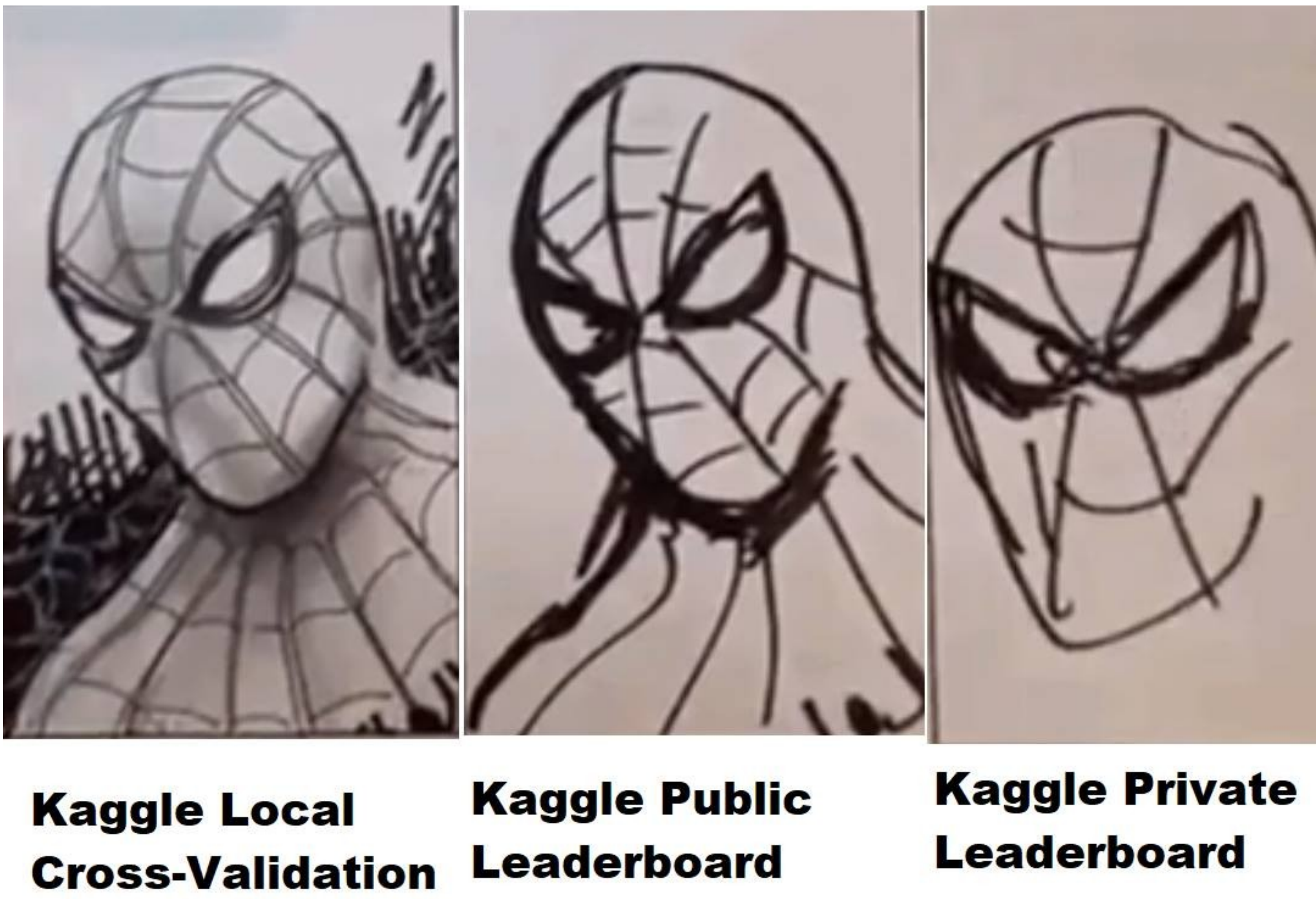
Kaggle is one of the most famous platforms to host competitions for Data Science. In every competition, the sponsor hosts the real task, provides a description of the task, the data for this task, the metric used to evaluate the solution and also sets deadlines and prizes. Participants are usually given from 3 to 5 attempts a day to “submit” a solution for this task.

The data is divided into a training set and a test set. You are given the value of a target variable for the training part, but not for the testing part. The participants should create a model which, when trained on the training part of the data, achieve the maximum result (according to the chosen metric) on the test set.

Each participant makes a prediction for a test set and sends the result on Kaggle, then the robot (which knows the target variable for the test set) evaluates the received result, scores it and displays it on the leaderboard.

But it’s not so simple, test data, in turn, is divided in a certain proportion for the public and private part. During the competition, the submitted solution is evaluated according to the established metric on the public part of the test set, and the result is put to the “public” leaderboard — in which participants can evaluate the quality of their models while competition is still going. The final solutions (usually two of the participant’s choice) are evaluated on the private part of the test data, and the result goes to the private leaderboard, which only becomes visible after the competition ends and according to which, in fact, the final ratings are evaluated and prizes and medals are distributed.

Thus, during the competition, the participants only have information on how their models behaved (a result or score it produced) on the public part of the test data. If in the case of a spherical chicken in a vacuum, private part of the data has the same distribution and statistics as the public then everything is likely to be fine. If it’s not then the model, which proved to be good for the public part of the test set may fail on the private part. It’s called “overfitting on a public board”. This leads to the “leaderboard flight” when people from 10th place on the public leaderboard crash down to 1000–2000 position on the private part since their chosen model was overfitted and was unable to give the required accuracy on unseen data.


Kaggle Local Cross-Validation
Kaggle Public Leaderboard
Kaggle Private Leaderboard

How to avoid it? First of all, you need to build a proper validation scheme, creating which is the first lesson in almost all courses of Data Science. Why is it so important? Well, if your model cannot give a correct prediction on a data set which it had never seen before then, it wouldn't matter whatever fancy technique you use or how complex your neural network was. You cannot release such a model in production because the results for the unseen data would be worthless.

For each competition, Kaggle admins create its own separate page with sections for data, timelines, description of the metric and — the most valuable parts for us — the forum and the kernels.

The forum is a standard forum where people can discuss and share ideas. But the kernel is much more interesting. In fact, it is the opportunity to run your code that has direct access to the competition data in the Kaggle cloud (similar to AWS, GCE, etc.) For each kernel limited computational resources are provided, so if the data set is not too big you can work with it directly from the browser on the Kaggle website. You can write the code, execute it, debug, and of course submit results. Two years ago, Kaggle was acquired by Google, so it is not surprising that "under the hood" this functionality uses Google Cloud Engine.

Moreover, there were several competitions (Mercari for example) where you could work with data through the kernel only. It is an exciting format, it compensates the differences in hardware among the participants and makes you use your brain on the subject of code optimization and approaches, as kernels have a hard limit on the resources at the time (4 cores / 16 GB RAM / 60 minutes run-time / 1 GB scratch disk space and output). Working on this competition, the author learned more about the optimization of neural networks, than from any theoretical course. Got a score a bit lower than the gold tier, finished 23rd in solo mode, but got lots of experience and joy…

I am glad to take this opportunity once again to say Thanks to my colleagues from the ods.ai — Arthur Stsepanenka (arthur), Konstantin Lopuhin (kostia), Sergei Fironov (sergeif) for the advice and support in this competition. In general, there were many interesting moments, Konstantin Lopuhin (kostia), who took the first place in this competition together with Paweł Jankiewicz, afterward did the post that was called "reference humiliation in 75 rows", they published the 75 lines kernel which gave the result for the golden leaderboard zone. You really have to see it!

Okay, let's return to the competitions. People write code and publish kernels with solutions, interesting ideas, etc. In every competition one or two beautiful EDA (exploratory data analysis) kernels emerge with a detailed description of the dataset, attributes statistics, characteristics, etc, usually in a few weeks from the start. And a couple of baseline kernels (basic solutions), which, of course, do not show the best results on the leaderboard, but you can use them as a starting point for creating your own solutions.

Why Kaggle?

Ben Hamner
@benhamner

Kaggle's community passed two million members this morning! Thrilled to see so many people passionate about learning from data, and applying insights and machine learning models from it to make the world a better place

2:19 PM - 23 Aug 2018

1 Retweet 16 Likes

In fact, there is no difference what platform you play. Kaggle is just one of the first and the most popular, with a great community and comfortable environment (I hope they will modify the kernels for stability and performance. Many people remember the hell that was going on at Mercari) But, in general, the platform is very convenient and self-sufficient, and kaggle ranks (Master/Grandmaster) are still valuable.

A bit off the top on the subject of competitive Data Science. Very often, in articles, conversations, and other communication the opinion, that experience gained during the competition can't be used in the real-life problems, that all these people are only tuning 5th decimal digit of the score, that is insanity and too far from reality. Let us look at this question a bit closer:

As a practicing Data Science-professionals, in contrast to the people from academy and science, we should and will solve real business problems in our work. In other words, (here is the reference to the CRISP-DM) to solve this problem it is necessary:

- to understand the business problem

- to evaluate data to determine whether it holds the answer to this business problem
- to collect additional data if the existing one is not enough to get the answer
- to choose the metric that best approximates business goal
- and only after that choose a model, transform the data for the selected model and "stack xgbosts" ©.

The first four items on this list are not taught anywhere (correct me if there are such courses — I will enroll to it immediately), the only way is to learn from the experience of colleagues working in a specific industry. But the last clause — from selecting the model and further, you can, and you should improve in the competitions.

In any competition, most of the work was already completed for us by the sponsors. We already have a business objective, the selected approximating metric, collected data, and our task is to build a working pipeline out of all of these. And here's where the skills will improve — how to deal with missing values, how to prepare data for neural networks and trees (and why neural networks require a special approach), how to build a validation scheme, how to not overfit, how to select appropriate hyperparameters, and many more other "how to", competent execution of which distinguishes the experts from a passerbys in our profession.

What you can "farm" on Kaggle

Basically, all the newcomers come to Kaggle to improve their practical experience, but do not forget that, in addition, there are at least two additional purposes:

- Farm medals and ranks
- Farm reputation in Kaggle community

> ***The key thing to remember is that these three goals are completely different, different approaches are required to achieve them, and they should not be mixed especially during the initial phase!***

Pay your attention to the words **"initial phase"**, so when you improve your skills — these three goals will merge into one and will be achieved in parallel, but while you are just starting — **don't mix them up**! This way you will avoid pain, frustration, and resentment for this unjust world.

Let's walk briefly on the objectives from the bottom up:

- **Reputation** — improved by writing good posts (and comments) on the forum and the creation of useful kernels. For example, EDA kernel (see above), posts with a description of innovative techniques, etc.

- **Medals** — a very controversial topic, but well… It can be improved by blending public kernels (*), participating in the team with experience imbalance, and of course by creating your own top-solutions
- **Experience** — enhanced by analysis of decisions and mistakes.

(*) *blending public kernels — technics to farm medals, when the kernels with high scores on the public leaderboard are selected, their predictions are averaged (blended), and the result submitted. Usually, this method leads to hard overfitting on the public leaderboard but sometimes allows you to get almost silver. The author, at the initial stage, does not recommend this approach (read below about the belt and the pants).*

I recommend first to choose the “experience” goal and stick with it until you feel ready to work on two/three goals at the same time.

There are two more things worth mentioning (Vladimir Iglovikov (ternaus) — thanks for the reminder).

The first is to convert the effort invested in Kaggle into a new, more interesting and/or well-paid job. For people who understand the topic the line in the CV “Kaggle Competition Master”, and other achievements still worth something.

As an illustration of this point, you can read two interviews (ones, two, *please note that these interviews are in the Russian language*) with our colleagues Sergey Mushinskiy (cepera_ang) and Alexander Buslaev (albu)

And also the opinion from Valeriy Babushkin (venheads):

Valeriy Babushkin — Head of Data Science at X5 Retail Group (lead a team of 50+ people divided into 5 departments: CV, Machine Learning, Data Analysis, NLP and Ad Hoc), Analytics Team Lead at Yandex Advisor

Kaggle Competition Master is an excellent proxy metric for assessing the future team member. Of course, in connection with the recent events in the form of teams of 30 participants and with almost nobody of them making anything it requires a more careful study of the profile than before, but it’s still a matter of minutes. People who have achieved the title of master, with high probability, are able to write at least an average quality code, well versed in machine learning, are able to clean the data and to build sustainable solutions. If you don’t have a master rank, the mere fact of participation is also a plus, at least the candidate knows about that Kaggle exists and did spend some time on familiarizing himself with it. And if solution he contributed with was something more than a public kernel (which is quite easy to check), it is a good chance for a specific conversation about technical details that is much better and more interesting than the classical job interview for a fresh grad which provides much less information of how people in the future will succeed in the job. The only thing we have to fear, and what I came across is that some people think that Data Science work is the same as at Kaggle, which is totally not true. Many think that Data Science = Machine Learning, which is also a mistake.

The second point is that many tasks can be arranged in the form of pre-prints or articles, on the one hand, it allows the knowledge that the collective intelligence got during the competition not to die in the wilds of the forum, and on the other hand it adds another line to authors’ portfolio and +1 to visibility, that in any case positively affects the career and the citation index.

For example, this is the list of our colleagues' works at the end of several competitions:

- Satellite imagery feature detection using deep convolutional neural network: A Kaggle competition
- TernausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation
- Angiodysplasia Detection and Localization Using Deep Convolutional Neural Networks
- Automatic Instrument Segmentation in Robot-Assisted Surgery Using Deep LearningFully convolutional network for automatic road extraction from satellite imagery
- Ternausnetv2: Fully convolutional network for instance segmentation
- Feature pyramid network for multi-class land segmentation
- Paediatric Bone Age Assessment Using Deep Convolutional Neural Networks
- Camera Model Identification Using Convolutional Neural Networks
- Deep Learning Approaches for Understanding Simple Speech Commands
- Deep Convolutional Neural Networks for Breast Cancer Histology Image Analysis
- Diabetic Retinopathy detection through integration of Deep Learning classification framework
- Land Cover Classification from Satellite Imagery With U-Net and Lovasz-Softmax Loss
- Land Cover Classification With Superpixels and Jaccard Index Post-Optimization
- Building Detection from Satellite Imagery Using a Composite Loss Function
- North Atlantic Right Whale Call Detection with Convolutional Neural Networks
- Run, skeleton, run: skeletal model in a physics-based simulation
- Learning to Run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments
- Comparison of Regularization Methods for ImageNet Classification with Deep Convolutional Neural Networks
- Doppelganger Mining for Face Representation LearningHard Example Mining with Auxiliary Embeddings
- Sales forecasting using WaveNet within the framework of the Kaggle competition

How to avoid the pain of losing medals

Don't care about them!

Let me explain. Practically in every competition closer to its termination somebody writes the public kernel with a solution that shifts the entire leaderboard up, but you, with your solution, down. And every time at the forum The Pain Starts! "Oh, here I had a solution for the silver, and now I'm not even in the bronze. What the hell is going on, let's bring everything back."

Remember that Kaggle is the competitive Data Science and your place on the leaderboard depends only on you. Not from the guy that posted the kernel, not from the stars that came together or not, just on how much effort you put into the decision and whether you used all the possible ways to improve it.

> ***If the public kernel knocks you off your place on the leaderboard — it is not your place.***

Instead of having to pour out the pain for the injustice of the world — thank this guy. Seriously, the public kernel with a better solution than yours means that you have missed something in your own pipeline. Locate it, improve it and go around all the hamsters with the same score. Remember that to return to the place you just have to be a little better than this public kernel.

Just how I was upset by this moment in the first competition! My hands were falling, and I was ready to give up. A moment ago you were in silver, and here you are now at… the bottom of the leaderboard. It's ok, I just had to get myself together to understand where and what I missed to alter my decision and to return to the game.

Moreover, this moment will be present only in the early phase of your competitive process. The more experienced you become, the less you will feel the influence of the published kernel and the stars. In a recent competition (Talking Data, where our team took 8th place) there was such kernel, but it had been mentioned just in one line in our team chat from Pavel Pleskov (ppleskov): "Guys, I blended it with our solution, it had become worse, so I threw it away". That means that all useful signal, which was pulled from the data by that kernel was already pulled by our models.

And, btw, about the medals, remember:

> ***"The belt without skills serve only to hold up one's pants"***©

Where and how to write code

My recommendation is **python 3.6** with **jupyter notebook** under **ubuntu**. **Python** has already become the de facto standard in Data Science, given a large number of libraries and community. Jupyter, especially with the presence of **jupyter_contrib_nbextensions**, is very convenient for rapid prototyping, analysis and data processing. **Ubuntu** is easy itself, plus the part of data processing is sometimes more natural to do in bash instead of python.

After installation of **jupyter_contrib_nbextensions** I recommend to enable:

- ***Collapsible headings*** *(really helps to organize the code blocks)*
- ***Code folding*** *(the same)*
- ***Split cells*** *(useful if you need to debug something)*

And your life will become much more comfortable and more pleasant.

Once your pipeline becomes more or less stable, I recommend moving your code to the individual modules. Believe me — you will rewrite it more than once or twice or even five times, and that's ok.

There is the opposite approach where the participants are trying to use jupyter notebook as infrequently as possible and only when necessary, preferring just to write pipeline by scripts. (The adept of this option is, for example, (Vladimir Iglovikov (ternaus)). You can also read "Ask Me Anything session with a Kaggle Grandmaster Vladimir I. Iglovikov" where he is describing such approach.

And there are those who are trying to combine jupyter with any IDE, such as Pycharm.

Each approach has the right to exist, and each has its pros and cons, as they say, tastes differ. Choose what you are comfortable to work with.

But under any scenario remember the rule

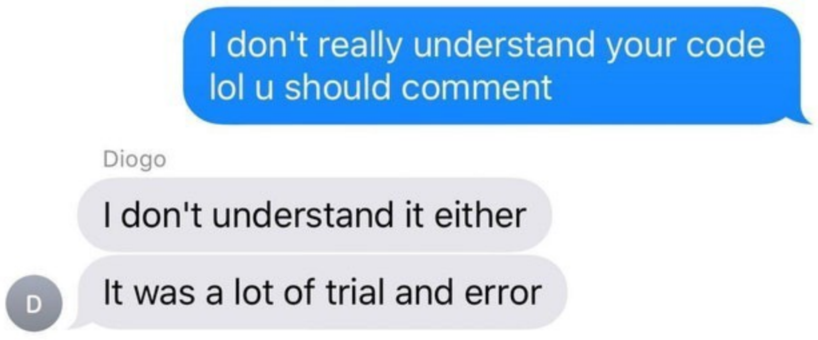
save the code for each submission/OOF made. (see below)

(*) OOF — out of folds, a technique of obtaining predictions of the model for the training part of a dataset using cross-validation. Is crucial for the further ensemble of several solutions.

How? Well, there are at least three options:

- For each competition, we create a separate repository on GitHub or BitBucket and commit code for each our submission to the repository with a comment that contains obtained score, model parameters, etc.
- A code after each submission assembles to a separate archive with the name of the file that contains all meta-information (obtained score, options, etc.)
- The specialized version control system for Data Science tasks is used. For example DVC.

In General, in the community, there is a tendency of gradual transition to the third option, because the first and the second ones have their drawbacks, but they are simple, reliable and, honestly, for Kaggle, they are more than enough.



I don't really understand your code lol u should comment

Diogo

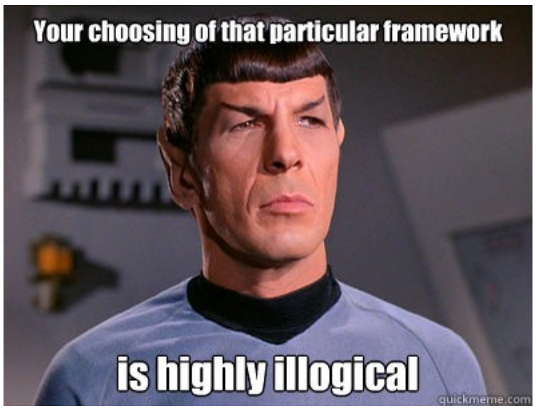
I don't understand it either

It was a lot of trial and error

A few more words about python for those who is not a programmer. Don't fear it. Your task is to understand the basic code structure and the basic essence of language. It will give you the ability to understand other people's kernels code and write your own libraries. On the Internet there are many excellent courses for beginners, maybe in the comments, somebody will give you the links.

Unfortunately (or fortunately) I can not evaluate the quality of such courses so I won't put such links in the article.

Well, to the framework we go



> All techniques described in this article are based on working with tabular and text data. Working with pictures is a separate task with separate frameworks. At a basic level, it's good to be able to work with them, at least to run something like ResNet/VGG and get features, but deeper and more subtle work with them is a separate and very broad topic not covered in this article.

I am not so good at working with imagery data. The only attempt to do it was in the competition Camera Identification, in which, by the way, the teams with the tag **[ods.ai]** blew up the whole leaderboard, so the Kaggle admins had to come to our slack to check that everything was according to rules. In this contest, I've got a silver medal with the 46th place, but when I read the description of the top solutions from our colleagues, I understood that this was my best place as they were using a real black magic with pictures augmentation, additioning more than 300GB of data, sacrificing, and other things. For example, you can read the post from Vladimir Iglovikov (ternaus) about "The importance of data augmentation".

In General, if you want to start working with images, then you need other frameworks and other guides.

The main goal

Your main task is to write pipelines (issued in the form of jupyter notebooks + modules) for the following tasks:

- EDA (exploratory data analysis). Here it is necessary to make a comment that there are specially trained people on Kaggle :), who write very nice and comprehensive EDA kernels in each competition. It is hard to beat them, but it's still necessary to understand how to look at the data because in your working tasks this specially trained person will be you. So you need to look at the methods and approaches and expand your libraries.
- Data Cleaning — all aspects of data cleaning. Working with missing values, outliers, etc.

- Data preparation is all about the preparation data for the model. Multiple blocks (Common, for regressions/neural networks, for tree models, special (time series, images, FM/FFM), text (Vectorizers, TF-IDF, Embeddings))
- Models (linear models, tree models, neural networks, exotic (FM/FFM))
- Feature selection
- Hyperparameters search
- Ensemble

In public kernels, all these tasks are usually gathered in one code, but I highly recommend to create a separate notebook and a separate module (or set of modules) for each of these subtasks. This approach will help you later.

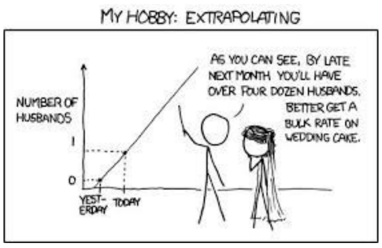
To prevent a possible holy war remember — the structure of this framework is not the truth in the last instance, there are many other ways to structure pipelines, and this is just one of them.

Data is transferred between the modules as CSV, or feather/pickle/hdf — the way you prefer and what you are used to or enjoy doing.

In fact, a lot still depends on the amount of data. In the TalkingData competition, for example, I had to go through memmap to avoid out of memory error when creating the dataset for lgb.

In other cases, the master data is stored in hdf/feather, something small (like the set of selected attributes) in the CSV. I repeat — there is no template, this is just one of the possible ways. You should try all of them and select the one which is best for you.

The initial stage


MY HOBBY: EXTRAPOLATING
NUMBER OF HUSBANDS
1
0
YEST-ERDAY
TODAY
AS YOU CAN SEE, BY LATE NEXT MONTH YOU'LL HAVE OVER FOUR DOZEN HUSBANDS. BETTER GET A BULK RATE ON WEDDING CAKE.

Start with any Getting started competition (as already mentioned, I started with House Prices: Advanced Regression Techniques), and begin to create your pipelines and notebooks. Read the public kernels, copy pieces of code, procedures, approaches, etc., then run your pipeline over your data, submit — look at the results, improve, rinse and repeat.

> ***The main task at this stage is to create an efficiently operating full cycle pipeline — from reading and cleaning the data to creating the final submission.***

A sample list of what should be ready and work at 100% before proceeding to the next step:

- EDA (dataset statistics, charts, a range of categories, …)
- Data Cleaning (filling missing using fillna, cleaning categories, combining categories)
- Data preparation (common (processing categories — label/ohe/frequency, the projection of the numeric to the categorical, the numeric transformation, binning), for regressions (different scaling))
- Models (linear models (different regression — ridge/logistic), tree models (lgb))

- Feature selection (grid/random search)
- Ensemble (regression / lgb)

Go into the battle

Choose any competition you like and let's start…

- Look at the data, read the forums and build a robust validation scheme. Without robust validation, you will fall down in the leaderboard, just like it happened in the competition from Mercedes, Santander, and others. Look at the leaderboard of the Mercedes and discussion, for example (green arrows and digits indicate the number of positions the people rose up in leaderboard compared to public, red — how many went down)

At this time (2018–02–10) the arrows were removed from the leaderboard of all competitions. I don't know the reason but hope that it will come back.

- Also, I recommend reading the article(Rus) and the speech(Rus) of Danila Savenkov (danila_savenkov):
- Also, this topic is well covered in course How to Win a Data Science Competition: Learn from Top Kagglers"

> ***Do not continue until you build a robust validation schema — !!!***

OK, you got robust validation, what to do next?

- Run your pipeline and submit the result
- Grab your head, freak out, calm down… and continue…
- Read all kernels/forum to know about used techniques and approaches
- Rebuild your pipelines with new techniques
- Go to step 1

Remember that your goal at this stage is to gain experience! We have to fill our pipeline with working approaches and methods, and our modules by working code. Don't bother about the medals — I mean, it would be great if you could immediately take some place in the leaderboard, but if not, don't get upset. We didn't come here for five minutes, so medals and ranks are not going anywhere.

OK, the competition is over, you are somewhere at the leaderboard, you learn something and want to go to the next competition?

> ***NO!***

What you do next:

- Wait for five days. Do not read the forum, forget about Kaggle for this time. Give the brain a break and let your eyes have a chance to sharpen again.
- Then return back to the competition. During these five days, according to the rules of etiquette, all top teams will post their solutions on the forum, in the form of posts, kernels, or GitHub repositories.

And here your personal hell begins

- You take a few sheets of A4, on each write the name of the module from the above framework (EDA/ Preparation/ Model/ Ensemble/ Feature selection/ Hyperparameters search/ …)
- Then consistently read all the top decisions and fill the corresponding sheets with new techniques, methods, approaches.

And the worst:

- Sequentially for each module, you write the implementation of these approaches and methods, expanding your pipeline and libraries.
- Then you run your updated pipeline and post-submit the solution.
- Repeat that until you have a solution in the gold zone of the leaderboard, well, or until you run out of patience and nerves.

And only after that move on to the next competition.

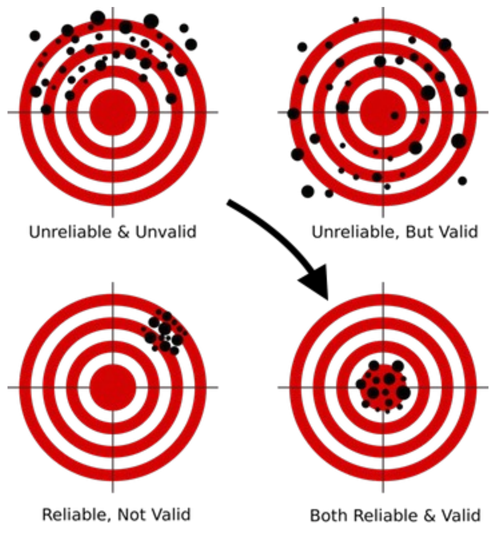
No, I'm not joking. Yes, you can do it easier. It's for you to decide.

Why wait 5 days and not read the forums immediately? Well, you are losing the opportunity to ask some questions, but at this stage (in my opinion) it's better to read the already formed threads with discussions of solutions then ask questions that either someone already asked, or it's better not to ask them yet, and look for the answer yourself.

Why should you do this exactly? Well, again — the purpose of this stage is to develop your own database of solutions, methods, and approaches. To create a working and efficient pipelines. So in the next competition you do not waste time and just say — yeah, mean target encoding could be used here, and by the way, I have the correct code for it using folds in folds. Or, well, I remember in that task the best way was to use ensemble using scipy.optimize and by the way I have the code already ready for it.

Something like that…

Go to work mode



With the same approach, you should participate in a few other competitions. Each time we notice that the entries on our sheets are becoming fewer and fewer, and the code in modules larger and larger. Gradually, the task of analysis converges to ensuring that you read the description of the solution, say yeah, hell yeah! And add to your library one or two new methods or approaches.

After this, the learning mode changes to error-correction mode. The base library is ready, now it just must be used correctly. After each competition, you read the description of the solutions, look what you did, what could have been done better, what you missed, or where you specifically messed up, as I had in Toxic. I was going quite well in the underbelly of the gold zone, and in the private fell down 1500 positions. It was insulting to tears… but I calmed down, found a bug, wrote a post to our Slack community — and learned a lesson.

A sign of establishing working mode is the fact that one of the descriptions of top solutions will be written under your nickname.

What roughly speaking should be in your pipeline by the end of this stage:

- Various methods for the preprocessing and creation of numeric features — projection, relationship
- Different methods on how to work with categories — Mean target encoding in the right form, frequency, label / ohe
- Various schemes of embeddings for the text (Glove, Word2Vec, Fasttext)
- Various vectorization schemes of text (Count, TF-IDF, Hash)
- Several validation schemes (N*M for the standard cross-validation, time-based, by group)
- Bayesian optimization / hyperopt / something else for the selection of hyperparameters
- Shuffle / Target permutation / Boruta / RFE for feature selection
- Linear models — write a code to run different models in the same style over one processed data set
- LGB/XGB/Catboost — write a code to run different models in the same style over one processed data set

The author created several metaclasses separately for linear and tree-based models with the same external interface in order to neutralize the differences in API between the different realization of models libraries. But now he can run different models of the same type (LGB or XGB for example) by one line of code over one processed data set.

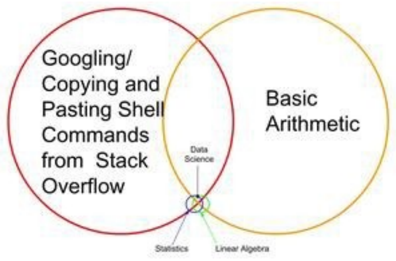
- Several neural networks of different types (not for pictures) — embeddings/CNN/RNN for text, RNN to sequence, a Feed-Forward for the rest. It’s good to understand and implement autoencoders.
- The ensemble on the basis of lgb/regression/scipy — separately for regressions and classifications tasks.
- It’s good to be able to use Genetic Algorithms, sometimes they work pretty well.

To summarize

As in any sport, in competitive Data Science, there is a lot of sweat and a lot of work. This is neither good nor bad, it is a fact. Participation in competitions (if you correctly approach the process) boosts technical skills significantly, plus more or less develops a sporting spirit — when you really do not want to do something, but you get up to the laptop, rebuild the model and run it to improve this damned 5th decimal digit of your score.

So go play Kaggle — to farm experience, medals, and fun!

A few words about the author’s pipelines


Googling/ Copying and Pasting Shell Commands from Stack Overflow
Basic Arithmetic
Data Science
Statistics
Linear Algebra

In this section, I’ll try to describe the basic idea behind my pipelines collected for a year and a half. Again — this approach does not claim universality or uniqueness, but perhaps you can find it useful.

- All the code on the feature engineering, except the mean target encoding, is written as functions in a separate module. I tried to do it through objects, but it turned out cumbersome, and in this case not necessary.
- All the functions of the feature-engineering are made in the same style and have the same call signature and return:

```
def do_cat_dummy(data, attrs, prefix_sep='_ohe_', params=None):
 # do something
return _data, new_attrs
```

The inputs are a dataset, the attributes, the prefix for the new attributes and additional parameters. The outputs are a new dataset with new attributes and a list of these attributes. Further, this new dataset is stored in a separate pickle/feather.

What it gives — we are able to assemble working dataset from pre-built parts quickly. For example, we made three different handling of categories — Label Encoding / OHE / Frequency and stored them in three separate feathers, and then at the modeling stage, all we need is just to play with these blocks, creating different datasets for training by one elegant move.

```
pickle_list = [
 ‘attrs_base’,
 ‘cat67_ohe’,
 # ‘cat67_freq’,
 ]
short_prefix = ‘base_ohe’

_attrs, use_columns, data = load_attrs_from_pickle(pickle_list)
cat_columns = []
```

If it is necessary to assemble another working dataset — we just need to replace pickle_list, reload and work with a new dataset.

A basic set of functions on tabular data (real and categorical) includes various categories encoding, the projection of the numerical attributes to categorical, as well as various transformations.

```
def do_cat_le(data, attrs, params=None, prefix=’le_’):
def do_cat_dummy(data, attrs, prefix_sep=’_ohe_’, params=None):
def do_cat_cnt(data, attrs, params=None, prefix=’cnt_’):
def do_cat_fact(data, attrs, params=None, prefix=’bin_’):
def do_cat_comb(data, attrs_op, params=None, prefix=’cat_’):
def do_proj_num_2cat(data, attrs_op, params=None, prefix=’prj_’):
```

Universal Swiss army knife for combining the attributes, it gets the list of source attributes and the list of transform functions as input, and its output is, as usual, dataset and a list of the new attributes.

```
def do_iter_num(data, attrs_op, params=None, prefix=’comb_’):
```

Plus various additional specific converters.

For processing text data a separate module is used, including different methods of preprocessing, tokenization, lemmatization/stemming, translation in the frequency table, etc., using sklearn, nltk and keras.

Time series are handled in a separate module with functions to convert source dataset for common tasks (regression/classification), as well as for sequence-to-sequence tasks. Thank you François Chollet for keras update, so now building seq-2-seq models do not look like a voodoo ritual of demons invocation.

There are, by the way, the functions of conventional statistical analysis of series in the same module — test for stationarity, STL-decomposition, etc… it is very helpful at the initial stage of analysis to “feel” the time series and to see what it actually is.

- Functions that cannot be applied immediately to the entire dataset, and you need to use it inside folds cross-validation, are moved to a separate module:
- Mean target encoding
- Upsampling / Downsampling
- They are passed into the inside of the model class (see below) at the training stage.

```
_fpreproc = fpr_target_enc
 _fpreproc_params = fpr_target_enc_params

 _fpreproc_params.update(**{
 ‘use_columns’ : cat_columns,
 })
```

- For modeling, the new metaclass was created, which generalizes the concept of model, with abstract methods: fit/predict/set_params/ etc. For each library (LGB, XGB, Catboost, SKLearn, RGF, …) separate implementation of this metaclass has been created.

In other words, to work with the LGB we create a model

```
model_to_use = ‘lgb’
model = KudsonLGB(task=’classification’)
```

For XGB:

```
model_to_use = ‘xgb’
metric_name= ‘auc’
task=’classification’

model = KudsonXGB(task=task, metric_name=metric_name)
```

And now in our code, we can operate with the model universarly.

- For validation several functions were created, one that calculates the prediction and OOF simultaneously for a few seeds, another one is the basic validation using train_test_split, and so on. All functions operate with meta-model methods, which gives the model-independent code and facilitates the connection to pipeline any other library

```
res = cv_make_oof(
model, model_params, fit_params, dataset_params,     XX_train[use_columns],
yy_train, XX_Kaggle[use_columns], folds, scorer=scorer, metric_name=metric_name,
fpreproc=_fpreproc, fpreproc_params=_fpreproc_params, model_seed=model_seed,
silence=True
)score = res[‘score’]
```

- For feature selection — there is nothing interesting, just standard RFE, and my favorite shuffled permutation in all possible ways
- To find hyperparameters I mainly use Bayesian optimization, again in a standardized form to be able to run a search for any model (via cross-validation module). This block of code resides in the modeling notebook
- For ensembles, several functions were created, standardized for regression problems and classification on the basis of the Ridge/Logreg, LGB, Neural network and my favorite scipy.optimize
- A little explanation — every model in the pipeline produces two files as an output: sub_xxx and oof_xxx, representing a prediction for a test set and OOF prediction for the training set. Next, in the ensemble module, these files (from all models) are loaded from the specified directory into two data frames — df_sub / df_oof. Then we are looking at the correlations, selecting the best, and then build models of the 2nd level over df_oof and apply to df_sub
- Sometimes searching for the best subset of models with genetic algorithms produces an excellent result (I use this library), sometimes a method from Caruana works well. In the simplest cases, standard regression and scipy.optimize do the job.
- Neural networks live in a separate module, I use keras in a functional style, Yes, I know, it’s not as flexible as pytorch, but quite enough. Again, universal training functions were written, invariant to the type of the network

This pipeline was once again tested in the recent competition from Home Credit, careful and accurate application of all the blocks and modules brought the author the 94th place and a silver medal.

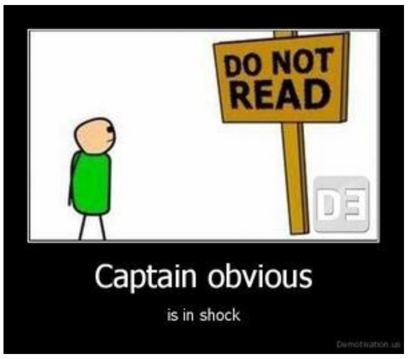
The author is actually willing to express a seditious thought that for tabular data and a decent pipeline, the final result for any competition must be in the top 100 of the leaderboard. Of course there are exceptions, but in general, this statement seems correct.

About teamwork

How to solve Kaggle — in a team or solo — mainly depends on a person (and a team), but my advice for those who are just starting — try to start solo. Why? I'll try to explain my point of view:

- At first, you will see your own strengths and weaknesses and will be able to evaluate your own potential as a data scientist in general.
- Secondly, even working in a team (unless it is a well-established team with a separation of roles), the team will still be waiting for a finished solution from you. So you have to have working pipelines to team well.
- And third, optimally, when the levels of the players in the team are about the same (and sufficiently high), then you can learn something really useful. In weak teams (there is nothing derogatory, I'm talking about the level of training and experience on Kaggle) IMHO it is very difficult to learn something, it is better to read the forum and kernel. Yes, it is possible to farm medals, but see above about the target and a belt for keeping your pants

Good bits of advice from captain obvious and the promised map of general mistakes:



These tips reflect the experience of the author and can (and should) be verified by your own experiments

- Always start with building a robust validation scheme — if not, all other efforts will be for naught. Another look at the leaderboard of the Mercedes.

The author really pleased that in this competition he has built a robust scheme of cross-validation (3x10 fold), which has kept a score and brought 42nd place.

- If you've built a robust validation — always trust the results of your validation. If the score of your models is improved by validation, but become worse on the public leaderboard, it is wiser to trust validation. In the analysis just consider the piece of data, which is a public leaderboard as another fold. You wouldn't want to overfit your model on a single fold, right?
- If the model and the scheme allows — always make OOF predictions and save it next to the model. At the stage of the ensemble, you never know what will help most.
- Always save the code next to the result/OOF to recreate them. It doesn't matter where to save — at GitHub, locally, anywhere. Twice I had the case that the best model in the ensemble is the one that was made two weeks ago out of the box, and for which there was no saved code. Pain.
- Give up the selection of the "right" seed for cross-validation. Better choose any three ones and make 3xN cross-validation. The result will be more stable.
- Do not chase the number of models in the ensemble — better fewer, but more diverse — diverse in models, pre-processing, the datasets. In the worst case — in the parameters, for example, one deep tree with hard regularization, one shallow.
- For feature selection use the shuffle permutation / boruta / RFE, remember that the feature importance in different tree-based models is the metric of parrots at a bag of sawdust.
- The author's personal opinion (may not coincide with the opinion of the reader) Bayesian optimization > random search > hyperopt for the selection of hyperparameters. (">" == better)
- If you see the public kernel with a better score on the public leaderboard the best way to handle it is:

1. If you have time — see what's new it has and update your pipelines
2. Less time — rebuild it for your validation scheme, run, get OOF prediction — and use it in your ensemble
3. No time — just blend with your best solution and see the score.

- How to choose the two final submits — by intuition, of course. But seriously, usually everyone practices the following approaches:

1. Conservative submit (sustainable models) / risky submit.
2. With the best score on validation / public leaderboard

> ***Remember — everything around you is the digits, and the possibilities to use them depend only on your imagination. Use classification instead of regression, consider the sequence as a picture, etc.***

And finally:

- Join the ods.ai, enjoy and have fun from Data Science and from life!

Useful links

Note that some materials below are in the Russian language,

Common

http://ods.ai/ — for those who want to join the best Data Science community

https://mlcourse.ai/ — website of the nice Machine Learning course

https://www.Kaggle.com/general/68205 — post about the course (previous link) on Kaggle

Overall, I would highly recommend viewing the set of video mltrainings (Rus) using the method that was described in this article, there are a lot of interesting approaches and techniques.

Video

- very good video about how to become a grandmaster :) from Pavel Pleskov (ppleskov)
- video about the hacking, unconventional approach and target mean encoding on the example of BNP Paribas competition from Stanislav Semenov (stasg7)
- Another video with Stanislav “What does Kaggle teach”

Courses

You can learn more methods and approaches to solve problems on Kaggle from the second course of this specialization — How to Win a Data Science Competition: Learn from Top Kagglers.

Extra reading:

- Laurae++, XGBoost/LightGBP parameters
- FastText — embeddings for text from Facebook

- WordBatch/FTRL/FM-FTRL — a set of libraries from @anttip
- Another implementation of the FTRL
- Bayesian Optimization library for selection of hyperparameters
- Regularized Greedy Forest (RGF) library — another tree method
- A Kaggler's Guide to Model Stacking in Practice
- ELI5 is a wonderful library for visualization of weights of the models from Konstantin Lopuhin (kostia)
- Feature selection: target permutations, and follow on the links inside
- Feature Importance Measures for Tree Models
- Feature selection with null importance
- Brief introduction about autoencoder
- Presentation about Kaggle on SlideShare
- Another one
- Lots of interesting things here
- Winning solutions of kaggle competitions
- Data Science Glossary on Kaggle

Conclusion

The topic of Data Science in general and competitive Data Science, in particular, is *"as inexhaustible as the atom"* ©. In this article, the author only scratched the surface of the topic of leveling practical skills using a competitive platform. If you are interested — sign up on Kaggle, inspect it, collect experience — and write your own articles. The higher amount of good content the better for all of us!

Anticipating questions — no, author's pipelines and libraries are not yet made public.

Many thanks to the colleagues from the ods.ai: Vladimir Iglovikov (ternaus), Yury Kashnitskiy (yorko), Valeriy Babushkin (venheads), Alexey Pronkin (pronkin_alexey), Dmitry Petrov (dmitry_petrov), Artur Kuzin (n01z3), as well as everyone who read the article prior to publication for corrections and reviews.

Special thanks to Nikita Zavgorodnii (njz) — for the final proofreading.

Also thanks to Alexey Iskrov (optimystic) and Vlad Ivanov(fateful) for the translation corrections.

Thank you for your attention, I hope this article will be useful to someone.