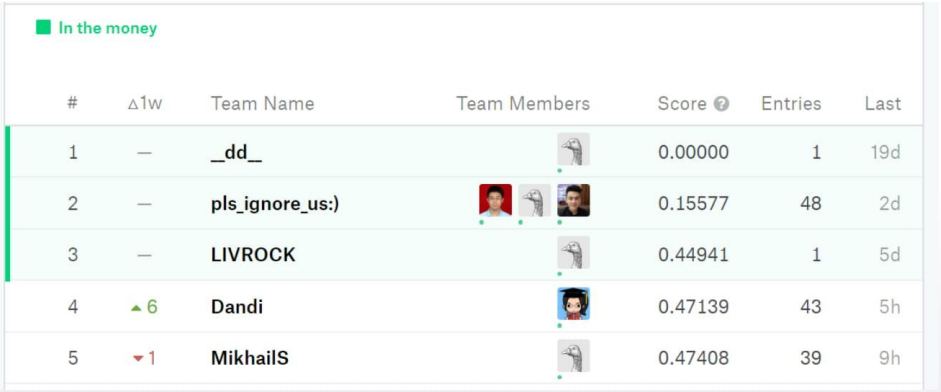
Practical Guide

Alexander Guschin

Before you enter a competition

Define your goals. What you can get out of your participation?

1. To learn more about an interesting problem
2. To get acquainted with new software tools
3. **To hunt for a medal**



In the money

#	△1w	Team Name	Team Members	Score	Entries	Last
1	—	_dd_		0.00000	1	19d
2	—	pls_ignore_us:)		0.15577	48	2d
3	—	LIVROCK		0.44941	1	5d
4	▲ 6	Dandi		0.47139	43	5h
5	▼ 1	MikhailS		0.47408	39	9h

After you enter a competition: Working with ideas

1. Organize ideas in some structure
2. Select the most important and promising ideas
3. Try to understand the reasons why something does/doesn't work

After you enter a competition: Everything is a hyperparameter

Sort all parameters by these principles:

1. Importance
2. Feasibility
3. Understanding

Note: changing one parameter can affect the whole pipeline

Dmitry Altukhov

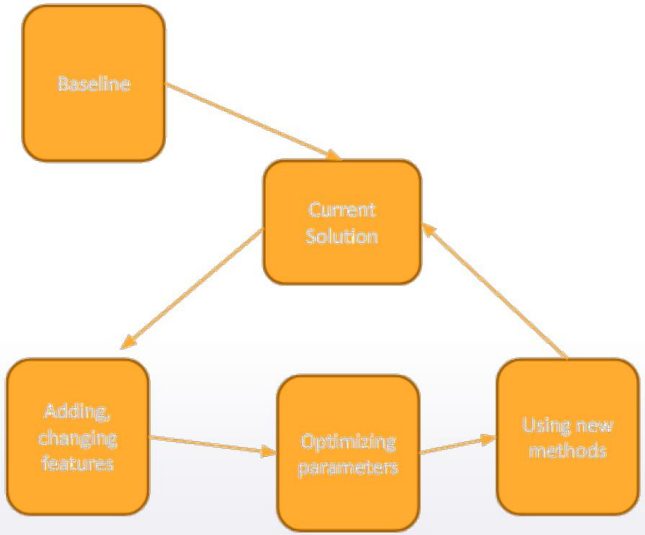
Data loading

- Do basic preprocessing and convert csv/txt files into hdf5/npy for much faster loading
- Do not forget that by default data is stored in 64-bit arrays, most of the times you can safely downcast it to 32-bits
- Large datasets can be processed in chunks

https://towardsdatascience.com/3-simple-ways-to-handle-large-data-with-pandas-d9164a3c02c1

Performance evaluation

- Extensive validation is not always needed
- Start with fastest models - LightGBM



Fast and dirty always better

- Don’t pay too much attention to code quality
- Keep things simple: save only important things
- If you feel uncomfortable with given computational resources - rent a larger server

Mikhail Trofimov

Initial pipeline

- Start with simple (or even primitive) solution
- Debug full pipeline
  - From reading data to writing submission file
- “From simple to complex”
  - I prefer to start with Random Forest rather than Gradient Boosted Decision Trees

Best Practices from Software Development

- Use good variable names
  - If your code is hard to read — you definitely will have problems soon or later
- Keep your research reproducible
  - Fix random seed
  - Write down exactly how any features were generated
  - Use Version Control Systems (VCS, for example, git)
- Reuse code
  - Especially important to use same code for train and test stages

Read papers

- This can get you ideas about ML-related things
  - For example, how to optimize AUC
- Way to get familiar with problem domain
  - Especially useful for feature generation

https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=ko

Dmitry Ulyanov

My pipeline

- **Read forums and examine kernels first**
  - There are always discussions happening!
- **Start with EDA and a baseline**
  - To make sure the data is loaded correctly
  - To check if validation is stable
- **I add features in bulks**
  - At start I create all the features I can make up
  - I evaluate many features at once (not “add one and evaluate”)
- **Hyperparameters optimization**
  - First find the parameters to overfit train dataset
  - And then try to trim model

Code organization: keeping it clean

- **Very important to have reproducible results!**
  - Keep important code clean
- **Long execution history leads to mistakes**

```
In [521]: pr = bst.predict(X_val)
          smape(y_val, pr)

Out[521]: 0.04287853677855423
```

- **Your notebooks can become a total mess**

```
s = qq.sum(1)
ss = s[:,3]/qq.var()
sss = ss[0]
```

Code organization: keeping it clean

- **One notebook per submission** (and use git)

- **Before creating a submission restart the kernel**
  - Use “Restart and run all” button

Code organization: test/val

- Split *train.csv* into *train* and *val* with structure of *train.csv* and *test.csv*

1.
```
train = pd.read_csv('data/train.csv')
test = pd.read_csv('data/test.csv')
```

2.
```
from sklearn.model_selection import train_test_split

# train set into new train and validation
train_train, train_val = train_test_split(train, random_state=660)

# save to disk
train_train.to_csv('data/val/train.csv')
train_val.to_csv('data/val/val.csv')
```

https://roboreport.co.kr/기계학습-훈련-데이터-구성-방법/

Code organization: test/val

- When validating, set it at the top of the notebook

```
train_path = 'data/val/train.csv'
test_path = 'data/val/val.csv'
```

- To retrain models on the whole dataset and get predictions for test set just change

```
train_path = 'data/train.csv'
test_path = 'data/test.csv'
```

Code organization: macros

I use macros for a frequent code

```
In [2]: __imp
```

```
In [5]: __fsmall
```

Code organization: macros

In [3]:
```
print __imp
```

```
import os
PYTHON_UTILS_PATH = os.environ['PYTHON_UTILS_PATH']

import numpy as np
import pandas as pd
from tqdm import tqdm_notebook

import matplotlib.pyplot as plt
get_ipython().magic(u'matplotlib inline')
plt.rcParams['figure.figsize'] = (14, 14)

# This will populate `globs_to_run`
get_ipython().magic(u'run {PYTHON_UTILS_PATH}/config.py')

for g in globs_to_run:
    files_to_run = get_ipython().getoutput(u'ls {os.path.join(PYTHON_UTILS_PATH,g)}')

    for f in files_to_run:
        if 'Macro.py' not in f:
            get_ipython().magic(u'run {f}')

import matplotlib as mpl

mpl.rcParams['axes.color_cycle'] =['#ff0000', '#0000ff',   '#00ffff','#ffA300', '#00ff00',
     '#ff00ff', '#990000', '#009999', '#999900', '#009900', '#009999']
from matplotlib import rc
rc('font', size=16)
rc('font',**{'family':'serif','serif':['Computer Modern']})
rc('text', usetex=False)
rc('figure', figsize=(16, 14))
rc('axes', linewidth=.5)
rc('lines', linewidth=1.75)
```

Code organization: custom library

- **I use a library with frequent operations implemented**
  - Out-of-fold predictions
  - Averaging
  - I can specify a classifier by it's name

```
param = {
    'C':1.2,
}
res = trylib(train_2lv, y_train,
             'lsvc', param,
             one = False, skf_seed = 660, skf = 4,
             test_mode='whole', X_test=test_2lv).res
```