Командный практический тур

Описание задачи финала

В качестве задачи командного практического тура заключительного этапа была предложена задача, связанная с производством сверхпроводников. Сверхпроводник — это лента, на которую наносится специальное покрытие. Сверхпроводники обладают малым сопротивлением, что особо важно при производстве мощных магнитов, однако имеют применение и в других сферах. Производство сверхпроводника — сложный технологический процесс. В управлении напылением из-за ряда факторов происходит проседание качества, а из-за особенностей использования необходимо производить длинные участки без единого проседания.

В рамках задачи рассматривается вакуумное напыление с ионным ассистированием, процесс IBAD (Ion Beam Assisted Deposition). Суть процесса заключается в том, что лента едет по зоне осаждения, и испаритель осаждает на ленту оксид магния. Лента едет по так называемой улитке. Т. е. проехав зону осаждения один раз, лента разворачивается и проезжает эту же зону осаждения второй, 3-й разы. Конкретно для изучаемого процесса лента проезжает зону 3 раза. На пути отсутствуют развороты, но важно понять суть, что события в испарителе одномоментно действуют на три участка ленты, которые удалены друг от друга на 2,42 м.

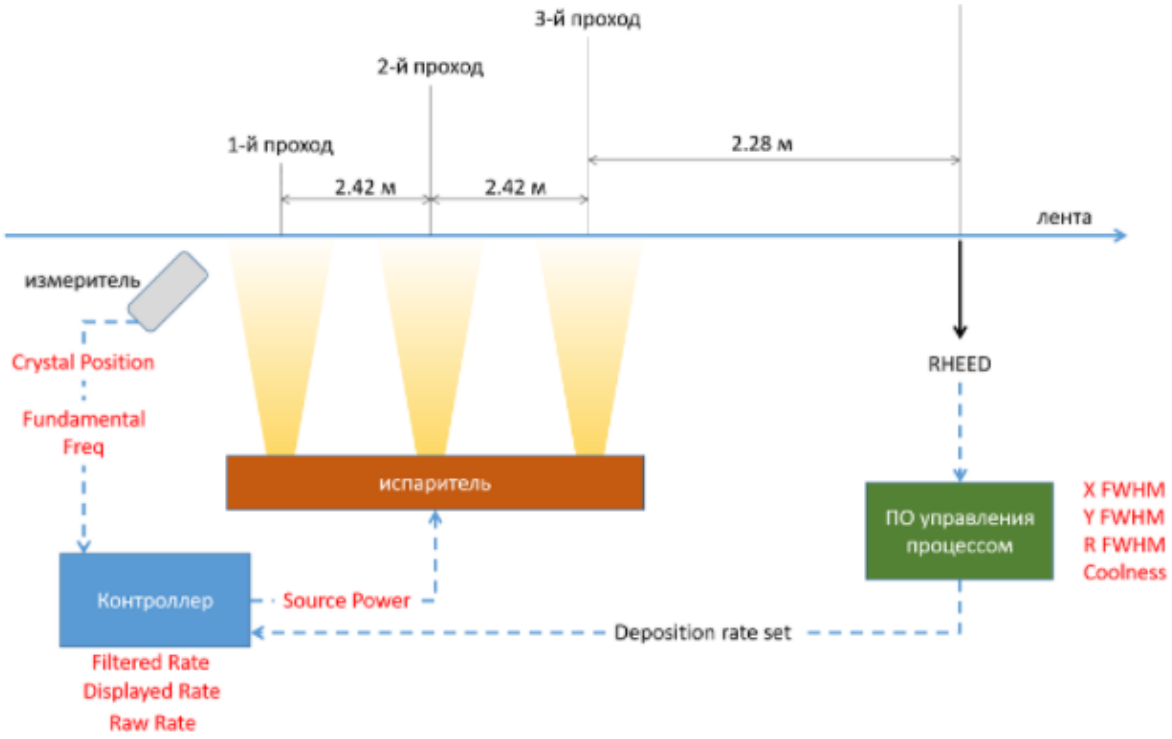


Рис. III.2.1: Процесс вакуумного напыления с указанием считываемых параметров

Испаритель работает на мощности Source Power (все параметры, которые присутствуют в датасете выделены красным). Эту мощность задаёт контроллер. Кон-

троллер получает данные по частоте резонатора (Fundamental Freq) от измерителя и по падению частоты определяет скорость осаждения. Измеритель имеет расходный элемент резонатора, который периодически переключается оператором, что записывается в параметр Crystal Position. По мере работы резонатора стабильность его частоты ухудшается, и измерения становятся шумными, поэтому оператор переключает его на следующий. Контроллер отдаёт сразу три измеренных скорости: Filtered Rate, Displayed Rate и Raw Rate. У контроллера есть постоянная времени фильтра скорости осаждения. Она зафиксирована для всех процессов. На основе измеренной скорости осаждения и уставки по скорости осаждения контроллер рассчитывает мощность испарителя (Source Power). Чем больше мощность, тем больше скорость осаждения. На самом деле в эту же зону осаждения светит ионный источник. Этот ионный источник травит слой, который осаждает испаритель. Т. е. отсутствие или снижение осаждения не означает, что просто ничего не происходит. В этом случае происходит травление слоя, полученного на предыдущих проходах, или даже предыдущих процессах. Кроме того, травление происходит неравномерно по проходам зоны осаждения. 1-й и 3-й проходы имеют типично около 70% скорости травления от 2-го прохода.

Под таким процессом лента проезжает три раза, и через 2,28 м. от 3-го прохода качество полученного слоя анализируется дифракционным инструментом RHEED (Reflected High Energy Electron Diffraction).

Софт управления измеряет три сечения выбранного пика (X FWHM, Y FWHM и R FWHM). FWHM — полная ширина на уровне половинной амплитуды (англ. FWHM — full width at half maximum). На основе пика также рассчитывается параметр Coolness («холодок»). Софт управления имеет уставку по параметру Coolness и меняет уставку скорости осаждения в контроллере. Параметры дифракции в RHEED определяются кооперативным влиянием процессов на всех проходах. В общем случае, чем больше скорость осаждения, тем больше Coolness (на самом деле есть максимум на высоких скоростях осаждения, но по факту не применяется). Характеристика Coolness(Rate) нелинейная, чем меньше скорость осаждения и Coolness, тем круче зависимость, а работать необходимо именно на низких значениях Coolness. Собственно прогноз Coolness является основной задачей.

Типично логи данных строятся не от времени, а от координаты ленты (Length).

В данных изначально присутствуют два типа логов (#915_M1 — лог испарителя и #915_RHEED — лог анализатора качества). Для задачи они уже сведены в один датасет, те колонки которые присутствовали в логе RHEED, помечены как название колонки +_RHEED. Важно, в этих логах координата для событий испарителя сдвинута так, как будто они происходят на 2-м проходе ленты, а внутренне мы понимаем, что эти же события происходят и 2,42 м. назад, и 2,42 м. вперёд по координате. В принципе, для каждой строки есть время записи этого параметра, и оно остается истинным.

Получается такое распределение, например:

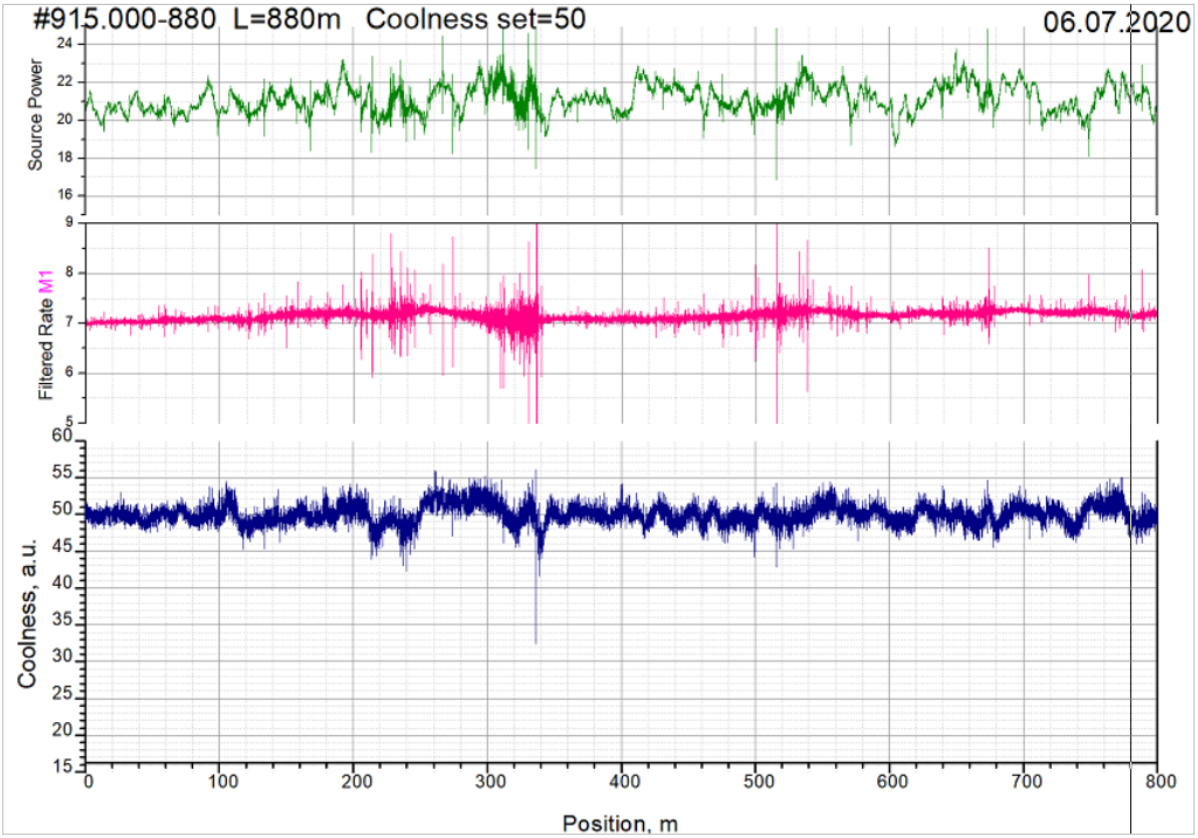


Рис. III.2.2: распределение значений занчимых параметров

Для более полного понимания, ниже описан ещё специальный эксперимент искусственного возмущения. Был сымитирован провал по скорости осаждения. Уставка скорости осаждения линейно снижалась примерно один период между проходами, затем возвращалась в регулярное значение. Для наглядности лог испарителя продублирован для всех проходов. Лента едет меньшей координатой вперёд, поэтому в анализатор RHEED сначала попадает часть ленты, которая воспринимала возмущение на 3-м проходе. Видно, что есть возмущение на Coolness, но текстура не исчезла, т. е. стравилась часть слоя.

RHEED с 2-го прохода показывает, что слой был полностью стравлен и даже повреждён слой предыдущего процесса (места, где coolness не определён, т. е. = 0).

Ну и наконец. Возмущение на 1-м проходе практически не повлияло на Coolness, т. к. слой успел нарости во 2-м и 3-м проходах.

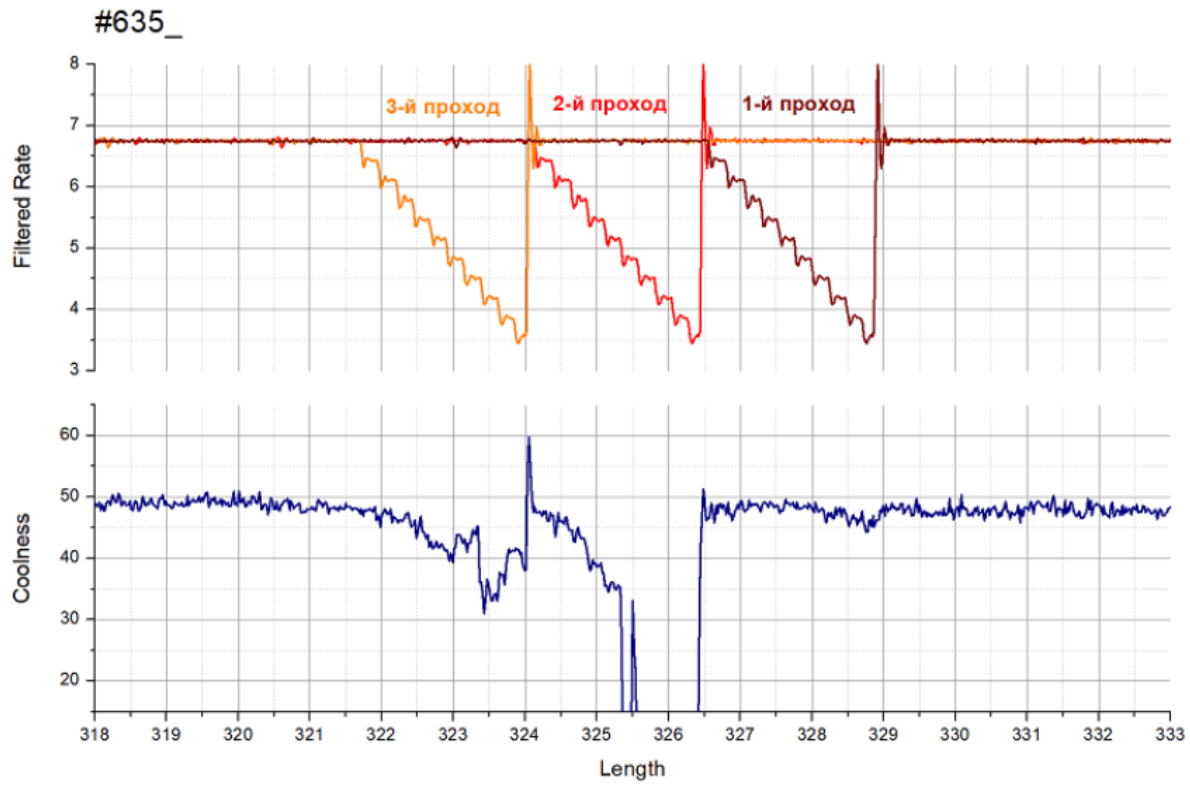


Рис. III.2.3: Распределение значений значимых параметров в эксперименте

Формулировка задачи для участников

В рамках задачи необходимо прогнозировать уменьшение параметра холодок («Coolness», среднее за 50 измеренений, примерно 1 метр) на 5 более по сравнению со средним значением за последние 1 метра, через 435 измерений, примерно 9 метров относительно точки измерения.

Формат входа: два массива значений аналогичные входным данным.

Формат выхода: вероятность относительно уменьшения значения более чем на 5 на 9 метров, относительно последней метки в массивах. Глубина данных для анализа, которые будут передаваться 10,000 замеров (примерно 23 метра).

Целевая метрика ROC AUC.

Решения необходимо присылать на платформу `https://sim.avt.global`.

Для загрузки решения необходимо создать архив, содержащий файл submission.py, в котором представлено ваше решение. В этом файле должен быть объявлен класс Predictor с функциями init(self) и forecast(self,df).

- init(self), нужен для инициализации класса, вызывается один раз за тестирование, нужен чтобы загрузить все модели, для экономии времени.
- forecast(self,df) нужен для генерации прогноза.

В качестве df в функцию forecast передается pandas DataFrame, на 10000 значений, с колонками соответствующим данным. DataFrame полностью аналогичен приведенному, в нем могут быть пропуски, замены лент и т. д. В качестве возвращаемого

значения функция forecast возвращает одно число от 0 до 1 – вероятность уменьшения более, чем на 5 усредненного Coolness за ближайшие 9 метров.

С учетом большего количества вопросов про целевую переменную, поясняем дополнительно: целевая переменная равна 1(True), если усредненное значение параметра Coolness за последние 50 измерений больше, чем на 5 среднего значения Coolness в 385-435 измерениях после данного момента, в ином случае 0 (false). Иными словами, для каждой строчки можно подсчитать среднее значение Coolness за последние 50 показателей, и, если среднее сейчас больше, чем на 5 среднего через 435 строчек.

Вычисление этого параметра, можно представить так:

```
cool_col= rheed_long.rolling(50).mean()
y = ((cool_col- cool_col.shift(-435))>5),
```

где rheed_long — входные данные.

Метрика по которым будут сравнивать решения — roc_auc (она измеряется от 0 до 1, чем больше, тем лучше).

Баллы за задачу начисляются относительно лучшего решения среди участников.

С примером корректного submisson, содержащим модель и класс, поддерживающего созданную модель, можно ознакомится по ссылке: https://drive.google.com/file/d/1QKBUYRGXTIXJ4HsFZ7T2EsEEOLFDw1o-/view?usp=sharing.

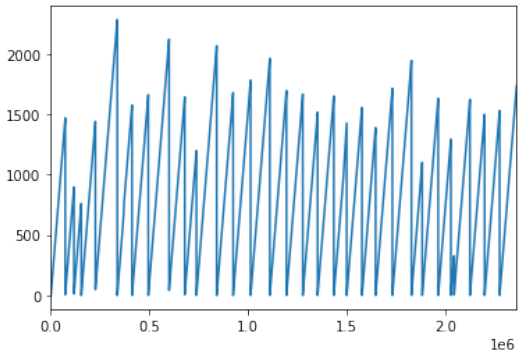
Для корректной работы модели необходимо хранить список колонок.

Базовое решение

https://drive.google.com/file/d/1QKBUYRGXTIXJ4HsFZ7T2EsEEOLFDw1o-/view?usp=sharing.

```
import pandas as pd
import os
import numpy as np
import pandas as pd
import sklearn
import swifter
import numpy as np
import lightgbm
from sklearn.model_selection import train_test_split
import joblib
import scipy as sp
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

from sklearn.linear_model import LogisticRegression, Ridge, RidgeClassifier

from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

import catboost

rheed_long = pd.read_csv("ONTI_Dataset_parsed.csv")
```

```
def df_comma_to_float(df_r):
    for c in df_r.columns:
        if "Time" in c:
            continue
        df_r[c] =
        ↪  df_r[c].astype(str).str.replace(',','.').astype(float).replace([np.inf,
        ↪  -np.inf], np.nan)
        df_r[c] = df_r[c].astype("float64")
    return df_r

rheed_long = df_comma_to_float(rheed_long)
rheed_long.to_csv("ONTI_Dataset_parsed.csv")
rheed_long.Length.plot()
```




```
rheed_long_old_parsed.columns
```

```
Index(['Unnamed: 0', 'Coolness_RHEED', 'Length_RHEED', 'R FWHM_RHEED',
    'Speed_RHEED', 'Time_RHEED', 'X FWHM_RHEED', 'Y FWHM_RHEED',
    'Crystal Position', 'Displayed Rate', 'Filtered Rate',
    'Fundamental Freq', 'Length', 'Raw Rate', 'Source Power', 'Speed',
    'Time'],
   dtype='object')
```

```
#проверка на корректную склейку
print(rheed_long.shape)
rheed_long[:1]
```

```
(2363772, 17)
```

	Unnamed: 0	Coolness_RHEED	Length_RHEED	R FWHM_RHEED	Speed_RHEED	Time_RHEED	X FWHM_RHEED	Y FWHM_RHEED	Crystal Position	Displayed Rate	Filtered Rate	Fundamental Freq	Length	Raw Rate	Source Power	Speed	Time
0	0	0.0	0.003721	0.0	0.0406	2021-02-23 02:21:05	0.0	0.0	1.0	6.027	6.0	inf	0.003721	5.99	21.64	NaN	2021-02-23 02:21:05

```
#rheed_long = df_comma_to_float(rheed_long)
rheed_long.shape
```

```
(2363772, 17)
```

```
#важные колонки для использования
val_col = ['Coolness_RHEED',  'R FWHM_RHEED',  'X FWHM_RHEED', 'Y FWHM_RHEED',
        'Filtered Rate', 'Displayed Rate', 'Raw Rate',
       'Source Power', 'Crystal Position']

#создаем датасет разными аггрегационными функциями
def generated_dataset(series, wind_size=5):
    result = series.rolling(wind_size).agg({"mean": "mean", "std":"std","var":"var"})
    print(result.shape)
    delta = series - series.shift(wind_size)
    print(delta.shape)
    result = result.join(delta)
    result.rename(columns={c:str(c)+"_"+str(wind_size) for c in result.columns},
    ↪  inplace=True)

    print(result.shape)

    return result

df_X = generated_dataset(rheed_long[val_col])
df_X.shape, df_X.columns
```

```
(2363772, 27)
(2363772, 9)
(2363772, 36)

((2363772, 36),
 Index(['('mean', 'Coolness_RHEED')_5', '('mean', 'R FWHM_RHEED')_5',
        '('mean', 'X FWHM_RHEED')_5', '('mean', 'Y FWHM_RHEED')_5',
        '('mean', 'Filtered Rate')_5', '('mean', 'Displayed Rate')_5',
        '('mean', 'Raw Rate')_5', '('mean', 'Source Power')_5',
        '('mean', 'Crystal Position')_5', '('std', 'Coolness_RHEED')_5',
        '('std', 'R FWHM_RHEED')_5', '('std', 'X FWHM_RHEED')_5',
        '('std', 'Y FWHM_RHEED')_5', '('std', 'Filtered Rate')_5',
        '('std', 'Displayed Rate')_5', '('std', 'Raw Rate')_5',
        '('std', 'Source Power')_5', '('std', 'Crystal Position')_5',
        '('var', 'Coolness_RHEED')_5', '('var', 'R FWHM_RHEED')_5',
        '('var', 'X FWHM_RHEED')_5', '('var', 'Y FWHM_RHEED')_5',
        '('var', 'Filtered Rate')_5', '('var', 'Displayed Rate')_5',
        '('var', 'Raw Rate')_5', '('var', 'Source Power')_5',
        '('var', 'Crystal Position')_5', 'Coolness_RHEED_5', 'R FWHM_RHEED_5',
        'X FWHM_RHEED_5', 'Y FWHM_RHEED_5', 'Filtered Rate_5',
        'Displayed Rate_5', 'Raw Rate_5', 'Source Power_5',
        'Crystal Position_5'],
       dtype='object'))
```

```
df_X =
↪  df_X.join(generated_dataset(rheed_long[val_col],wind_size=50))#,rsuffix="_100_")
df_X.shape
```

```
(2363772, 27)
(2363772, 9)
(2363772, 36)

(2363772, 72)
```

```
df_X =
↪  df_X.join(generated_dataset(rheed_long[val_col],wind_size=500))#,rsuffix="_500_")
df_X.shape
```

```
(2363772, 27)
(2363772, 9)
(2363772, 36)

(2363772, 108)
```

```
df_X =
↪  df_X.join(generated_dataset(rheed_long[val_col],wind_size=1500))#,rsuffix="_1500_")
df_X.shape
```

```
(2363772, 27)
(2363772, 9)
(2363772, 36)

(2363772, 108)
```

```
df_X =
↪  df_X.join(generated_dataset(rheed_long[val_col],wind_size=1500))#,rsuffix="_1500_")
df_X.shape
```

```
(2363772, 27)
(2363772, 9)
(2363772, 36)

(2363772, 144)
```

```
#df_X =
↪  df_X.join(generated_dataset(rheed_long[val_col],wind_size=100),rsuffix="_100_")

mask_base = df_X.isna().sum(axis=1)==0
#df_X.fill(inplace=True)
df_X.shape
```

```
(2363772, 144)
```

```
mean_coolness = rheed_long.Coolness_RHEED.rolling(50).mean()
y1 = ((mean_coolness - mean_coolness.shift(-50))>5)
y1.value_counts()
```

```
False    2343612
True       20160
Name: Coolness_RHEED, dtype: int64
```

```
import random
from sklearn.svm import LinearSVC, SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
wind_size = 435

df = df_X[wind_size:-wind_size]

#y = (rheed_long.Coolness < rheed_long.Coolness.shift(wind_size) )
#y = y[wind_size:-wind_size]
cool_col = '(\'mean\', \'Coolness_RHEED\')_50'
y = ((df[cool_col] - df[cool_col].shift(-wind_size))>5)
index_list = random.sample([ i for i in y[(y==1)].index.tolist() if i > 10001],18000)
```

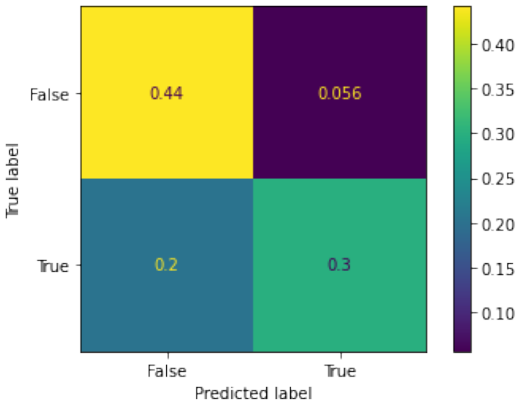
```
index_list+=random.sample([ i for i in y[(y==0)].index.tolist() if i > 10001],25000)
df = df.loc[index_list]
y = y.loc[index_list]
mask = mask_base #&(y>30)
df = df[mask]
y=y[mask]
X_train, X_test,y_train,y_test = train_test_split(df,y,test_size=0.25, shuffle=True)
from  sklearn.linear_model import Ridge
model_cols = [c for c in X_train.columns if c not in ["index","Time"]]
lg = make_pipeline(StandardScaler(),
↪  sklearn.linear_model.LogisticRegression(C=0.0006))

lg.fit(X_train[model_cols], y_train)
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
    ('logisticregression', LogisticRegression(C=0.0006))])
```

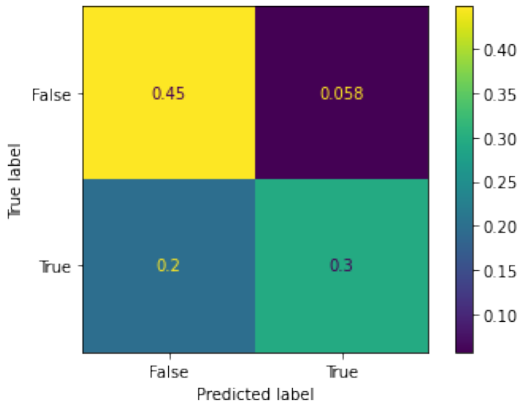
```
import pickle
with open("sandbox/model.pkl","wb") as f:
    pickle.dump(lg,f)

y_test.value_counts()
```

```
True     4278
False    4222
Name: ('mean', 'Coolness_RHEED')_50, dtype: int64
```

```
sklearn.metrics.plot_confusion_matrix(lg, X_train[model_cols],
↪  y_train,normalize='all')  # doctest: +SKIP
plt.show()  # doctest: +SKIP
```




```
sklearn.metrics.plot_confusion_matrix(lg, X_test[model_cols], y_test,normalize='all')
↪  # doctest: +SKIP
plt.show()  # doctest: +SKIP
```




```
import copy
#lg = lgs[2]
list_val= lg[1].coef_[0]
#list_val = copy.deepcopy(lg[1].coef_)
#list_val = list_val[0]
top_five_ind = [(list_val[np.argmax(np.abs(list_val)
↪  )],X_train[model_cols].columns[np.argmax(np.abs(list_val))] )]
list_val[np.argmax(np.abs(list_val))] = 0
for k in range(55):
    top_five_ind.append((list_val[np.argmax(np.abs(list_val))],
                         X_train.columns[np.argmax(np.abs(list_val))]))
    list_val[np.argmax(np.abs(list_val))] = 0

top_five_ind[:5]
```

```
[(0.1602192022356989, "('std', 'Coolness_RHEED')_500"),
(0.14144279522376765, "('std', 'Coolness_RHEED')_50"),
(0.13916986209265583, "('std', 'Y FWHM_RHEED')_500"),
(0.1299507334114571, "('std', 'Y FWHM_RHEED')_50"),
(0.1296147664780099, "('std', 'R FWHM_RHEED')_50")]
```

```
sklearn.metrics.f1_score(y_test, lg.predict_proba(X_test[model_cols])[:,1]>0.36),
↪  sklearn.metrics.precision_score(y_test, lg.predict(X_test[model_cols])),
↪  sklearn.metrics.recall_score(y_test, lg.predict(X_test[model_cols]))
```

```
(0.6618454533279765, 0.48812377424275444, 0.9061488673139159)
```

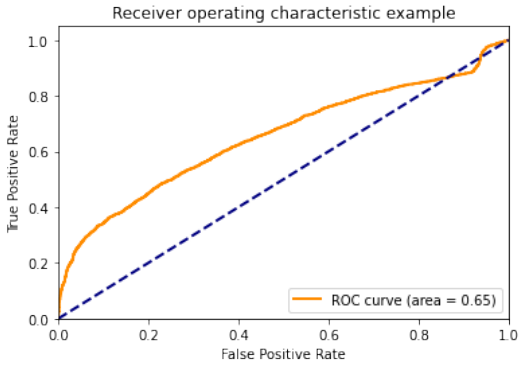
```
sklearn.metrics.f1_score(y_test, lg.predict_proba(X_test[model_cols])[:,1]>0),
↪  sklearn.metrics.precision_score(y_test, lg.predict(X_test[model_cols])),
↪  sklearn.metrics.recall_score(y_test, lg.predict(X_test[model_cols]))
```

```
(0.6616702355460385, 0.48812377424275444, 0.9061488673139159)
```

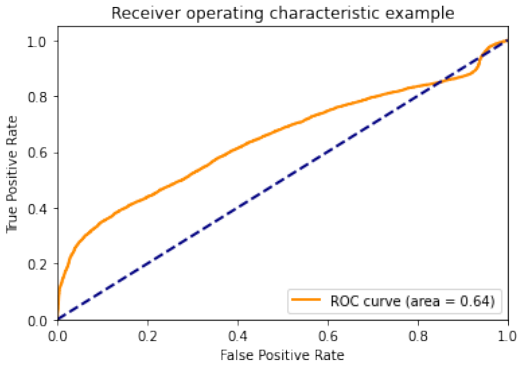
```
sklearn.metrics.f1_score(y_train, lg.predict_proba(X_train[model_cols])[:,1]>0.4)#,
↪  sklearn.metrics.precision_score(y_train, lg.predict(X_train[model_cols]))#,
↪  sklearn.metrics.recall_score(y_test, lg.predict(X_test[model_cols]))
```

```
0.6676447052547345
```

```
fpr, tpr, _ = sklearn.metrics.roc_curve(y_test,
↪  lg.predict_proba(X_test[model_cols])[:,1])
roc_auc = sklearn.metrics.roc_auc_score(y_test,
↪  lg.predict_proba(X_test[model_cols])[:,1])
plt.figure()
lw = 2
plt.plot(fpr, tpr,color='darkorange', lw=lw, label='ROC curve (area = %0.2f)' %
↪  roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```




```
fpr, tpr, _ = sklearn.metrics.roc_curve(y_train,
↪  lg.predict_proba(X_train[model_cols])[:,1])
roc_auc = sklearn.metrics.roc_auc_score(y_train,
↪  lg.predict_proba(X_train[model_cols])[:,1])
plt.figure()
lw = 2
plt.plot(fpr, tpr,color='darkorange', lw=lw, label='ROC curve (area = %0.2f)' %
↪  roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```




```
with open("sandbox/col_list.txt","w") as f:
    f.write(str(X_test[model_cols].columns.tolist()))
```

Всем участникам накануне соревнований предоставляются ключи доступа по ssh к виртуальным машинам “Крок”, на которых можно проводить все вычисления. Каждой команде предоставляется виртуальная машина для вычислений на время соревнования.

Приемка решения осуществляется на платформе https://sim.avt.global/. Организаторы поддерживают приемку решений в течение рабочего времени согласно расписанию, в другое время автоматическая приемка решений не поддерживается.

Участники имеют право на:

- Получения набора данных, необходимых для решения задачи.
- Получение пояснений по данным задачи. Происходят ежедневно в течении финала согласно расписанию.
- Отправку своих решений.
- Обработку данных в удобной им среде, включая предоставленные организаторами серверы.
- Использование интернета, любых курсов и литературы для решения задачи.
- Использование своего ПО для разработки на python.
- Использование библиотек на языке python, включая собственные. Библиотеки, которых нет на принимающем сервере необходимо загружать вместе с решением. При отсутствии возможности загрузить используемую в решении библиотеку (например из-за объема библиотеки) ее использовать в финальном решении нельзя.

Оценка решений

Баллы за задачу начисляются следующим образом:

1. За промежуточный результат — 15%.
2. За финальной результат — 85%.

Порядок подсчета командных очков:

1. Каждой команде начисляется количество баллов, равное отношению наименьшей ошибки на публичных данных теста на 24 февраля к ошибке их решения, это отношение умножается на 15, это результат пункта 1.

2. Каждой команде начисляется количество баллов, равное отношению наименьшей ошибки на всех данных теста на 26 февраля к ошибке их решения, это отношение умножается на 85, это результат пункта 2
3. Сумма пунктов 1 и 2 — это финальный балл команды (не меньше 0 и не больше 100).

Формула подсчета командного балла.

Ошибкой считается 1-roc_auc_score команды. Где roc_auc_score — метрика roc_auc на тестовых данных, этот результат отображается в лидерборде

$$\frac{<\text{наименьшая достигнутая ошибка на 24.02 10:00}>}{<\text{ошибка команды на 24.02 10:00}>} \cdot 15 + \\ + \frac{<\text{наименьшая достигнутая ошибка}>}{<\text{ошибка команды}>} \cdot 85$$