План

1. Напоминание: переобучение, регуляризация и кросс-валидация
2. Pipeline решения ML задачи
3. Подбор гиперпараметров и ансамблирование моделей на примере размеченных данных

Переобучение и методы борьбы с ним

Переобучение --- это одна из главных проблем, с которыми сталкиваются модели машинного обучения. Эффект переобучения состоит в том, что модель, подстраиваясь под обучающую выборку, "обращает внимание" на закономерности в выборке, которые не проявляются в общем случае и имеют характер совпадения.

Переобучение проявляется тем больше, чем больше степеней свободы имеет модель.

Примеры:

- Решающее дерево со слишком большой глубиной может идеально подстроиться под обучающую выборку. Параметры решающего дерева --- это решающие правила во всех вершинах (всего $\sim 2^n$ решающих правил в дереве глубины n).
- Алгоритм k ближайших соседей подстраивается под обучающую выборку и может рассматриваться как эталонный пример переобучения: любой локальный шум в данных приведёт к ошибке. Параметрами Knn являются **все элементы обучающей выборки**.
- Линейный алгоритм, построенный для выборки слишком маленького размера со слишком большим количеством признаков неизбежно переобучится под обучающую выборку.

Во всех примерах мы видим, что при слишком большой сложности модели по сравнению с количеством элементов в обучающей выборке переобучение имеет место. Строго говоря, **переобучение есть всегда, когда имеет место принятие решения в условии неполных данных**, то есть всегда в машинном обучении.

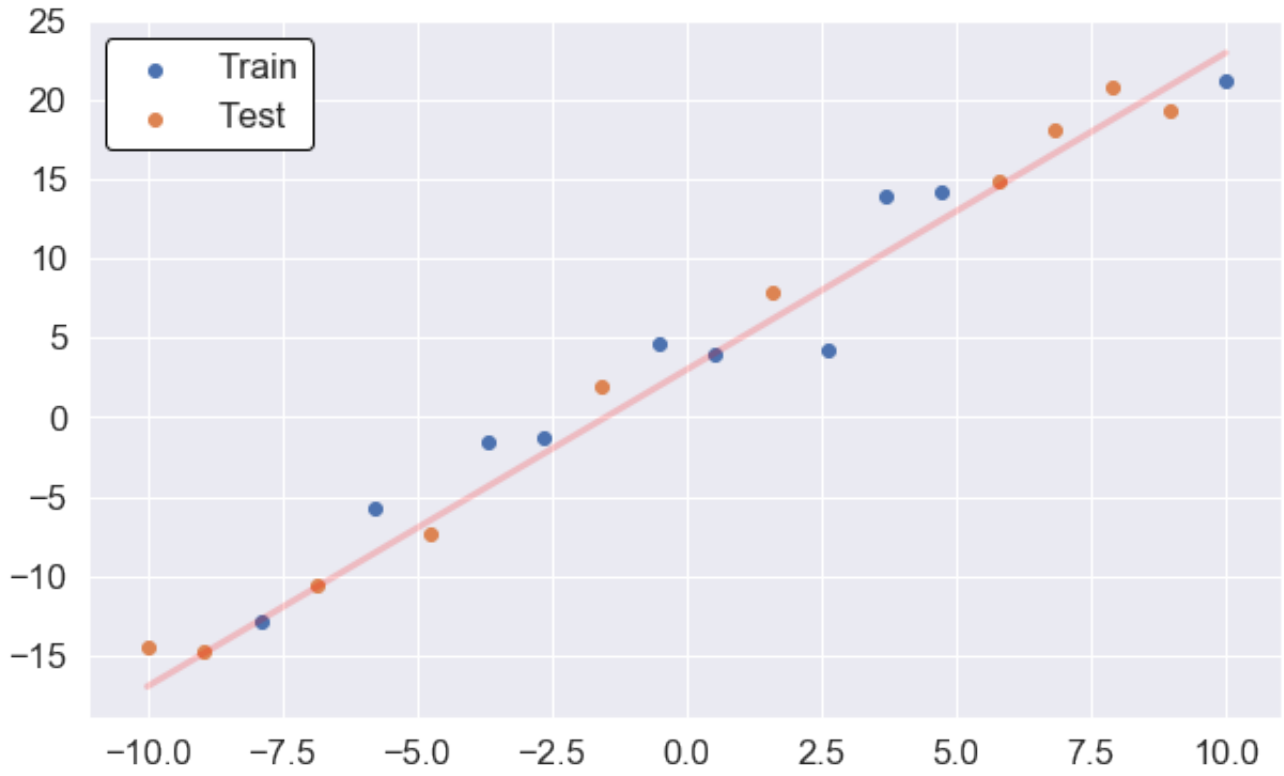
Чтобы нивелировать эффект переобучения, нужно поймать момент, в который качество на тестовой выборке начинает увеличиваться с ростом сложности модели. В этот момент обучение стоит останавливать.

Переобучение многочленов

Смоделируем переобучение сколонность к переобучению полиномиальной зависимости с ростом степени многочлена. Сгенерируем искусственные данные из линейной зависимости с шумом и восстановим зависимость с помощью многочленов степени 1, 3, 4, 7.

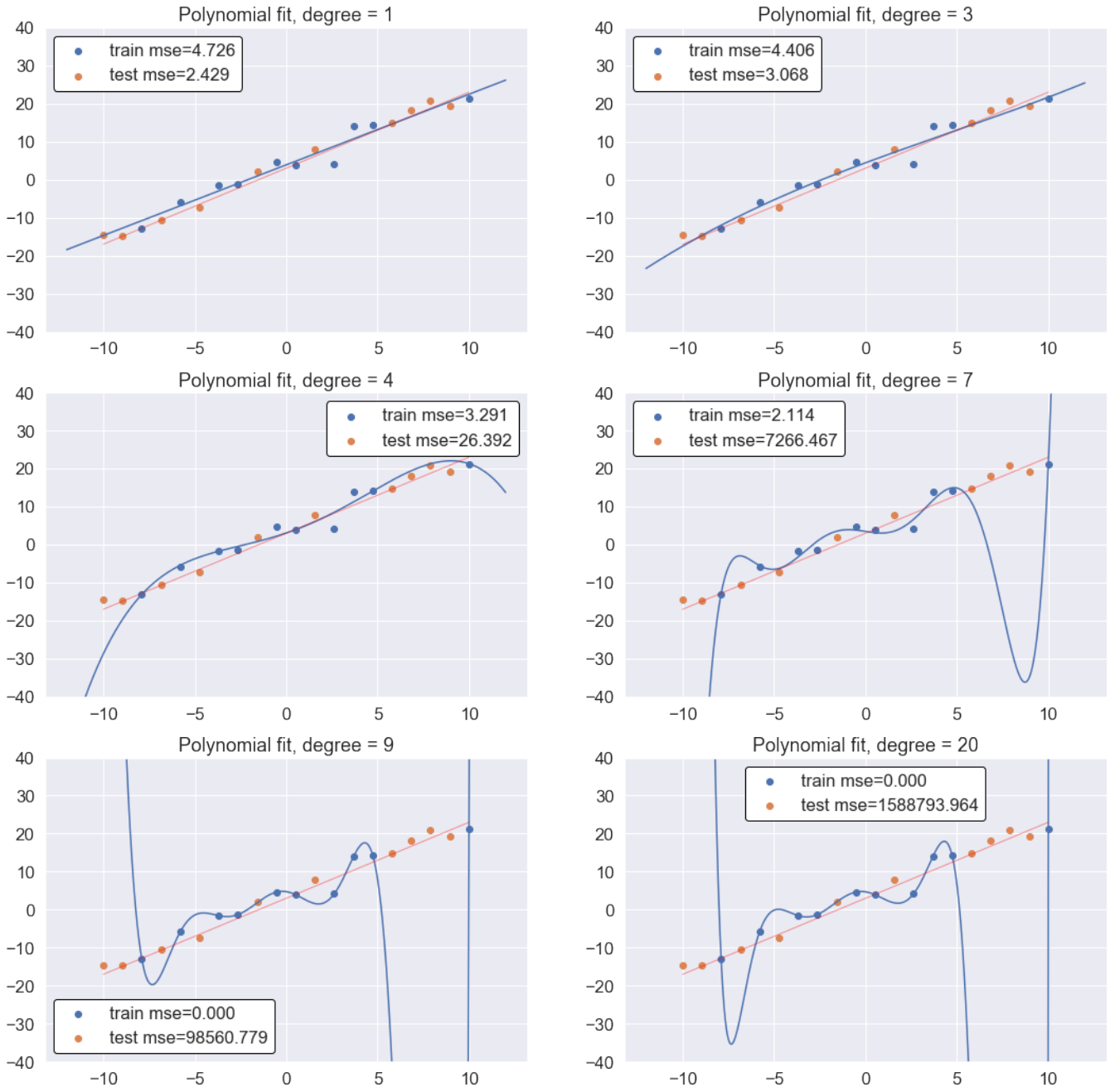
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

import seaborn as sns
sns.set(font_scale=1.5)

import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

```
X = np.linspace(-10, 10, 20)

y = 2 * X + 3 + np.random.randn(20) * 3

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.5, random
plt.figure(figsize=(10,6))
plt.scatter(X_train,y_train, label='Train')
plt.scatter(X_test,y_test, label='Test')
plt.plot(X, 2 * X + 3, color='red', lw=3, alpha = 0.2)
legend_box = plt.legend(framealpha=1).get_frame()
legend_box.set_facecolor("white")
legend_box.set_edgecolor("black")
plt.show()
```




```
grid = np.linspace(-12, 12, 500)

fig, ax = plt.subplots(3, 2, figsize=(18,18))

ax = ax.ravel()

for i, deg in enumerate([1,3,4,7,9, 20]):

    poly = np.polyfit(X_train, y_train, deg)
    ax[i].set_title('Polynomial fit, degree = ' + str(deg))
    ax[i].scatter(X_train,y_train,
                  label='train mse={:.3f}'.format(mean_squared_error(y_train, np
```

```
    ax[i].scatter(X_test,y_test,
                  label='test mse={:.3f}'.format(mean_squared_error(y_test, np.p
    ax[i].set_ylim(-40, 40)
    ax[i].plot(grid, np.polyval(poly, grid))

    ax[i].plot(X, 2 * X + 3, color='red', lw=1, alpha = 0.4)
    legend_box = ax[i].legend(framealpha=1).get_frame()
    legend_box.set_facecolor("white")
    legend_box.set_edgecolor("black")
fig.show()
```



$||x - y||$, x, y - векторы в 2-мерном пространстве

$$||x - y|| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

$||x - y||$, x, y - векторы в n-мерном пространстве

$$||x - y|| = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2}$$

Идея 1: регуляризация

Суть регуляризации состоит в том, чтобы добавлять к функции потерь слагаемое, ограничивающее рост весов модели. Например, обычная версия линейной регрессии выглядит так:

$$\frac{\sum_{i=1}^{\ell} \left|\left|\langle x^i, w\rangle - y^i\right|\right|^2}{\ell} \to \min_w.$$

Регуляризованная версия:

$$\frac{\sum_{i=1}^{\ell} \left|\left|\langle x^i, w\rangle - y^i\right|\right|^2}{\ell} + \frac{1}{C}||w||^2 \to \min_w.$$

Такая версия линейной регресси называется Ridge-регрессией.
Есть также Lasso-регрессия и ElasticNet.

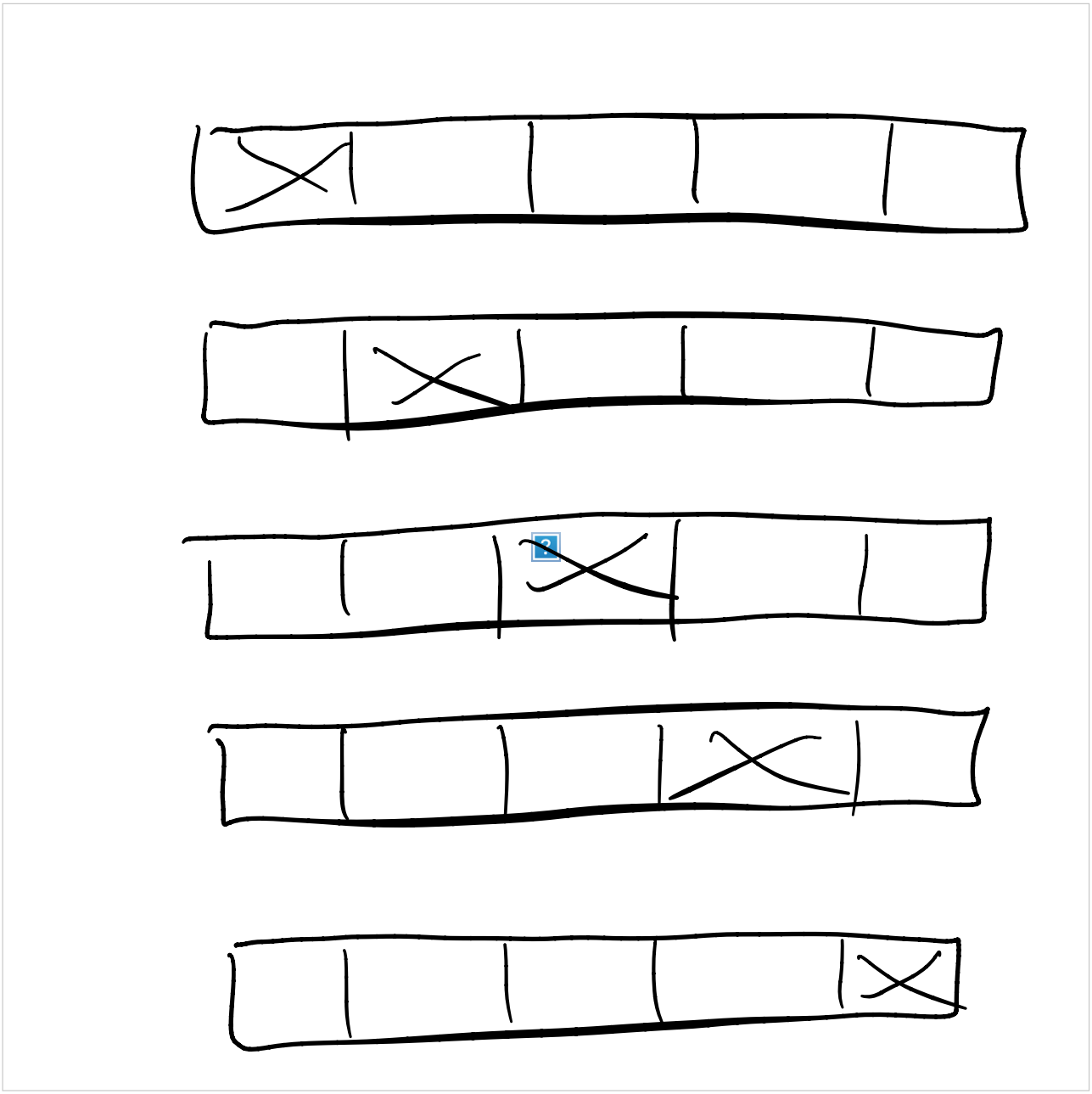
Обычная версия логрегрессии:

$$-\frac{1}{\ell}\left(\sum_{y^i=1} \ln \sigma(\langle x, w\rangle) + \sum_{y^i=-1} \ln(1 - \sigma(\langle x, w\rangle))\right) \to \min_w$$

Регуляризованная версия:

$$-\frac{1}{\ell}\left(\sum_{y^i=1} \ln \sigma(\langle x, w\rangle) + \sum_{y^i=-1} \ln(1 - \sigma(\langle x, w\rangle))\right) + \frac{1}{C}||w||^2 \to \min_w$$

Идея 2: кросс-валидация



Картинка говорит сама за себя. Чтобы получить более стабильное предсказание и точно увидеть переобучение, можно использовать кросс-валидацию. Это ещё пригодится дальше в ноутбуке.

Pipeline решения ML-задачи

Выбор оптимальной модели

Теперь мы потренируемся обучению, оценке и валидации моделей, подбору оптимальных гиперпараметров, смешиванию моделей. Вам предлагается решить задачу бинарной классификации, а именно построить алгоритм, определяющий превысит ли средний заработок человека порог $50k.

```
import pandas as pd


df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/adul
# Назначаем имена колонок
columns = ('age workclass fnlwgt education educ-num marital-status occupation re
           'race sex capital-gain capital-loss  hours-per-week native-country sa

numeric_indices = np.array([0, 2, 4, 10, 11, 12])
categorical_indices = np.array([1, 3, 5, 6, 7, 8, 9, 13])

df.columns = columns.split() #этот метод разделит датасет по колонкам как в масс

df = df.replace('?', np.nan)

df = df.dropna()

df['salary'] = df['salary'].apply((lambda x: x=='>50K')) # Будем предсказывать 1
```

```
numeric_data = df[df.columns[numeric_indices]]

categorial_data = df[df.columns[categorical_indices]]
categorial_data.head()
```

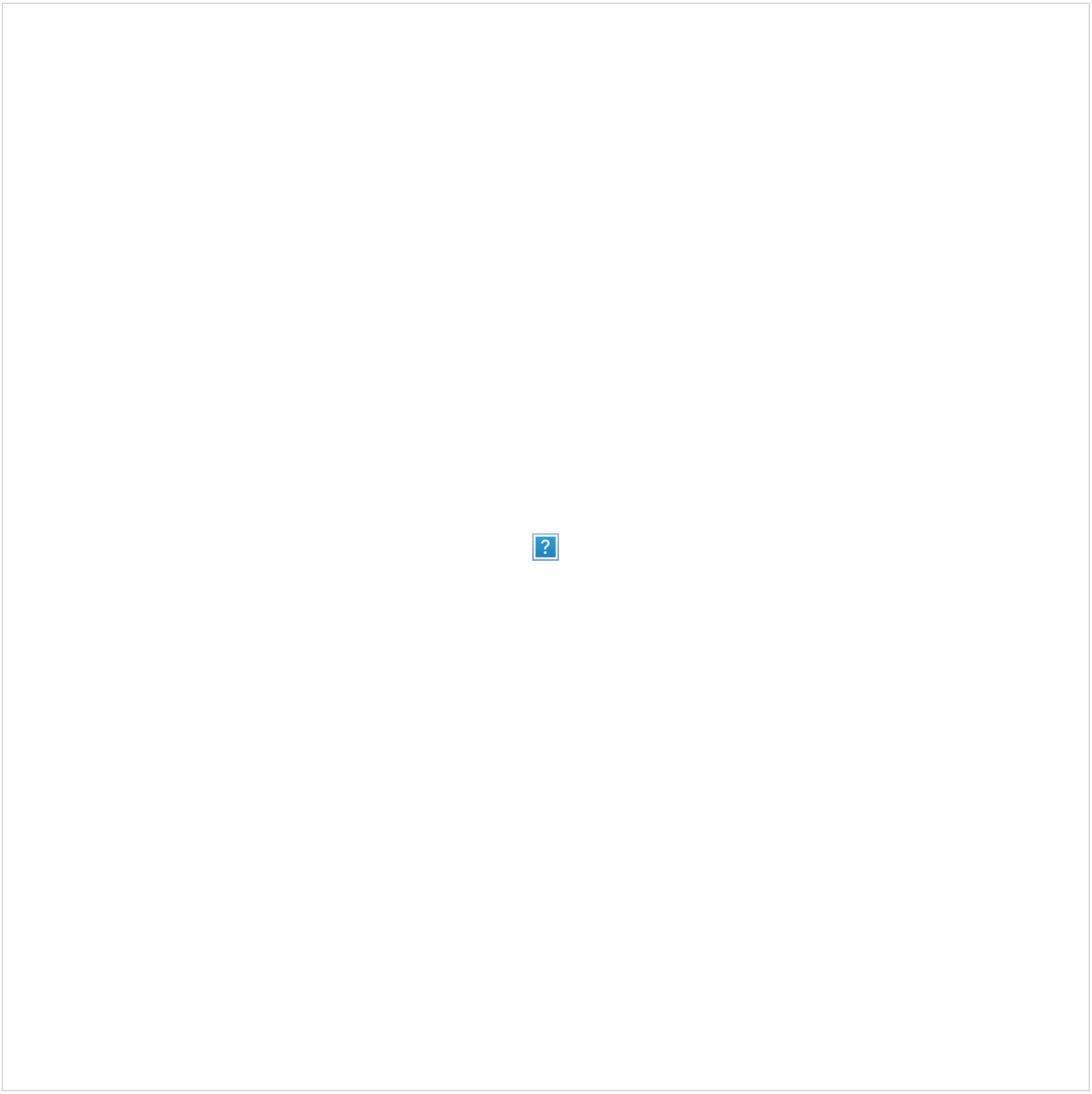
	workclass	education	marital-status	occupation	relationship	race	sex	na co
0	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	
1	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	
2	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	

```
df['education'].unique(), len(df['education'].unique())
```

```
(array(['Bachelors', 'HS-grad', '11th', 'Masters', '9th', 'Some-college',
        'Assoc-acdm', '7th-8th', 'Doctorate', 'Assoc-voc', 'Prof-school',
        '5th-6th', '10th', 'Preschool', '12th', '1st-4th'], dtype=object),
 16)
```

One-hot кодирование

Поскольку все алгоритмы машинного обучения, которые мы изучили, работают лишь с числовыми признаками, необходимо придумать способ обработки категориальных признаков, переводящий их в числовые. Одним из способов сделать это является One-hot кодирование. Его суть состоит в следующем. Пусть некоторая категориальная переменная (скажем, color) принимает n различных значений (Red, Yellow, Green). Тогда можно создать n новых переменных, соответствующих различным значениям категориального признака, каждая из которых равна 1 в том случае, если изначальный категориальный признак принимает такое значение, и 0 иначе. Принцип работы иллюстрирован на картинке.



В Pandas One-hot кодирование выполняется функцией pd.get_dummies. Сгенерируем One-hot признаки для нашего датасета. Сохраним полную матрицу объекты признаки в переменную X.

```
dummy_features = pd.get_dummies(categorial_data)
```

```
X = pd.concat([numeric_data, dummy_features], axis=1)
X_origin = df.iloc[:, :-1]
X.head()
```

	age	fnlwgt	educ-num	capital-gain	capital-loss	hours-per-week	workclass_Federal-gov	workcl
0	50	83311	13	0	0	13	0	
1	38	215646	9	0	0	40	0	
2	53	234721	7	0	0	40	0	
3	28	338409	13	0	0	40	0	
4	37	284582	14	0	0	40	0	

5 rows × 104 columns

```
y = df['salary']
```

```
X.shape, X_origin.shape
```

```
((30161, 104), (30161, 14))
```

Теперь всё готово для обучения алгоритмов.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X.values, y.values,
                                                    train_size=0.8,
                                                    random_state=42)
```

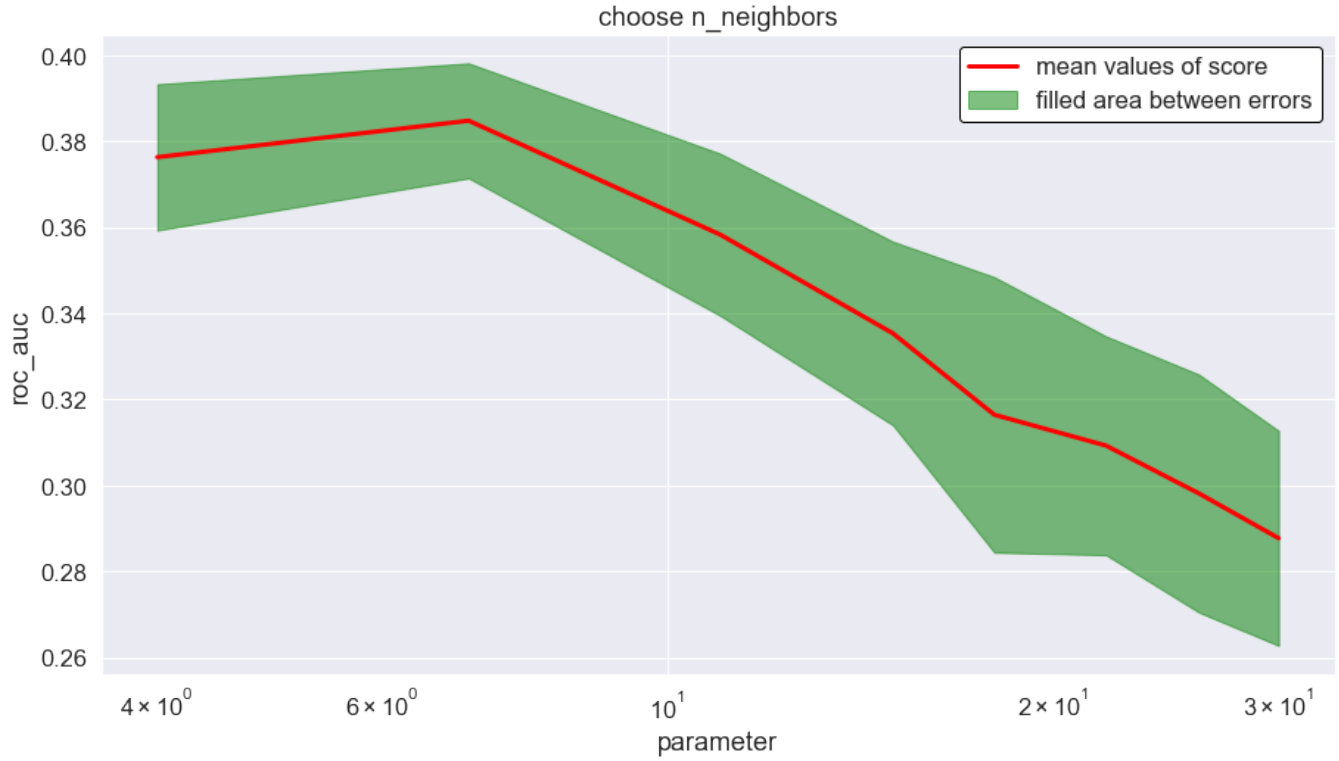
Напишем функцию, визуализирующую поиск оптимального гиперпараметра модели по сетке. Используем идею кросс-валидации.

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

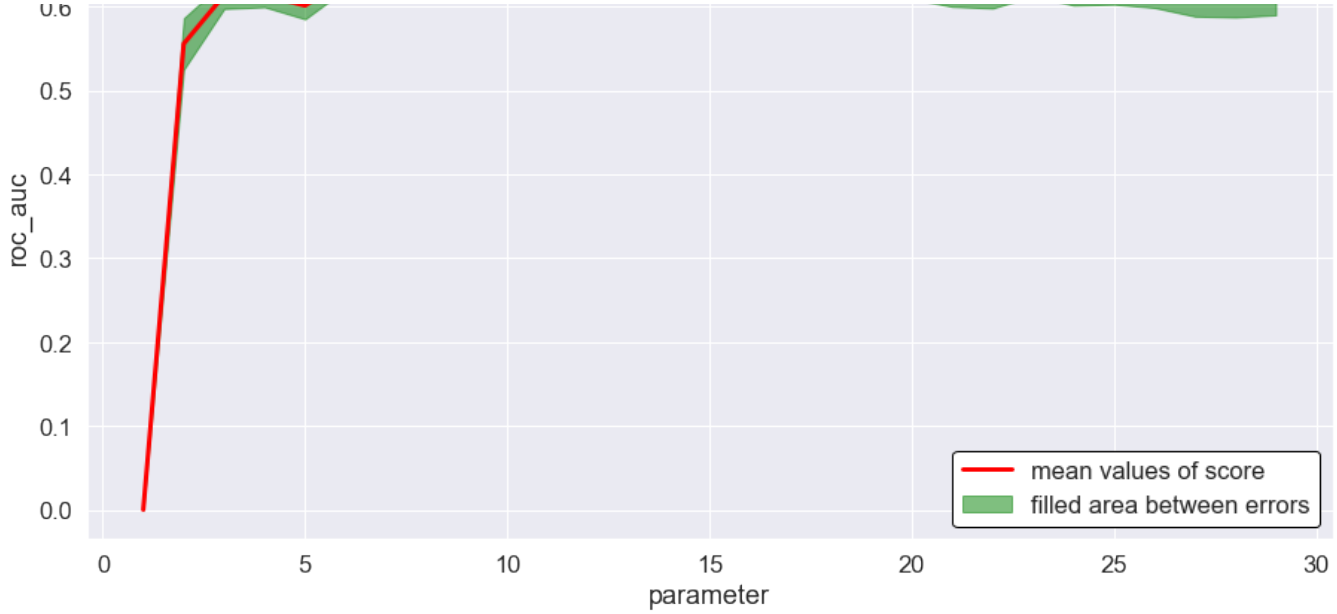
```
def search_and_draw(X, y, model, param_name, grid, param_scale='ordinary', draw=
    parameters = {param_name: grid}

    CV_model = GridSearchCV(estimator=model,
                            param_grid=parameters,
                            cv=5,
                            scoring='f1',
                            n_jobs=-1,
                            verbose=10)
    CV_model.fit(X, y)
    means = CV_model.cv_results_['mean_test_score']
    error = CV_model.cv_results_['std_test_score']

    if draw:
        plt.figure(figsize=(15,8))
        plt.title('choose ' + param_name)


        if (param_scale == 'log'):
            plt.xscale('log')

        plt.plot(grid, means, label='mean values of score', color='red', lw=3)

        plt.fill_between(grid, means - 2 * error, means + 2 * error,
                         color='green', label='filled area between errors', alph
        legend_box = plt.legend(framealpha=1).get_frame()
        legend_box.set_facecolor("white")
        legend_box.set_edgecolor("black")
        plt.xlabel('parameter')
        plt.ylabel('roc_auc')
        plt.show()

    return means, error


from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier


models = [KNeighborsClassifier(), DecisionTreeClassifier()]
param_names = ['n_neighbors', 'max_depth']
grids = [np.array(np.linspace(4, 30, 8), dtype='int'), np.arange(1, 30)]
param_scales = ['log', 'ordinary']


for model, param_name, grid, param_scale in zip(models,
                                                param_names,
                                                grids,
```

```
search_and_draw(X_train, y_train, model, param_name, grid, param_scale)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    3.5s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    5.7s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    6.2s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    9.9s
[Parallel(n_jobs=-1)]: Done  30 out of  40 | elapsed:   10.7s remaining:
[Parallel(n_jobs=-1)]: Done  35 out of  40 | elapsed:   12.7s remaining:
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   13.3s remaining:
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   13.3s finished
```




```
Fitting 5 folds for each of 29 candidates, totalling 145 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    1.9s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:    4.4s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    5.6s
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:    7.3s
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:    8.8s
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed:   10.8s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:   12.8s
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed:   15.2s
[Parallel(n_jobs=-1)]: Done 145 out of 145 | elapsed:   16.7s remaining:
[Parallel(n_jobs=-1)]: Done 145 out of 145 | elapsed:   16.7s finished
```



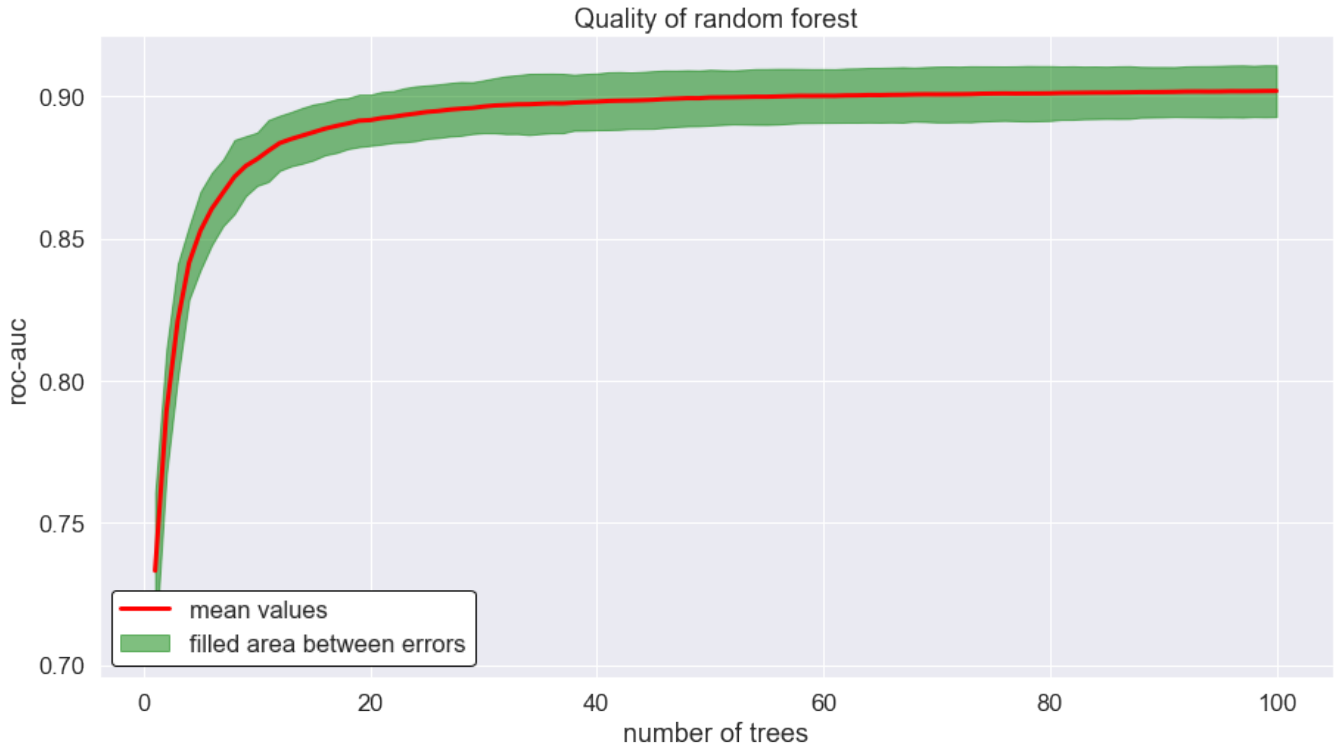
Подберём параметр n_estimators в алгоритме случайный лес. Известно, что случайный лес не переобучается. Поэтому график качества будет монотонно возрастать. Следовательно, необходимо найти минимальное значение n_estimators, при котором качество не изменяется. Поскольку каждое дерево обучается независимо от остальных, достаточно обучить сразу лес из большого количества деревьев, а затем рассмотреть подмножества нужного размера из исходного множества деревьев.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from tqdm.notebook import tqdm


max_trees = 100

values = np.arange(max_trees) + 1

kf = KFold(n_splits=5, shuffle=True, random_state=1234)

global_scores = []

for train_indices, val_indices in tqdm(kf.split(X_train), total=5):
    scores = []

    X_train_kf = X_train[train_indices]
    y_train_kf = y_train[train_indices]

    X_val_kf = X_train[val_indices]
    y_val_kf = y_train[val_indices]

    forest = RandomForestClassifier(n_estimators=max_trees)
    forest.fit(X_train_kf, y_train_kf)
    trees = forest.estimators_

    for number_of_trees in tqdm(values, leave=False):
        thinned_forest = RandomForestClassifier(n_estimators=number_of_trees)

        thinned_forest.n_classes_ = 2
        thinned_forest.estimators_ = trees[:number_of_trees]

        scores.append(roc_auc_score(y_val_kf, thinned_forest.predict_proba(X_val

    scores = np.array(scores)

    global_scores.append(scores)

global_scores = np.stack(global_scores, axis=0)
```

```
HBox(children=(FloatProgress(value=0.0, max=5.0), HTML(value='')))
HBox(children=(FloatProgress(value=0.0), HTML(value='')))
HBox(children=(FloatProgress(value=0.0), HTML(value='')))
HBox(children=(FloatProgress(value=0.0), HTML(value='')))
HBox(children=(FloatProgress(value=0.0), HTML(value='')))
HBox(children=(FloatProgress(value=0.0), HTML(value='')))
```

```
mean_cross_val_score = global_scores.mean(axis=0)
std_cross_val_score = global_scores.std(axis=0)

plt.figure(figsize=(15,8))
plt.title('Quality of random forest')

plt.plot(values, mean_cross_val_score, label='mean values', color='red', lw=3)
plt.fill_between(values,
                 mean_cross_val_score - 2 * std_cross_val_score,
                 mean_cross_val_score + 2 * std_cross_val_score,
                 color='green',
                 label='filled area between errors',
                 alpha=0.5)
legend_box = plt.legend(framealpha=1).get_frame()
legend_box.set_facecolor("white")
legend_box.set_edgecolor("black")
plt.xlabel('number of trees')
plt.ylabel('roc-auc')

plt.show()
```



Нормировка признаков

Нормируем признаки и проделаем тот же эксперимент с алгоритмом ближайших соседей. Посмотрим, изменилось ли качество предсказания.

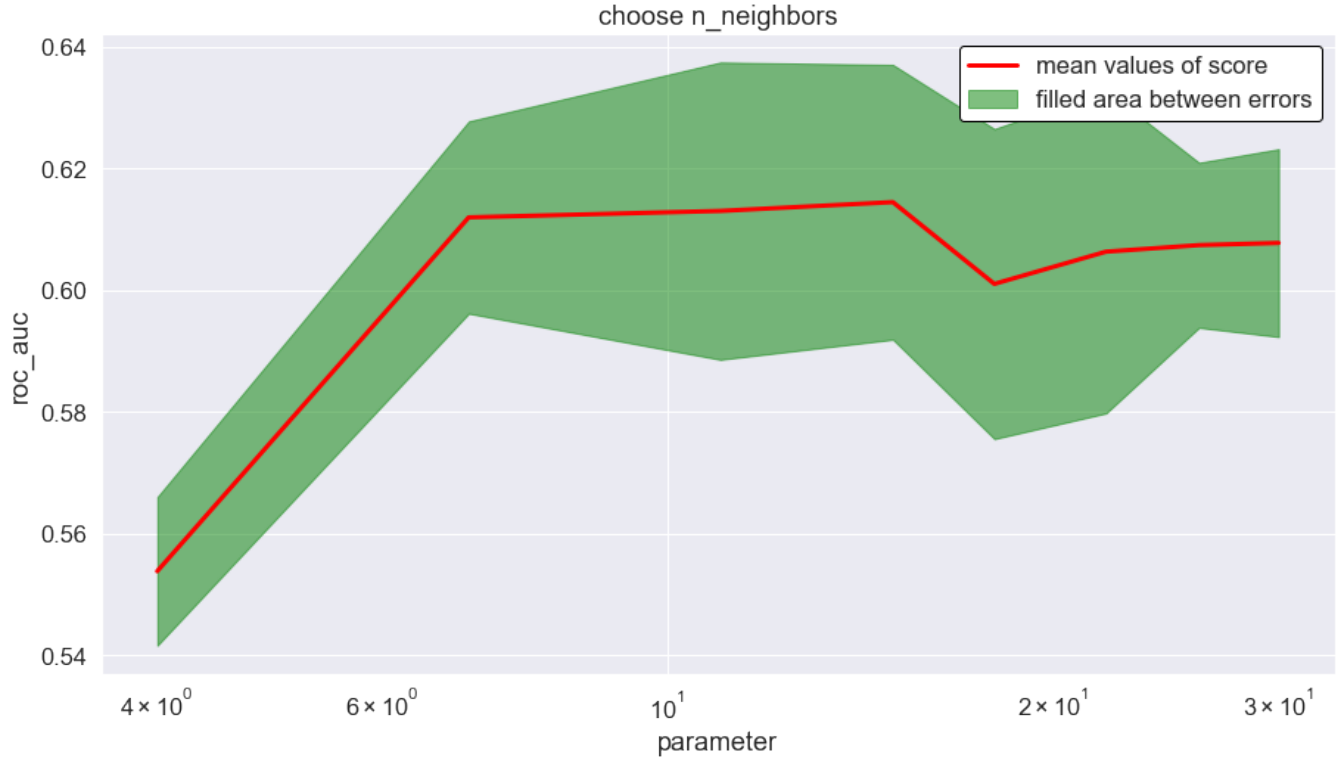
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

`StandardScaler` выполняет преобразование

$$z = \frac{x - \mu}{\sigma}, \text{ где } \sigma \text{ - стандартное отклонение, а } \mu \text{ - среднее}$$

```
search_and_draw(X_train_scaled, y_train, KNeighborsClassifier(), 'n_neighbors',
                np.array(np.linspace(4, 30, 8), dtype='int'), 'log');
```

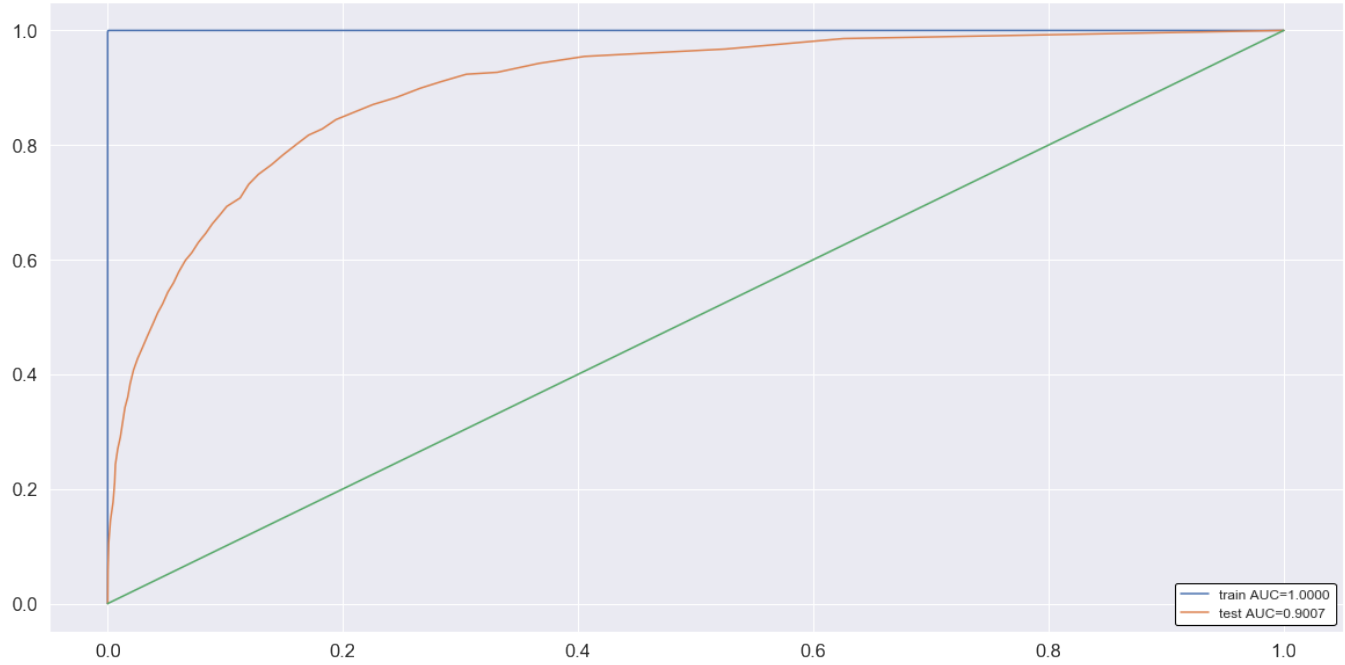
```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   43.3s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done  30 out of  40 | elapsed:  3.0min remaining:  1
[Parallel(n_jobs=-1)]: Done  35 out of  40 | elapsed:  3.7min remaining:
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:  3.7min remaining:
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:  3.7min finished
```



Как и следовало ожидать, ни один из наших алгоритмов не побил случайный лес. Итак, видим, что на больших выборках бэггинг работает. Вычислим итоговое качество на test.

```
model = RandomForestClassifier(n_estimators=50, n_jobs=-1)

model.fit(X_train, y_train)
y_train_predicted = model.predict_proba(X_train)[:, 1]
y_test_predicted = model.predict_proba(X_test)[:, 1]


from sklearn.metrics import roc_auc_score, roc_curve
```

```
train_auc = roc_auc_score(y_train, y_train_predicted)
test_auc = roc_auc_score(y_test, y_test_predicted)

plt.figure(figsize=(20,10))
plt.plot(*roc_curve(y_train, y_train_predicted)[:2], label='train AUC={:.4f}'.fo
plt.plot(*roc_curve(y_test, y_test_predicted)[:2], label='test AUC={:.4f}'.forma
legend_box = plt.legend(fontsize='large', framealpha=1).get_frame()
legend_box.set_facecolor("white")
legend_box.set_edgecolor("black")
plt.plot(np.linspace(0,1,100), np.linspace(0,1,100))
plt.show()
```



Что ещё можно делать:

Мы подбирали оптимальный одномерный параметр для алгоритма. Можно также:

- Искать по сетке не только численные гиперпараметры, но и категориальные, например, метрику в алгоритме ближайших соседей или критерий ветвления в решающем дереве.
- Искать оптимальный параметр по многомерной сетке. Перебрать все возможные варианты здесь не выйдет, потому что на это уйдёт слишком много времени. Зато можно перебирать случайные точки по сетке. Эта процедура называется Grid Random Search.

Стекинг

Идея стекинга состоит в том, чтобы обучать разнообразные алгоритмы и использовать их в качестве новых признаков объектов.

Чтобы избежать переобучения, необходимо разделить обучающую выборку на n фолдов. Для предсказания ответов на k-ом фолде алгоритм обучается на оставшихся n-1 фолдах и предсказывает ответ на k-ом фолде. Такую схему обучения-предсказания реализует функция sklearn.model_selection.cross_val_predict.

```
from sklearn.model_selection import cross_val_predict
```

Будем работать с тем же самым датасетом, что и ранее. Посмотрим, сумеем ли мы побить результаты случайного леса с помощью стекинга.

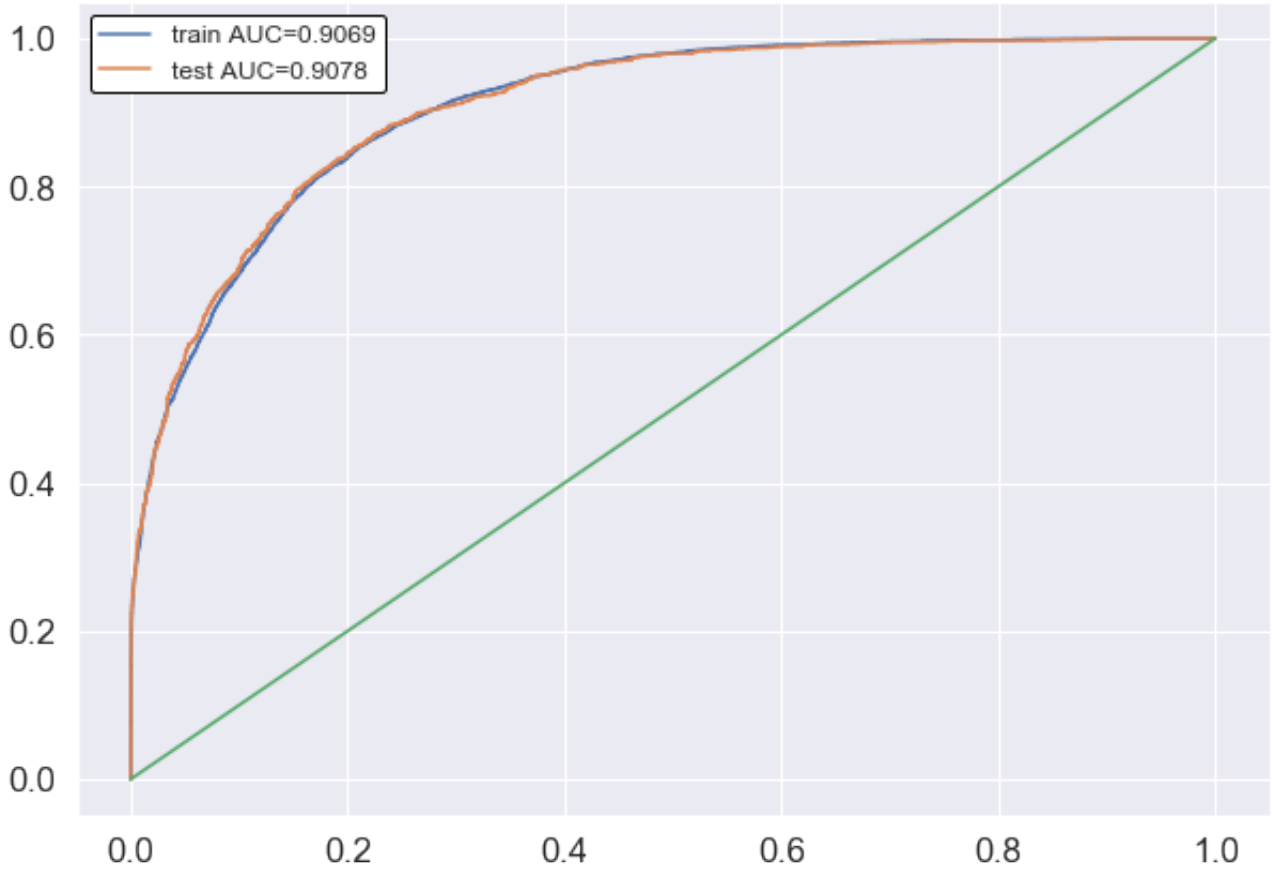
```
def compute_meta_feature(model, X_train, X_test, y_train, cv):
    try:
        train_answers = cross_val_predict(model, X_train, y_train, cv=cv, method
        model.fit(X_train, y_train)
        return train_answers, model.predict_proba(X_test)[:, 1]

    except Exception:
        train_answers = cross_val_predict(model, X_train, y_train, cv=cv, method
        model.fit(X_train, y_train)
        return train_answers, model.predict(X_test)[:, 1]
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier


models = []
models.append(KNeighborsClassifier(n_jobs=-1, n_neighbors=30))
models.append(LogisticRegression())
models.append(RandomForestClassifier(max_depth=3, n_estimators=50, n_jobs=-1))
models.append(RandomForestClassifier(max_depth=7, n_estimators=50, n_jobs=-1))
models.append(DecisionTreeClassifier(max_depth=8))


meta_features_train = np.zeros((X_train.shape[0], 0))
meta_features_test = np.zeros((X_test.shape[0], 0))


for model in tqdm(models):
    train, test = compute_meta_feature(model, X_train, X_test, y_train, 5)
    meta_features_train = np.append(meta_features_train, train.reshape((train.si
    meta_features_test = np.append(meta_features_test, test.reshape((test.size,
```

```
HBox(children=(FloatProgress(value=0.0, max=5.0), HTML(value='')))
```

```
stacking_model = LogisticRegression()
stacking_model.fit(meta_features_train, y_train)

y_train_predicted = stacking_model.predict_proba(meta_features_train)[:, 1]
y_test_predicted = stacking_model.predict_proba(meta_features_test)[:, 1]
```

```
train_auc = roc_auc_score(y_train, y_train_predicted)
test_auc = roc_auc_score(y_test, y_test_predicted)

plt.figure(figsize=(10,7))
plt.plot(*roc_curve(y_train, y_train_predicted)[:2], label='train AUC={:.4f}'.fo
plt.plot(*roc_curve(y_test, y_test_predicted)[:2], label='test AUC={:.4f}'.forma
legend_box = plt.legend(fontsize='large', framealpha=1).get_frame()
legend_box.set_facecolor("white")
legend_box.set_edgecolor("black")
plt.plot(np.linspace(0,1,100), np.linspace(0,1,100))
plt.show()
```

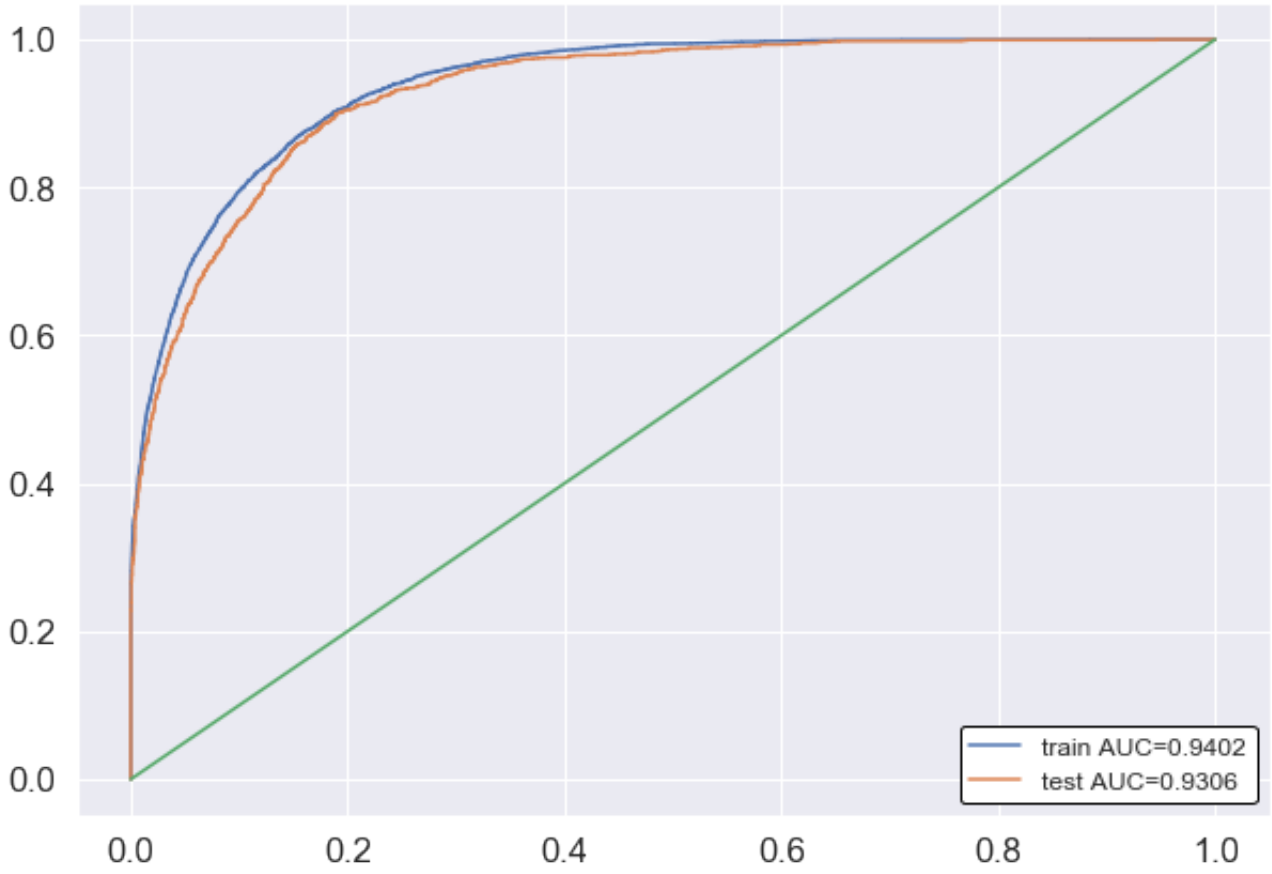


Бустинг

Попробуем в пару-тройку строк побить всё то качество, которое мы так усердно искали.

```
# если этого модуля нет, то нужно раскомментировать следующую строчку и запустит
#!pip install xgboost
```

```
import xgboost


boosting_model = xgboost.XGBClassifier(n_estimators=500)

boosting_model.fit(X_train, y_train)

y_train_predicted = boosting_model.predict_proba(X_train)[:, 1]
y_test_predicted = boosting_model.predict_proba(X_test)[:, 1]


train_auc = roc_auc_score(y_train, y_train_predicted)
test_auc = roc_auc_score(y_test, y_test_predicted)

plt.figure(figsize=(10,7))
plt.plot(*roc_curve(y_train, y_train_predicted)[:2], label='train AUC={:.4f}'.fo
plt.plot(*roc_curve(y_test, y_test_predicted)[:2], label='test AUC={:.4f}'.forma
legend_box = plt.legend(fontsize='large', framealpha=1).get_frame()
legend_box.set_facecolor("white")
legend_box.set_edgecolor("black")
plt.plot(np.linspace(0,1,100), np.linspace(0,1,100))
plt.show()
```



Круто, да? А теперь попробуем "отечественного" производителя - CatBoost от Яндекса.

```
# если этого модуля нет, то нужно раскомментировать следующую строчку и запустит
#!pip install catboost


import catboost # документация: https://catboost.ai/docs


# CatBoost умеет работать с категориальными признаками сам
X_train_origin, X_test_origin, _, _ = train_test_split(X_origin.values, y.values
                                                       train_size=0.8,
                                                       random_state=42)
```

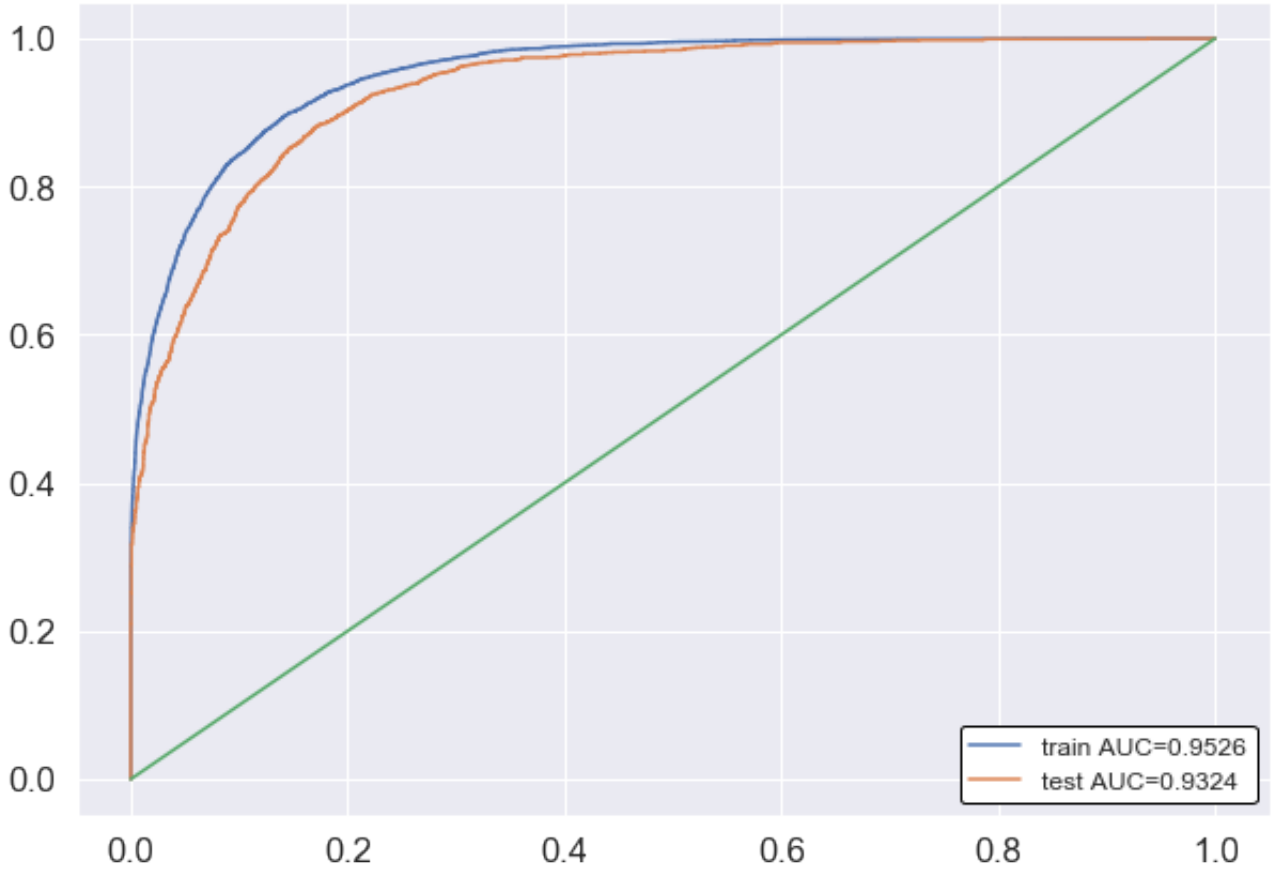
(из документации CatBoost)

```
boosting_model = catboost.CatBoostClassifier(n_estimators=200,
                                             cat_features=categorical_indices)

boosting_model.fit(X_train_origin, y_train)

y_train_predicted = boosting_model.predict_proba(X_train_origin)[:, 1]
y_test_predicted = boosting_model.predict_proba(X_test_origin)[:, 1]
```

```
Learning rate set to 0.175479
0:      learn: 0.5380235        total: 90.4ms   remaining: 18s
1:      learn: 0.4479261        total: 187ms    remaining: 18.5s
2:      learn: 0.3998054        total: 250ms    remaining: 16.4s
3:      learn: 0.3720098        total: 338ms    remaining: 16.5s
4:      learn: 0.3591503        total: 464ms    remaining: 18.1s
5:      learn: 0.3471251        total: 552ms    remaining: 17.9s
6:      learn: 0.3356031        total: 632ms    remaining: 17.4s
7:      learn: 0.3267387        total: 716ms    remaining: 17.2s
8:      learn: 0.3217937        total: 797ms    remaining: 16.9s
9:      learn: 0.3175740        total: 915ms    remaining: 17.4s
10:     learn: 0.3137769        total: 1.02s    remaining: 17.5s
11:     learn: 0.3116092        total: 1.11s    remaining: 17.4s
12:     learn: 0.3087479        total: 1.21s    remaining: 17.5s
13:     learn: 0.3075583        total: 1.28s    remaining: 17s
14:     learn: 0.3051900        total: 1.36s    remaining: 16.8s
15:     learn: 0.3033366        total: 1.44s    remaining: 16.5s
16:     learn: 0.3019215        total: 1.54s    remaining: 16.6s
17:     learn: 0.3012249        total: 1.61s    remaining: 16.3s
18:     learn: 0.3003474        total: 1.66s    remaining: 15.8s
19:     learn: 0.2993494        total: 1.74s    remaining: 15.7s
20:     learn: 0.2971107        total: 1.83s    remaining: 15.6s
```

```
21:     learn: 0.2959619        total: 1.91s    remaining: 15.4s
22:     learn: 0.2950362        total: 1.99s    remaining: 15.3s
23:     learn: 0.2941084        total: 2.08s    remaining: 15.2s
24:     learn: 0.2932537        total: 2.19s    remaining: 15.3s
25:     learn: 0.2922845        total: 2.26s    remaining: 15.1s
26:     learn: 0.2909579        total: 2.35s    remaining: 15.1s
27:     learn: 0.2901457        total: 2.41s    remaining: 14.8s
28:     learn: 0.2896557        total: 2.48s    remaining: 14.6s
29:     learn: 0.2888863        total: 2.56s    remaining: 14.5s
30:     learn: 0.2884196        total: 2.61s    remaining: 14.2s
31:     learn: 0.2876371        total: 2.68s    remaining: 14.1s
32:     learn: 0.2869579        total: 2.78s    remaining: 14.1s
33:     learn: 0.2864635        total: 2.88s    remaining: 14.1s
34:     learn: 0.2862244        total: 2.95s    remaining: 13.9s
35:     learn: 0.2857662        total: 3.03s    remaining: 13.8s
36:     learn: 0.2852480        total: 3.15s    remaining: 13.9s
37:     learn: 0.2849213        total: 3.21s    remaining: 13.7s
38:     learn: 0.2845431        total: 3.28s    remaining: 13.5s
39:     learn: 0.2839791        total: 3.35s    remaining: 13.4s
40:     learn: 0.2836364        total: 3.44s    remaining: 13.4s
41:     learn: 0.2833513        total: 3.52s    remaining: 13.2s
42:     learn: 0.2831114        total: 3.58s    remaining: 13.1s
43:     learn: 0.2828483        total: 3.65s    remaining: 12.9s
44:     learn: 0.2825778        total: 3.72s    remaining: 12.8s
45:     learn: 0.2823769        total: 3.79s    remaining: 12.7s
46:     learn: 0.2820698        total: 3.87s    remaining: 12.6s
47:     learn: 0.2818086        total: 3.93s    remaining: 12.5s
48:     learn: 0.2816115        total: 3.98s    remaining: 12.3s
49:     learn: 0.2812934        total: 4.08s    remaining: 12.2s
50:     learn: 0.2811264        total: 4.15s    remaining: 12.1s
51:     learn: 0.2807356        total: 4.22s    remaining: 12s
52:     learn: 0.2804947        total: 4.3s     remaining: 11.9s
53:     learn: 0.2800889        total: 4.36s    remaining: 11.8s
54:     learn: 0.2797278        total: 4.42s    remaining: 11.6s
55:     learn: 0.2792745        total: 4.49s    remaining: 11.6s
56:     learn: 0.2790923        total: 4.55s    remaining: 11.4s
57:     learn: 0.2786108        total: 4.63s    remaining: 11.3s
58:     learn: 0.2782595        total: 4.74s    remaining: 11.3s
```

```
train_auc = roc_auc_score(y_train, y_train_predicted)
test_auc = roc_auc_score(y_test, y_test_predicted)

plt.figure(figsize=(10,7))
plt.plot(*roc_curve(y_train, y_train_predicted)[:2], label='train AUC={:.4f}'.fo
plt.plot(*roc_curve(y_test, y_test_predicted)[:2], label='test AUC={:.4f}'.forma
legend_box = plt.legend(fontsize='large', framealpha=1).get_frame()
legend_box.set_facecolor("white")
legend_box.set_edgecolor("black")
plt.plot(np.linspace(0,1,100), np.linspace(0,1,100))
plt.show()
```




```
boosting_model = catboost.CatBoostClassifier(n_estimators=200, silent=True,
                                             cat_features=categorical_indices,
                                             eval_metric='AUC')
boosting_model.grid_search({'l2_leaf_reg': np.linspace(0, 1, 20)},
                           X_train_origin,
                           y_train, plot=True, refit=True)
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
0:      loss: 0.9227514 best: 0.9227514 (0)     total: 13.9s    remaining:
1:      loss: 0.9238508 best: 0.9238508 (1)     total: 29.3s    remaining:
2:      loss: 0.9236316 best: 0.9238508 (1)     total: 44.8s    remaining:
3:      loss: 0.9233129 best: 0.9238508 (1)     total: 1m 2s    remaining:
4:      loss: 0.9237572 best: 0.9238508 (1)     total: 1m 15s   remaining:
```

```
5:      loss: 0.9232062 best: 0.9238508 (1)     total: 1m 26s   remaining:
6:      loss: 0.9239117 best: 0.9239117 (6)     total: 1m 40s   remaining:
7:      loss: 0.9231683 best: 0.9239117 (6)     total: 1m 58s   remaining:
8:      loss: 0.9233554 best: 0.9239117 (6)     total: 2m 15s   remaining:
9:      loss: 0.9232125 best: 0.9239117 (6)     total: 2m 38s   remaining:
10:     loss: 0.9232851 best: 0.9239117 (6)     total: 2m 58s   remaining:
11:     loss: 0.9233200 best: 0.9239117 (6)     total: 3m 21s   remaining:
12:     loss: 0.9234769 best: 0.9239117 (6)     total: 3m 57s   remaining:
13:     loss: 0.9236351 best: 0.9239117 (6)     total: 4m 17s   remaining:
14:     loss: 0.9236565 best: 0.9239117 (6)     total: 4m 38s   remaining:
15:     loss: 0.9235567 best: 0.9239117 (6)     total: 4m 59s   remaining:
16:     loss: 0.9233254 best: 0.9239117 (6)     total: 5m 19s   remaining:
17:     loss: 0.9234545 best: 0.9239117 (6)     total: 5m 44s   remaining:
18:     loss: 0.9237148 best: 0.9239117 (6)     total: 6m 11s   remaining:
19:     loss: 0.9233875 best: 0.9239117 (6)     total: 6m 35s   remaining:
Estimating final quality...
{'params': {'l2_leaf_reg': 0.3157894736842105},
 'cv_results': defaultdict(list,
             {'iterations': [0,
               1,
               2,
               3,
               4,
               5,
               6,
               7,
               8,
               9,
               10,
               11,
               12,
               13,
               14,
               15,
               16,
               17,
               18,
               19,
               20,
               21,
               22,
               23,
               24,
               25,
               26,
               27,
               28,
               29,
               30,
               31,
               32,
               33,
               34,
               35
```

```
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59,
60,
61,
62,
63,
64,
65,
66,
67,
68,
69,
70,
71,
72,
73,
74,
75,
76,
77,
78,
79,
80,
81,
82,
83,
84,
85,
86,
87,
88,
89
```

```
 89,
 90,
 91,
 92,
 93,
 94,
 95,
 96,
 97,
 98,
 99,
100,
101,
102,
103,
104,
105,
106,
107,
108,
109,
110,
111,
112,
113,
114,
115,
116,
117,
118,
119,
120,
121,
122,
123,
124,
125,
126,
127,
128,
129,
130,
131,
132,
133,
134,
135,
136,
137,
138,
139,
140,
141,
142,
143
```

```
143,
144,
145,
146,
147,
148,
149,
150,
151,
152,
153,
154,
155,
156,
157,
158,
159,
160,
161,
162,
163,
164,
165,
166,
167,
168,
169,
170,
171,
172,
173,
174,
175,
176,
177,
178,
179,
180,
181,
182,
183,
184,
185,
186,
187,
188,
189,
190,
191,
192,
193,
194,
195,
196,
197
```

```
 197,
 198,
 199],
'test-AUC-mean': [0.8787976610476503,
 0.8852556743471777,
 0.889584073318887,
 0.8899086972629142,
 0.8960222976772979,
 0.8973998637245018,
 0.8986302867994703,
 0.8994069308284751,
 0.9000850206151433,
 0.900331484440093,
 0.9012762965079898,
 0.9022252823207143,
 0.9024863775814671,
 0.9028770678746619,
 0.9032347643027697,
 0.9035579742191248,
 0.9043414410068961,
 0.9047577386026862,
 0.9049681828614614,
 0.9053751577949548,
 0.9055590226340259,
 0.9059074383752339,
 0.9060234933435867,
 0.9060811288356128,
 0.9064905957113921,
 0.9069917025095999,
 0.9071207899022272,
 0.9074414778268157,
 0.9076280397959072,
 0.907653024421674,
 0.9080248397924646,
 0.9082555043190673,
 0.9085039825233117,
 0.9086835703606302,
 0.9088972547386223,
 0.9093203091337939,
 0.9095899463149154,
 0.9097187596233759,
 0.910123572766962,
 0.9103675607156623,
 0.9105592264414164,
 0.9108858667949239,
 0.9111394928297702,
 0.9114054376137223,
 0.9116300876893503,
 0.9118112630292553,
 0.9119603991128254,
 0.9121485204991302,
 0.9123642887817285,
 0.9124695802142693,
 0.9125115612164599,
```

```
0.9128118894853733,
0.9130229923422738,
0.9131946320212876,
0.913324131079042,
0.9135168285195867,
0.9136870042251731,
0.913818509712371,
0.9139191778427166,
0.9140602722199472,
0.9142781016581362,
0.9144480137266947,
0.9145233793238076,
0.9146951323081393,
0.9149780107765667,
0.9151503691481103,
0.9153223468154265,
0.9154520950459911,
0.9155593348528704,
0.9156209236493947,
0.9157886783917201,
0.9159095076980491,
0.9159883879223557,
0.9160600785258427,
0.916259903562873,
0.9163895996967338,
0.9165306484460564,
0.9166327635495883,
0.9166593784941225,
0.916777615476768,
0.9168628613012402,
0.9169494431997269,
0.9170348934782417,
0.9170892952879469,
0.9171572346029327,
0.9172531037117998,
0.917352013418896,
0.9174837807521837,
0.9175422483696561,
0.9175870960986945,
0.9176091974758679,
0.9176964910740231,
0.9177969010016759,
0.9179032081627195,
0.9179496272553944,
0.9180286761582734,
0.9181671114936788,
0.9182216244167395,
0.9183404240023155,
0.9183943054009488,
0.9184725263367303,
0.918539310047756,
0.9185886113522853,
0.9186280596945512,
0.9187075312838902,
```

```
0.9187488908442251,
0.9188212178846871,
0.9188717595984256,
0.918936512401575,
0.918971123021238,
0.9190453456215925,
0.9191413540636223,
0.9191677829830041,
0.9192737825629581,
0.9193407857596023,
0.9194139648810272,
0.9194857824884579,
0.9195378755693024,
0.919578381023046,
0.9196209122943914,
0.9196827378570361,
0.919772578217045,
0.9198244224167332,
0.9198674970122213,
0.9199306951944993,
0.9200137133881636,
0.9200741357257289,
0.9201572235768604,
0.92021785529013,
0.920254556789032,
0.9203148966340734,
0.9203461796838769,
0.9204003933083668,
0.9204440747630084,
0.9204776136930483,
0.9205006496733293,
0.9205526195599609,
0.9205924792437999,
0.9206180427846173,
0.9206441547626373,
0.9206933907321035,
0.9207368528036611,
0.9207480821656677,
0.9207914061299212,
0.9208285449868486,
0.9208686112004867,
0.9209262589510968,
0.9209673546830058,
0.9209799718648343,
0.9210214517279057,
0.9210351161072529,
0.9210735459343523,
0.9211092280704684,
0.9211384222529899,
0.9211798471483879,
0.9212148687063211,
0.9212209709018705,
0.9213055149110124,
0.9213460091793678,
```

```
 0.9213531294890593,
 0.9213622562236372,
 0.9213848664323053,
 0.921537795522298,
 0.9215718679080923,
 0.9215919904376365,
 0.9215999610738096,
 0.921632921129813,
 0.9216424463298938,
 0.9216539098194634,
 0.921735663314605,
 0.9217681285658502,
 0.9218308479615573,
 0.9218358786839459,
 0.9218587778602517,
 0.9219185418298248,
 0.9219818773264729,
 0.9220109344133648,
 0.9220426313749428,
 0.9220546445321306,
 0.9221095430560163,
 0.9221337896557803,
 0.9222066060689439,
 0.9222254354226821,
 0.9222380530237633,
 0.9222557285941008,
 0.9222604032752573,
 0.9223143364743832,
 0.9223397628056439,
 0.9223588539325585,
 0.9223896450987014,
 0.9224302862467678,
 0.9224511789799589,
 0.9224487333514237,
 0.9224740530375599,
 0.9224807746379442,
 0.9225187652887374,
 0.9225295406570452,
 0.9225437929061333,
 0.9225567400398282,
 0.9225934651854933],
'test-AUC-std': [0.004880780851739435,
 0.0056362882651357065,
 0.009159916111939023,
 0.01164946990928873,
 0.008169726732700153,
 0.0081546699205 1814,
 0.007271260566978086,
 0.007634073326450518,
 0.0083063202502 8167,
 0.007886431254433425,
 0.00842386484720114,
 0.007322031793307574,
 0.00748322169400162,
```

```
0.007422051666004917,
0.0075405821973596856,
0.0070910360184317095,
0.007158940310314591,
0.006801493343537337,
0.006437335942020964,
0.006169936620065218,
0.006231605310757027,
0.006098380699125295,
0.005927512733357988,
0.005740842803596849,
0.005637890645421116,
0.005836180455892275,
0.005642956446152196,
0.005491805288045297,
0.00555799041536 6201,
0.0055038501078874145,
0.005312043397035655,
0.00523779226326297,
0.005074323525831793,
0.005042956557938244,
0.004896480304673372,
0.004669008816188662,
0.004748388306720271,
0.004790477020819169,
0.004789295637994479,
0.004773919817881617,
0.004612638843925359,
0.004697772067317494,
0.004680307821424074,
0.004629025075554672,
0.004525592539307005,
0.004495901735286139,
0.004448791529170965,
0.0043364878535024735,
0.004255887760504171,
0.004110989681105989,
0.004102195584714647,
0.004126916404974036,
0.003960791599363305,
0.003881700876101997,
0.003828428049897119,
0.0038253105226544954,
0.0038154806667070567,
0.0038059750583637644,
0.0038468804802832255,
0.0038692655754765906,
0.003838521794073914,
0.0038189657262287405,
0.003816986262726768,
0.0037084458663314878,
0.003774284072026499,
0.003793300575174272,
0.00383474095190201,
```

```
0.0038261141873062224,
0.0038268954512114747,
0.0037957336814232554,
0.0037055817762661532,
0.0036913056689831963,
0.003735326681949562,
0.003727817379280715,
0.0037532575874870016,
0.0036934220329149043,
0.003615357562235138,
0.0036405339843442915,
0.0036515678289029234,
0.0036846926675650525,
0.003680225639492647,
0.0036758992419380033,
0.0036223205083618024,
0.0036228675156646656,
0.0036063516734278628,
0.00356611526512288,
0.0036176005151583755,
0.0035275371757770126,
0.003447477158893205,
0.0034212020086015803,
0.003444800128003268,
0.0033763506182982257,
0.00338619307253428,
0.00339711621320031,
0.0034125559159829378,
0.0034134648341927715,
0.0033171808270878625,
0.003310700721631879,
0.003297517102273653,
0.0033196896432506995,
0.003251283492803488,
0.0032537301601987135,
0.0032321996337385493,
0.0032066600619691976,
0.0031772044366742583,
0.0031757793734067245,
0.0032135557387260078,
0.003208374259877907,
0.0031879968380202767,
0.003181572921616875,
0.0032026294753245564,
0.0032470007852392273,
0.003204925426413119,
0.003181304877275291,
0.0031829056171323884,
0.0031392502769375307,
0.003091023567341977,
0.0031047475050759625,
0.00306480579216436,
0.0031024442822129234,
0.0030998341802646336,
```

```
0.003151614859449453,
0.003136356863820224,
0.003091751591431771,
0.003056474266855468,
0.003050863558513385,
0.003060855517839588,
0.0030750205603240614,
0.003094251044380634,
0.0031132829298532493,
0.003108422710995172,
0.003079539958411018,
0.0031263325913154097,
0.0031012024284128034,
0.0031147203824085164,
0.0031106480157278705,
0.003107737708662268,
0.0030840737234574464,
0.0031019510379103107,
0.0030339308100376847,
0.003056718803633318,
0.00305736464081176,
0.0030516721756561143,
0.0030286289930400805,
0.00299193416217894,
0.0029814894950542094,
0.002989981543779519,
0.0029655574270147944,
0.002975259382318175,
0.00295445945297395 3,
0.00296534673879089 5,
0.00294332419068927 7,
0.0029324570502118332,
0.00293394063906035,
0.0029309328649609967,
0.0029109942112048883,
0.0029136186392807613,
0.0029214291229736214,
0.0029360502790891025,
0.0029415162888239395,
0.0029375086382955632,
0.0029269252803664565,
0.003002002376620462,
0.002974978389838822,
0.002967668065520858,
0.0029640544759513545,
0.0029350162900919218,
0.0029403644308840927,
0.0029365910846295765,
0.0029393355029424654,
0.0029258721838577883,
0.00295289155264637 2,
0.0029570175896139303,
0.0029563643101150616,
0.002937396245134736,
```

```
 0.0028942707630018436,
 0.00291049780353616 9,
 0.00293118125678430 8,
 0.00292641029492128 1,
 0.00293745955028162 27,
 0.00292446982008299 57,
 0.00282551646573066 08,
 0.00282534256866192 77,
 0.00281592950091671 44,
 0.00281315925655190 3,
 0.00281877551837932 94,
 0.00275623746639886 4,
 0.00274591060643475 95,
 0.00274374738636642 66,
 0.00277036365096488 1,
 0.00272544165665747 13,
 0.00273273394350570 6,
 0.00272580312312344 5,
 0.00274645951400628 47,
 0.00275628095143658 6,
 0.00274281532667159 86,
 0.00272874925883528 95,
 0.00271355687519889 84,
 0.00269315531112766 66,
 0.00265437643900993 73],
'test-Logloss-mean': [0.6619277972515668,
 0.633352470821178,
 0.6079751150976301,
 0.585783149755902,
 0.5645159400724818,
 0.5451865868418122,
 0.5263870019527084,
 0.5112323553239159,
 0.49689347296155256,
 0.48329253645498627,
 0.4707796794694431 6,
 0.459100146324969 54,
 0.449477422381613 2,
 0.440682473725767 6,
 0.432312160910776 34,
 0.424099434321786 26,
 0.41694911215493 43,
 0.409888482432704 54,
 0.403389785044773 6,
 0.398163811113054 3,
 0.392528797831061 26,
 0.38764595044068 37,
 0.384284212937429 26,
 0.38051403475504 16,
 0.376452788063013 8,
 0.37267095374353 78,
 0.36919315039697 94,
 0.366181744156391 76,
 0.363281547407897 03,
```

```
0.36061399630913044,
0.35810718967023014,
0.35548896629588245,
0.35292934128325953,
0.35051877936585196,
0.3486015741394683,
0.3464198926056823,
0.344869043019572,
0.3435767288428279,
0.342181131840142,
0.3408360702294441,
0.33936982823968087,
0.3381200505145883,
0.33657770727708275,
0.33514777629147335,
0.3340991654090802,
0.3328700137125811,
0.33168469005659823,
0.33068247272425544,
0.3295427182296766,
0.3286569780069974,
0.3276960800320088,
0.3265853805774759,
0.3255934187324358,
0.3246614369225553,
0.32387873542062734,
0.32313769119869806,
0.3223949414630127,
0.3218281733538528,
0.3212891412424417,
0.3207351474688526,
0.3200065881658538,
0.31929281145220023,
0.318751742017088,
0.31811862548773534,
0.31737941270731723,
0.3166795490688929,
0.3161086640672534,
0.31565523991821176,
0.3151323127201448,
0.31469158548721604,
0.3142661049337037,
0.31394667879235844,
0.31361709983404257,
0.31333077648121216,
0.3127632110929896,
0.3124018774251524,
0.3119705688817666,
0.3115977820479044,
0.3112993130872163,
0.3109469700269787,
0.3105722026248079,
0.31023426118198516,
0.30993562165662997,
```

```
0.30969518237411225,
0.3094797256112403,
0.3091404403090981,
0.30884076114054676,
0.30851908148177926,
0.3082527614161252,
0.3080934393139183,
0.307919672107048,
0.3075990782920511,
0.30734862781668437,
0.30706765220607185,
0.3068442815490537,
0.3065608722025799,
0.3062980393515567,
0.3061200880091168,
0.30582816181963235,
0.30563107865987177,
0.30543991581609276,
0.30521964237861615,
0.3050410382269367,
0.30484577283976266,
0.3046417456187866,
0.3044949968555896,
0.304272661784182,
0.3041080230915965,
0.30392206047745435,
0.3037874957807883,
0.3035865361906452,
0.3033339017497069,
0.30319207362489237,
0.30289918721552084,
0.30272035824683985,
0.30252359610966467,
0.30227769989137093,
0.3021625783941575,
0.30203491087377815,
0.30186639646407104,
0.3017374430015788,
0.3015246106529008,
0.3014068500038225,
0.30129802250522514,
0.30112548890311197,
0.3009727672518845,
0.30085317423530616,
0.30070862987918884,
0.3005590780085024,
0.30043603981089445,
0.3002849398786475,
0.30019532405777183,
0.3000444975916283,
0.29994730858320723,
0.2998416433811103,
0.299732377133195,
0.29962072232813347,
```

```
0.29950711306502836,
0.2994000589917757,
0.2993257925308697,
0.2992099932750117,
0.2991126628647505,
0.299068691734299,
0.29894679581236927,
0.2988339779592864,
0.2987499021995937,
0.2985953844901841,
0.2985103957121001,
0.2984092154380766,
0.298321221997696,
0.29826668929355477,
0.2981720412106252,
0.29809571522999384,
0.29803211369337973,
0.2979159877198952,
0.2978315686701132,
0.2977787957148074,
0.2975993269995674,
0.29748958774513984,
0.297449637258187,
0.29737476372309884,
0.29730783135266375,
0.297010406412951,
0.29694051815575306,
0.29688444480880993,
0.2968487448766323,
0.29674809051299716,
0.2967052316939425,
0.29666760613751175,
0.29651514978187776,
0.2964313803564495,
0.2962948230005018,
0.29625774194685933,
0.29619117866270955,
0.29608627681513733,
0.2959609679380688,
0.29588296549479337,
0.2957918627323032,
0.29574356164895116,
0.2956397671575131,
0.29558560452790644,
0.29544065742886344,
0.29537576491306444,
0.29532311363028274,
0.2952640041382089,
0.2952118569683413,
0.2950867789820047,
0.2950221741541756,
0.2949705718363666,
0.29487759882963,
0.29479801025637425,
```

```
 0.2947595679859905,
 0.29472847709438077,
 0.2946640602140803,
 0.29463098640304564,
 0.29454179008279563,
 0.2944999815487177,
 0.2944440779030479,
 0.29441675940361034,
 0.2943605216677096],
'test-Logloss-std': [0.0004815231957404724,
 0.0012123121216483933,
 0.0026942576513129127,
 0.004266002630438518,
 0.005633579820544494,
 0.00574781044616808 7,
 0.0036278358162015805,
 0.003809853956894585,
 0.00503469515185944 9,
 0.004944445116065673,
 0.005474533698047862,
 0.005544864686672889,
 0.006119653494357698,
 0.0051291165903892735,
 0.004957959965907101,
 0.0053500571556709085,
 0.005561950725415701,
 0.005055793321713159,
 0.004886747380776494,
 0.004570413001356731,
 0.005044934445806659,
 0.004360508629735958,
 0.00496844387530814,
 0.005066130764874758,
 0.004899609144763377,
 0.005316036290914191,
 0.00488630858633007,
 0.004792198109779253,
 0.004913299125274926,
 0.005092605334472941,
 0.005067495497998179,
 0.005156815448243944,
 0.005070471725534231,
 0.00530751435696589 4,
 0.0053082578617819585,
 0.005230946329309367,
 0.004851359143714329,
 0.005261136155790326,
 0.0050773242668967385,
 0.005537243198720632,
 0.005064721759868439,
 0.005292197730159766,
 0.005020096498972035,
 0.0047887929788707735,
 0.004698290765858559,
```

```
0.004783100908900113,
0.004711221803809502,
0.004832014369406063,
0.004911413846994724,
0.004796981298309675,
0.004901976277552676,
0.004828714100142662,
0.004738498591635497,
0.004613805038981319,
0.004476574805436958,
0.004418833446634928,
0.0043980321010898375,
0.004134795584391907,
0.00401939480726614,
0.004123947426708803,
0.004072842348463235,
0.003889946985084709,
0.0040035140031791875,
0.0038849442953518853,
0.003956128853424487,
0.00403842040333615,
0.00403974393363879,
0.00400873882085097,
0.004056444117766849,
0.003954927031799195,
0.003890537652312528,
0.0038844116373812298,
0.004009862757254476,
0.004031943124459497,
0.004015833241818873,
0.003954313440681121,
0.0039042544486926982,
0.003990563707941665,
0.004051982986753137,
0.004100340138486231,
0.004045350694399622,
0.004035027600573812,
0.0040144143621252725,
0.004033136866784444,
0.00399413995552677,
0.003927389083184023,
0.004016046913840642,
0.0038742247963408924,
0.0036843476361594603,
0.0036132160917571893,
0.003568358866668753,
0.00354475437620983 8,
0.00354726986586410 9,
0.0035653327576486604,
0.003569225161759438,
0.0035652465777352297,
0.003516589542211796,
0.00344067026486408 5,
0.0034495052112279412,
```

```
0.0034271905239688584,
0.0034063606309609313,
0.0033696147843546697,
0.0033447312241794134,
0.0033301113183554335,
0.003317373170070594,
0.0033732299793911833,
0.00337069539283441,
0.0033715579915439674,
0.0033028855282904,
0.0033213193567251205,
0.003391950038837126,
0.00334673614024365,
0.00342727121293418 9,
0.0034182597525603017,
0.0034457037305072684,
0.00340906960182116 6,
0.0034389553027461325,
0.0034456370944576788,
0.003363559516366124,
0.00346179663324608 1,
0.0034676691329014447,
0.003600169724672979,
0.00358174394509959 9,
0.0035315274844188354,
0.0035360088303347716,
0.0035164031500563475,
0.0035534302189530013,
0.00356109325872028 8,
0.00367589899182513,
0.00376308362362941 3,
0.0038228323136451346,
0.0037870840089639394,
0.00385899306602252 1,
0.0038362867421646734,
0.0038371143503046425,
0.00380897790171234 2,
0.0037829955618292393,
0.0037932636603963056,
0.00375525574179280 5,
0.0036673339012129403,
0.00371555160364303 6,
0.00374448154671 12,
0.00374251164852170 4,
0.0037192900411646924,
0.0036849340959144395,
0.0036652170929835992,
0.00365101886383926 2,
0.00361477371812904 9,
0.0036166079728509153,
0.00358001750471014 1,
0.00361336222579952 7,
0.0035791906246582573,
0.00358794025044524 1,
```

```
0.0035935249249130032,
0.0035952289355390553,
0.003607802933282495,
0.003595810048241921,
0.00362509801501853 8,
0.003678890695521068,
0.003711973795972078,
0.003716558968975707,
0.003675973577456051,
0.003780979571447422,
0.0037411272873059376,
0.0037318986894037,
0.0037400290933098856,
0.0037389606704809814,
0.0037479459557363437,
0.003759312011314574,
0.0037637382021900342,
0.003747506006257857,
0.003822866292252748,
0.003814363582821846,
0.0038432422798768888,
0.0038097823804814123,
0.003757689916212474,
0.0037745170249801284,
0.0038094384078392485,
0.003799127469365114,
0.0038326846886494984,
0.003814626256739448,
0.003677027006720832,
0.0036755372067804205,
0.0036957791300402643,
0.0036817504104487333,
0.0037347559978107487,
0.0036553705558281507,
0.0036375652154286772,
0.003641863127173109,
0.0036929943084354207,
0.003634574652584597,
0.0036266479428636563,
0.0036278998493133843,
0.003673460954827481,
0.003692714407585181,
0.00368047452418684,
0.0036973330762986876,
0.003678306129694728,
0.0036568926237728998,
0.0035989182703416362],
'train-Logloss-mean': [0.6622899104779973,
0.6338334455814484,
0.6086729096298628,
0.5867224851087957,
0.5655935123744574,
0.5463752207845051,
0.527408464522565 8,
```

```
0.5274064643225058,
0.5123194028900616,
0.4980829397221373,
0.48448083905000544,
0.4720776538851938,
0.46030973260477404,
0.45069215971593124,
0.4418109570753259,
0.4333404769790417,
0.4250844253379668,
0.41795832249423787,
0.4109267673920911,
0.4044852851615581,
0.39917549796517715,
0.39359313202692414,
0.388648437121416,
0.3851830042562842,
0.38138435132561993,
0.3772952477205966,
0.37355629353166303,
0.3700264063590291,
0.3669914052107301,
0.36398471861596443,
0.3612369487611884,
0.3586552737701337,
0.35598040401037384,
0.35335029218785835,
0.3509434704120585,
0.3490360587890852,
0.3469222638187015,
0.3452481411408947,
0.3438556386371256,
0.3424051447665062,
0.34104807074828497,
0.3394972034092419,
0.33819522303088423,
0.33648724530539925,
0.335019022626871,
0.3339366116598912,
0.33262230453714126,
0.33132543366732575,
0.33027168527384737,
0.3291081853101259,
0.32811586487862016,
0.32706888166388365,
0.3259478863933221,
0.3249937569404666,
0.323948179064032,
0.3230958143736691,
0.3222753558627647,
0.32144811983465743,
0.32085351265960066,
0.3201988167264315,
0.3196331600019773,
0.3189133021779218
```

```
0.3189133021779218,
0.31808000400419983,
0.3174783083113335,
0.31681589760271245,
0.3160480719931017,
0.31528314384699746,
0.31463350018991454,
0.3141452982543225,
0.31350744916536066,
0.31295592795293037,
0.3124532694540784,
0.3120918234872936,
0.3116653417823289,
0.31130947120165475,
0.31067542080073013,
0.3102570625053096,
0.30980254147511627,
0.30937405615475083,
0.3090125976000988,
0.3086094278708638,
0.30818483673767677,
0.3078176585765449,
0.3075023722119212,
0.307159289374755,
0.30689499180482516,
0.30653061055907127,
0.3062006146861149,
0.3058404601325852,
0.3055355581773831,
0.30532206337340106,
0.30506370697840646,
0.3046779411761327,
0.30433765809458796,
0.3040305575517959,
0.3037249193301998,
0.30338676706535356,
0.3030799285395936,
0.3028163751883109,
0.30250800029897035,
0.30226956774608255,
0.30201422253720334,
0.30172632001504845,
0.30145723063225355,
0.30114046023321045,
0.30090440033780297,
0.3006975696635179,
0.30046526321105543,
0.3002221308574404,
0.2999664083397208,
0.2997632945907226,
0.2995206979943699,
0.2992798560622513,
0.2990276891731447,
0.29871732512456245,
0.29851003287085847,
```

```
0.29851003870858 47,
0.2982721933778992,
0.29798001085680953,
0.2978127345652068,
0.2975768000433141,
0.2973051836632093,
0.297149900699107,
0.2968619502669649,
0.29663699484853595,
0.29647321207030847,
0.29619979620071296,
0.29599383678024166,
0.29582802950314396,
0.29563630242980204,
0.295445560536711,
0.2952497386374102,
0.2950440324428821,
0.2948963438560385,
0.29466149105645534,
0.2944773826240648,
0.2943044299228763,
0.2941346564836151,
0.29395839279601305,
0.29378937186264853,
0.2935896045080402,
0.29344493107727954,
0.29330518164796077,
0.2931710820525187,
0.29306395050745676,
0.2928766434343953,
0.2927086823838447,
0.29255891252847627,
0.29235672840393634,
0.29221710906169873,
0.2920316436584347,
0.2919082757446007,
0.2917832054779016,
0.29165244581331234,
0.2915356852302658,
0.2914313771962966,
0.2912699138634695,
0.2911631186051024,
0.2910355698766012,
0.2908323198932905,
0.2906935908620521,
0.2905854584019642,
0.2904404118401278,
0.29031867188979427,
0.29003981138865415,
0.28990477547526866,
0.28978920038633404,
0.2896988864898875,
0.28949580841231565,
0.28933789648627123,
0.2892424404111358
```

```
 0.2892424404111358,
 0.28907294430645697,
 0.2889257318234057,
 0.2887745132608906,
 0.28866618898095237,
 0.28854521336713107,
 0.2884261794600617,
 0.2882552161706773,
 0.2881568465915048,
 0.28803072363154514,
 0.28789714233318126,
 0.2877556858148373,
 0.28763693135147356,
 0.28744428057109256,
 0.28731631225856197,
 0.2871998999563152,
 0.2870940061708143,
 0.2869728811628364,
 0.2868022270294874,
 0.2866847015744303,
 0.2865917930732887,
 0.286431516472253,
 0.286315382124988,
 0.28622415113482086,
 0.28610447269023137,
 0.2859639360227212,
 0.2858628374267684,
 0.28571732478350037,
 0.285599989378875,
 0.28551438056541895,
 0.2854363578794396,
 0.2852994647004066],
'train-Logloss-std': [0.0006758371198890583,
 0.0010084276532632166,
 0.000979322253372526,
 0.0020738904098911667,
 0.0028145304088380094,
 0.0025868469552483868,
 0.001028096728620874,
 0.0017435098902552444,
 0.0011944578545244867,
 0.0007603930218898285,
 0.0007396085344304075,
 0.0009711600728040632,
 0.000982086467352035,
 0.00084747759056518,
 0.0016909281491874575,
 0.0014270568186882046,
 0.0005896290217929836,
 0.0011433730779760272,
 0.0013970354963524717,
 0.0017648247332378627,
 0.001593468489883777,
 0.002370674556071361,
```

```
0.0015431627456651002,
0.0013661561988886759,
0.0015699428713854898,
0.0013399124380108 36,
0.0017233444746046246,
0.0019088575241370813,
0.002028953868729136,
0.002040161526335074,
0.002013242281970153,
0.0018368969866696344,
0.0017685452478851404,
0.0015098276106993462,
0.0014025249019826442,
0.0014952529268929587,
0.00183035464274331,
0.0013752322848943062,
0.0016678216197731196,
0.0011389834641337148,
0.0015764208676432242,
0.0013339453391803952,
0.0015555441356162529,
0.001804549256174666,
0.0018350264909501402,
0.001785606577540949,
0.0018041139842598693,
0.0016740875583562218,
0.0015922924564457965,
0.001847283032921499,
0.0017654986857961515,
0.0018390964304954213,
0.0018595721139330668,
0.0018906097962974858,
0.001994225786917644,
0.0020556434289902632,
0.002140749420474707,
0.002325484562031229,
0.0023839268777847197,
0.0022727880644945755,
0.0022675774330164333,
0.0025088917581238563,
0.0023414250273255696,
0.0024435911954328936,
0.002360912499948214,
0.0022704696532884648,
0.0022162852191939773,
0.002207474954872996,
0.0021518476379164733,
0.0022924777047263695,
0.002286729223153918,
0.002264853142290109,
0.002139947600113425,
0.0021158011962037738,
0.0020361142855305474,
0.0020504657213256585,
```

```
0.002020860390306047,
0.0019441963153415493,
0.0019137034806048139,
0.0019096362753070006,
0.0019328208533323226,
0.0019557288058552735,
0.0019379112422759034,
0.0019749822358847953,
0.001984339865433604,
0.0020132775958266918,
0.0019398507710718864,
0.001983085007691663,
0.0020449523349331546,
0.0020860154977313237,
0.0021404163635242775,
0.002169010051420115,
0.0021933721428182918,
0.0022246693810444394,
0.0022852189235151176,
0.002318632056041959,
0.0023741998932181917,
0.002423325737883078,
0.00241171211608014,
0.0024428779380854496,
0.0024590019956121033,
0.0024371575573523766,
0.002443799040517186,
0.0025253372642473175,
0.002464024793540458,
0.00239072977214720 4,
0.00247564466523829 8,
0.0024571966290403056,
0.0025360661084792534,
0.002541802461984961,
0.00247775122883669,
0.0024336686649056413,
0.002441988446830068,
0.002389957520905505,
0.0023501713457868534,
0.0023973937898821704,
0.002347555843568957,
0.0023531532367201463,
0.002459001305506478,
0.0024462234527226324,
0.0024001481482258283,
0.0022828166374761444,
0.0023282578345947373,
0.0023526072704620916,
0.002369629783851458,
0.002408940679406986,
0.0023585226715843806,
0.0024071211632547967,
0.0023203053951421438,
0.002292082635072194,
```

```
0.0022481717370447384,
0.002268070559543982,
0.0022225885365608207,
0.002262192000001821,
0.0023043937871093556,
0.00230898217259606,
0.002334368281267858,
0.002333303058535095,
0.00239294533048733,
0.0024264630879972755,
0.002431637363071556,
0.00244646445006348277,
0.00245161658593948814,
0.0025070132426479115,
0.00253818980269391 6,
0.0025188022842722976,
0.00254384160229434 9,
0.0025693260967016717,
0.0025312332200591397,
0.0025254105080362963,
0.0025178499902540185,
0.0025174392901922194,
0.0025150188103724466,
0.0025190410512316483,
0.00250016509618 6849,
0.0024606374699326188,
0.0024558238299950356,
0.0024463052909310954,
0.0024190631884166637,
0.0023550058081419512,
0.0023233884489108265,
0.0023267015187158624,
0.00230420015772 4587,
0.002310737671117243,
0.0023183795700388605,
0.0022806555051340304,
0.0022846776937686737,
0.002276991353788541,
0.0022674603437705964,
0.00229300653226 7939,
0.002276049944283871,
0.00223514845166 8369,
0.00225149659124 5988,
0.0022217465342373767,
0.00221351288584 0858,
0.00224593601439 5611,
0.0022349048115319656,
0.00223016815601 1983,
0.00228871405336 4743,
0.00224940618380 3928,
0.0022950173876788676,
0.00238555243913 5945,
0.0023666294821086393,
0.00232329480695 0134,
```

```
0.0023192712973228737,
0.0022787071585641896,
0.002337705886921147,
0.0023455767481175027,
0.0023134930081286473,
0.002310415113281136,
0.002327420022366643,
0.00234344479490085,
0.002361569820954491,
0.0023423479076410594,
0.002323810925067819,
0.0023214187597617 92,
0.002271743039431852,
0.002285127066108061,
0.00229264662263457,
0.0023340007831545917]})}
```