Аналитический отчёт

Романенков М.Г.

Содержание

1. Анализ представленных данных (EDA)

Выведем несколько верхних строк датасета, для ознакомления.

```
dataset.head(10)
```

✓ 0.0s — Python

	Unnamed: 0	IC50, mM	CC50, mM	SI	MaxAbsEStateIndex	MaxEStateIndex	MinAbsEStateIndex	MinEStateIndex	qed	SPS	...	fr_sulfide	fr_sulfonamd	fr_sulfone	fr_term_acetylene	fr_tetrazole	fr_thiazole	fr_thiocyan	fr_thiophene	fr_unbrch_alkane	fr_ure
0	0	6.239374	175.482382	28.125000	5.094096	5.094096	0.387225	0.387225	0.417362	42.928571	...	0	0	0	0	0	0	0	0	3	
1	1	0.771831	5.402819	7.000000	3.961417	3.961417	0.533868	0.533868	0.462473	45.214286	...	0	0	0	0	0	0	0	0	3	
2	2	223.808778	161.142320	0.720000	2.627117	2.627117	0.543231	0.543231	0.260923	42.187500	...	0	0	0	0	0	0	0	0	3	
3	3	1.705624	107.855654	63.235294	5.097360	5.097360	0.390603	0.390603	0.377846	41.862069	...	0	0	0	0	0	0	0	0	4	
4	4	107.131532	139.270991	1.300000	5.150510	5.150510	0.270476	0.270476	0.429038	36.514286	...	0	0	0	0	0	0	0	0	0	
5	5	15.037911	30.075821	2.000000	5.758408	5.758408	0.278083	0.278083	0.711012	28.600000	...	0	0	0	0	0	0	0	0	0	
6	6	18.908167	14.559288	0.770000	2.584472	2.584472	0.429649	0.429649	0.328539	36.564103	...	0	0	0	0	0	0	0	0	0	
7	7	28.773087	23.593931	0.820000	13.821880	13.821880	0.079845	-0.301260	0.217332	31.333333	...	0	0	0	0	0	0	0	0	4	
8	8	50.057068	153.709268	3.070681	14.171614	14.171614	0.019123	-0.411828	0.187781	29.157895	...	0	0	0	0	0	0	0	0	0	
9	9	6.400847	400.906360	62.633333	6.135893	6.135893	0.239226	0.239226	0.439915	36.400000	...	0	0	0	0	0	0	0	0	0	

10 rows × 214 columns

Обратим внимание на ключевые параметры

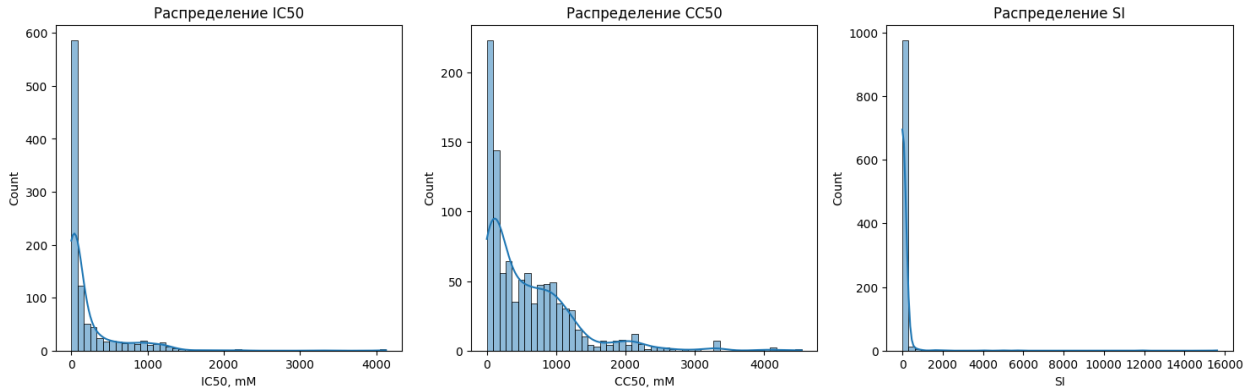
- IC50 – концентрация для 50% ингибирования
- CC50 – концентрация для 50% цитотоксичности
- SI – индекс селективности (CC50 / IC50)

Также заметим, что присутствует не информативный столбец 'Unnamed: 0' и удалим его.

```
# Удалим ненужный столбец
dataset = dataset.drop(['Unnamed: 0'], axis=1)
dataset.head()
```

✓ 0.0s

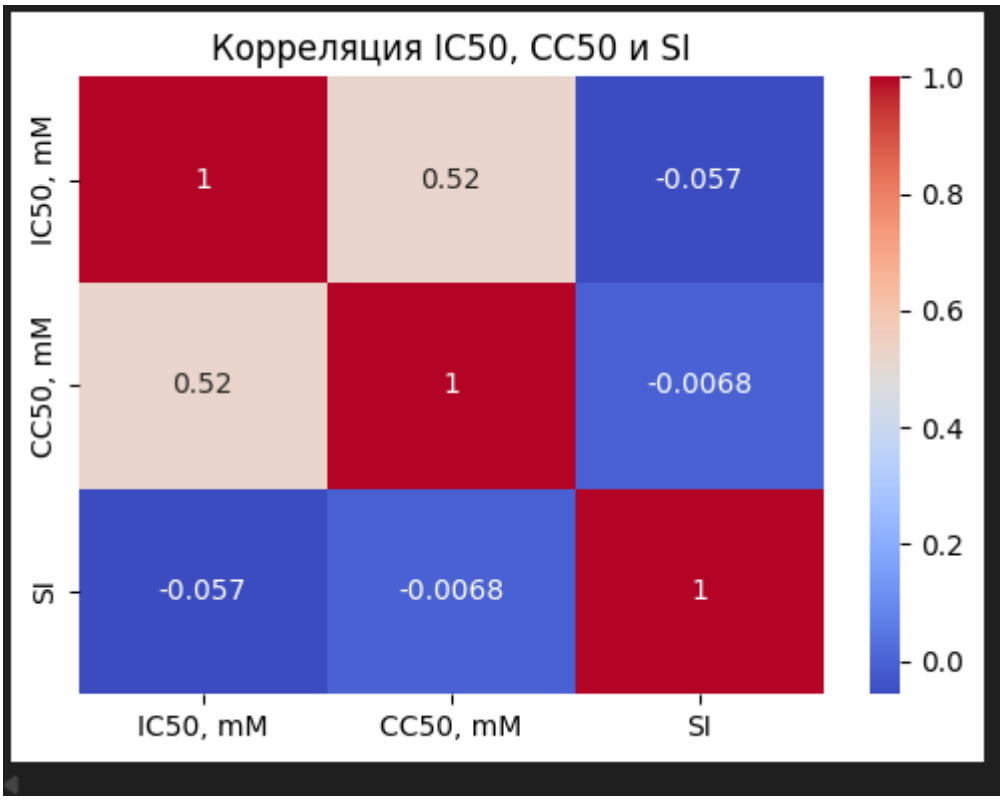
Визуализируем распределения ключевых параметров



Видим, что все распределения имеют сильный сдвиг вправо.

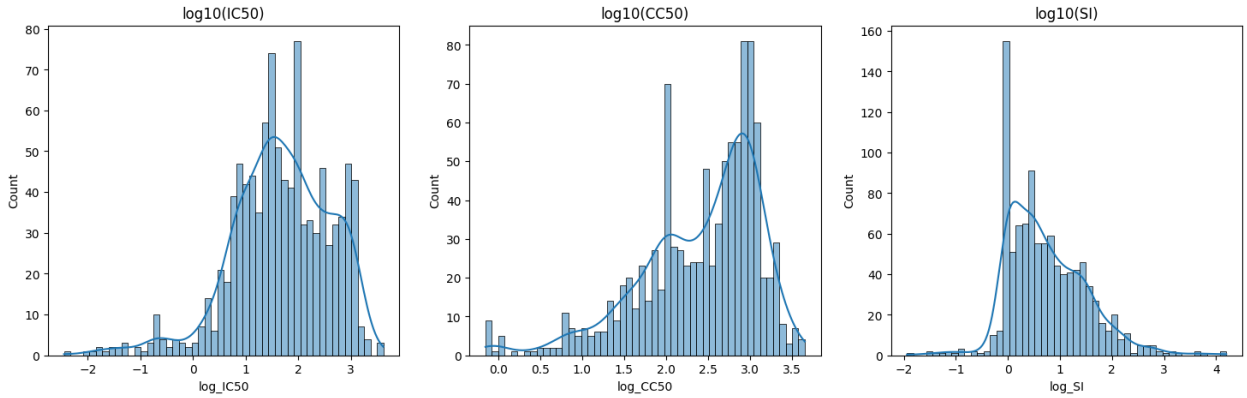
Также, в распределениях IC50 и SI присутствуют аномально большие значения.

Затем обратим внимание на корреляцию между ключевыми признаками



Видно, что между IC50 и CC50 корреляция низкая. Следовательно это независимые характеристики.

Чтобы уменьшить влияние выбросов и лучше увидеть структуру данных, применим логарифмирование.



После логарифмирования распределения стали более похожи на нормальные. Это позволит нам использовать линейные модели и стандартные статистические тесты.

P.S. В дальнейшем логарифмированными столбцами я планирую пользоваться для задач регрессии.

Далее я реализовал метод ‘def prepare_dataset()’ для подготовки отдельных датасетов для задач регрессии и классификации.

Сигнатура функции

```python
def prepare_dataset(
    dataset,
    target_col,
    key_cols,
    log_transform=True,
    corr_threshold=0.85,
    min_corr_target=0.05,
    classification=False,
):
    """
    :param dataset:         датасет
    :target_col:            название целевой переменной
    :key_cols:              список всех ключевых признаков (IC50, CC50, SI и derived)
    :log_transform:         логарифмирование целевой переменной (для регрессии)
    :corr_threshold:        порог корреляции между признаками
    :min_corr_target:       минимальная корреляция признака с target
    :classification:        если True, не фильтруем признаки по корреляции с target

    :return dataset_prepared:    датафрейм (признаки + target)
    """

    data = dataset.copy()
```

Тело функции

```python
    # Удаление константных признаков
    nunique = data.nunique()
    constant_features = nunique[nunique == 1].index
    data = data.drop(columns=constant_features)
    print(f"Удалено константных признаков: {len(constant_features)}")
```

Удаляем все столбцы в которых значения неизменны

```python
# Удаление всех key_cols, кроме целевого
drop_key = [c for c in key_cols if c in data.columns and c != target_col]
data = data.drop(columns=drop_key)
print(f"Удалены другие ключевые признаки: {drop_key}")
```

Оставляем только целевой признак, необходимый для конкретной задачи. Это сделано чтобы не возникало ситуаций как эта – (SI = CC50 / IC50, значит SI напрямую содержит информацию о целевой переменной.)

```python
# Удаление выбросов (только для регрессии)
if not classification:
    Q1 = data[target_col].quantile(0.25)
    Q3 = data[target_col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    before = data.shape[0]
    data = data[
        (data[target_col] >= lower_bound) & (data[target_col] <= upper_bound)
    ]
    print(f"Удалено выбросов: {before - data.shape[0]} строк")
```

На этом этапе выбросы удаляются с помощью IQR(межквартильный размах**).**

Для классификации выбросы не удаляются, т.к. даже если у соединения экстремальное значение ключевого параметра, это всё равно один из классов (0 или 1).

Если мы удалим такие строки, мы искусственно исказим баланс классов.

```python
# Целевая переменная
if log_transform and not classification:
    data[f"log_{target_col}"] = np.log10(data[target_col] + 1e-6)
    target_name = f"log_{target_col}"
else:
    target_name = target_col
```

Логарифмируем целевую переменную только для задач регрессии (причины описаны ранее).

Теперь уберём признаки, которые слабо коррелируют с ключевым признаком (< min_corr_target) и признаки, которые имеют высокую корреляцию между собой (> corr_threshold).

```
# Формирование признаков
drop_cols = [
    c
    for c in [target_col, f"log_{target_col}"]
    if c in data.columns and c != target_name
]
features = data.drop(columns=drop_cols)
print(f"Признаков перед фильтрацией: {features.shape[1]}")

# Очистка признаков до корреляции
stds = features.std()
zero_std = stds[stds == 0].index
features = features.drop(columns=zero_std)
features = features.dropna(axis=1)
print(f"После предварительной очистки (NaN/нулевая дисперсия): {features.shape[1]}")

# Отбор признаков
if not classification:
    corrs = features.corrwith(data[target_name]).abs().dropna()
    selected_features = corrs[corrs > min_corr_target].index
    features = features[selected_features]
    print(f"После отбора по корреляции с target: {features.shape[1]}")

# Корреляция между признаками
corr_matrix = features.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
drop_high_corr = [
    column for column in upper.columns if any(upper[column] > corr_threshold)
]
features = features.drop(columns=drop_high_corr)
print(f"После удаления сильно коррелирующих признаков: {features.shape[1]}")
```

В завершении вернём датасет, готовый для выполнения дальнейших задач.

```
# Итоговый датасет
dataset_prepared = pd.concat([features, data[target_name]], axis=1)

dataset_prepared = dataset_prepared.loc[:, ~dataset_prepared.columns.duplicated()]

return dataset_prepared
```

С помощью ранее описаного метода подготовим датасеты для дальнейшей работы.

Все они будут храниться в папке ‘prepared_datasets’ в формате ‘.csv’.

```python
# список ключевых колонок
key_cols = ["IC50, mM", "CC50, mM", "SI",
            "IC50_binary", "CC50_binary", "SI_binary", "SI_gt8"]

# создаем бинарные целевые переменные
dataset["IC50_binary"] = (dataset["IC50, mM"] > dataset["IC50, mM"].median()).astype(int)
dataset["CC50_binary"] = (dataset["CC50, mM"] > dataset["CC50, mM"].median()).astype(int)
dataset["SI_binary"]   = (dataset["SI"] > dataset["SI"].median()).astype(int)
dataset["SI_gt8"]      = (dataset["SI"] > 8).astype(int)

# список задач
tasks = [
    ("IC50, mM", False, "./prepared_datasets/regression_ic50.csv"),
    ("CC50, mM", False, "./prepared_datasets/regression_cc50.csv"),
    ("SI", False, "./prepared_datasets/regression_si.csv"),
    ("IC50_binary", True, "./prepared_datasets/classification_ic50.csv"),
    ("CC50_binary", True, "./prepared_datasets/classification_cc50.csv"),
    ("SI_binary", True, "./prepared_datasets/classification_si.csv"),
    ("SI_gt8", True, "./prepared_datasets/classification_si_gt8.csv"),
]

# прогоняем все задачи
for target, is_class, filename in tasks:
    prepared = prepare_dataset(dataset, target_col=target, key_cols=key_cols, classification=is_class)
    prepared.to_csv(filename, index=False)
    print(f"Сохранено: {filename}")
```

Пример подготовки датасета для задачи регресии

```
=== Подготовка для IC50, mM (regression) ===
Удалено константных признаков: 18
Удалены другие ключевые признаки: ['CC50, mM', 'SI', 'IC50_binary', 'CC50_binary', 'SI_binary', 'SI_gt8']
Удалено выбросов: 147 строк
Признаков перед фильтрацией: 193
После предварительной очистки (NaN/нулевая дисперсия): 179
После отбора по корреляции с target: 99
После удаления сильно коррелирующих признаков: 72
Сохранено: ./prepared_datasets/regression_ic50.csv
```

Пример подготовки датасета для задачи классификации

```
=== Подготовка для SI_binary (classification) ===
Удалено константных признаков: 18
Удалены другие ключевые признаки: ['IC50, mM', 'CC50, mM', 'SI', 'IC50_binary', 'CC50_binary', 'SI_gt8']
Признаков перед фильтрацией: 193
После предварительной очистки (NaN/нулевая дисперсия): 181
После удаления сильно коррелирующих признаков: 125
Сохранено: ./prepared_datasets/classification_si.csv
```

2. Задачи регрессии

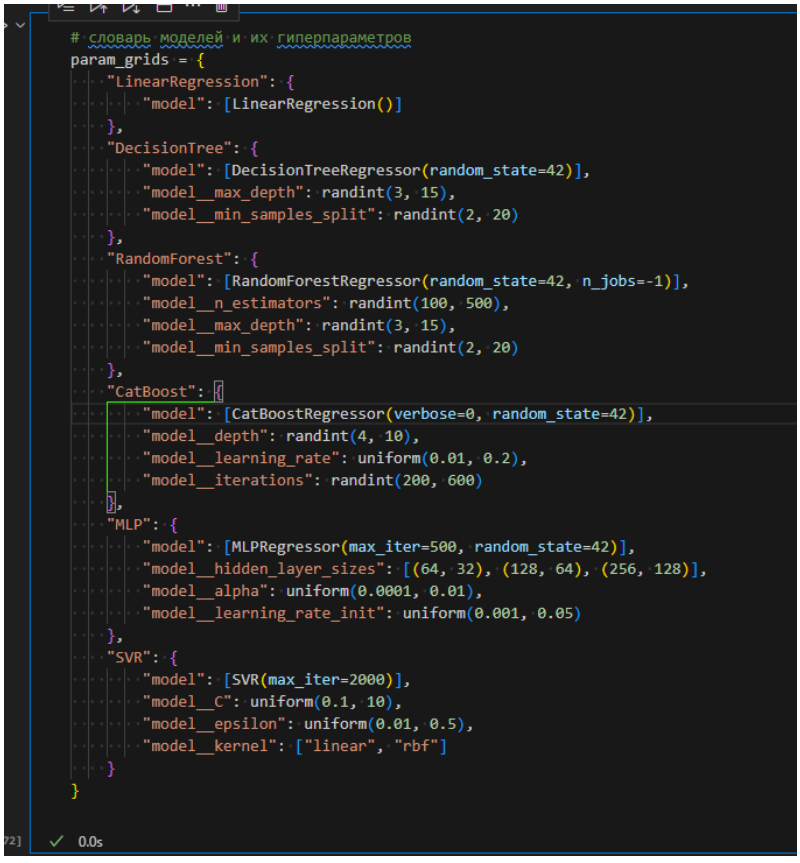
Для решения задач регресии будут использоваться такие модели:

- **Линейная регрессия** (LinearRegressio**n)**
- **Дерево решений** (DecisionTreeRegressor)
- **Случайный лес** (RandomForestRegressor)
- **Градиентный бустинг деревьев решений (**CatBoostRegressor)
- **Многослойный перцептрон (**MLPRegressor**)**
- **Метод опорных векторов (**Support Vector Regressor**)**

Сравнивать данные модели я буду по следующим метрикам:

- **MAE** (Mean Absolute Error) - показывает, насколько в среднем модель ошибается в тех же единицах, что и целевая переменная.
- **RMSE** (Root Mean Squared Error) - более чувствителен к крупным ошибкам, т.к. ошибки возводятся в квадрат.
- **R^2** (коэффициент детерминации) - насколько хорошо модель объясняет вариацию данных.

Для подбора лучших гиперпараметров будет использоваться следующий словарь



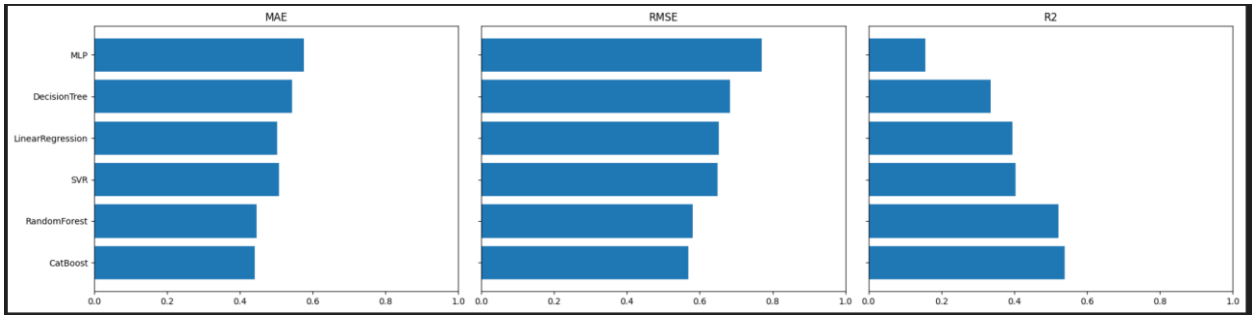
```
# словарь моделей и их гиперпараметров
param_grids = {
    "LinearRegression": {
        "model": [LinearRegression()]
    },
    "DecisionTree": {
        "model": [DecisionTreeRegressor(random_state=42)],
        "model__max_depth": randint(3, 15),
        "model__min_samples_split": randint(2, 20)
    },
    "RandomForest": {
        "model": [RandomForestRegressor(random_state=42, n_jobs=-1)],
        "model__n_estimators": randint(100, 500),
        "model__max_depth": randint(3, 15),
        "model__min_samples_split": randint(2, 20)
    },
    "CatBoost": {
        "model": [CatBoostRegressor(verbose=0, random_state=42)],
        "model__depth": randint(4, 10),
        "model__learning_rate": uniform(0.01, 0.2),
        "model__iterations": randint(200, 600)
    },
    "MLP": {
        "model": [MLPRegressor(max_iter=500, random_state=42)],
        "model__hidden_layer_sizes": [(64, 32), (128, 64), (256, 128)],
        "model__alpha": uniform(0.0001, 0.01),
        "model__learning_rate_init": uniform(0.001, 0.05)
    },
    "SVR": {
        "model": [SVR(max_iter=2000)],
        "model__C": uniform(0.1, 10),
        "model__epsilon": uniform(0.01, 0.5),
        "model__kernel": ["linear", "rbf"]
    }
}

0.0s
```

2.1. Задача регрессии IC50

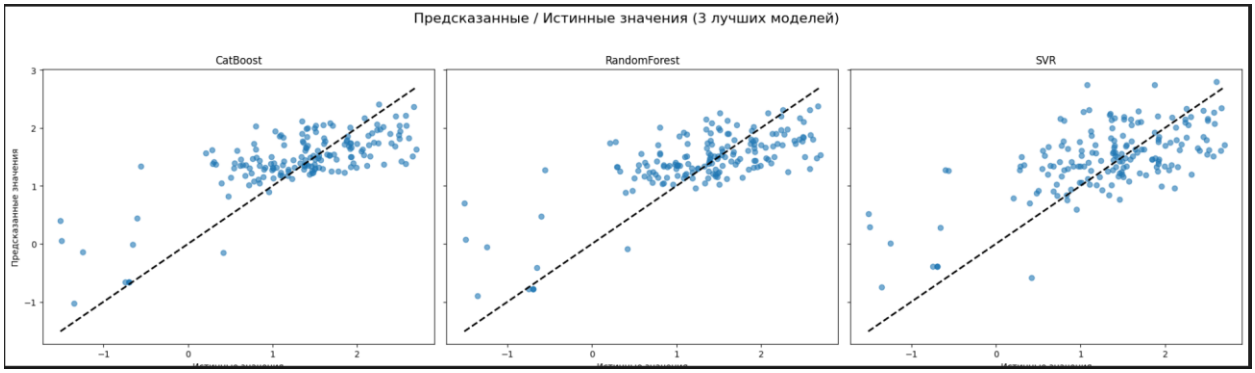
Метрики для каждой модели с наиболее удачными наборами гиперпараметров.

	Model	Best Params	MAE	RMSE	R2
3	CatBoost	{'model': <catboost.core.CatBoostRegressor obj...	0.440335	0.569124	0.539629
2	RandomForest	{'model': RandomForestRegressor(n_jobs=-1, ran...	0.446543	0.580440	0.521141
5	SVR	{'model': SVR(max_iter=2000), 'model__C': 7.89...	0.507152	0.648014	0.403155
0	LinearRegression	{'model': LinearRegression()}	0.503099	0.652469	0.394919
1	DecisionTree	{'model': DecisionTreeRegressor(random_state=4...	0.543274	0.683518	0.335961
4	MLP	{'model': MLPRegressor(max_iter=500, random_st...	0.575479	0.770097	0.157083

Графики сравнения моделей по метрикам



Графики предсказанных против истинных значений 3х лучших моделей

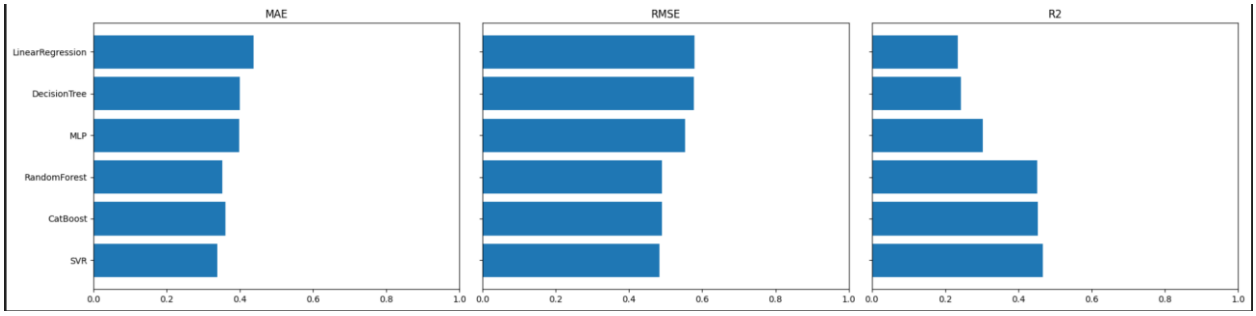


2.2. Задача регрессии CC50

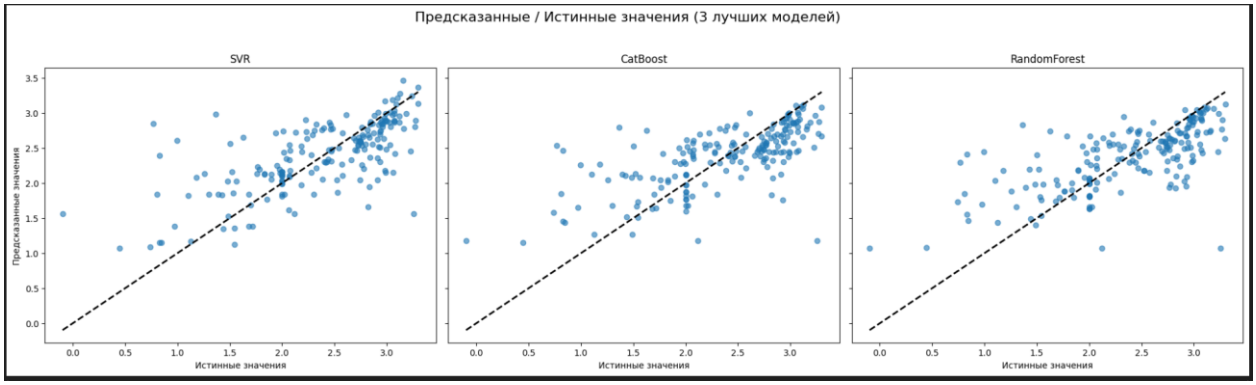
Метрики для каждой модели с наиболее удачными наборами гиперпараметров.

	Model	Best Params	MAE	RMSE	R2
5	SVR	{'model': SVR(max_iter=2000), 'model__C': 1.93...	0.338550	0.483626	0.467260
3	CatBoost	{'model': <catboost.core.CatBoostRegressor obj...	0.360343	0.489647	0.453913
2	RandomForest	{'model': RandomForestRegressor(n_jobs=-1, ran...	0.352388	0.490452	0.452115
4	MLP	{'model': MLPRegressor(max_iter=500, random_st...	0.398769	0.553385	0.302489
1	DecisionTree	{'model': DecisionTreeRegressor(random_state=4...	0.400765	0.576521	0.242947
0	LinearRegression	{'model': LinearRegression()}	0.437637	0.579489	0.235133

Графики сравнения моделей по метрикам



Графики предсказанных против истинных значений 3х лучших моделей

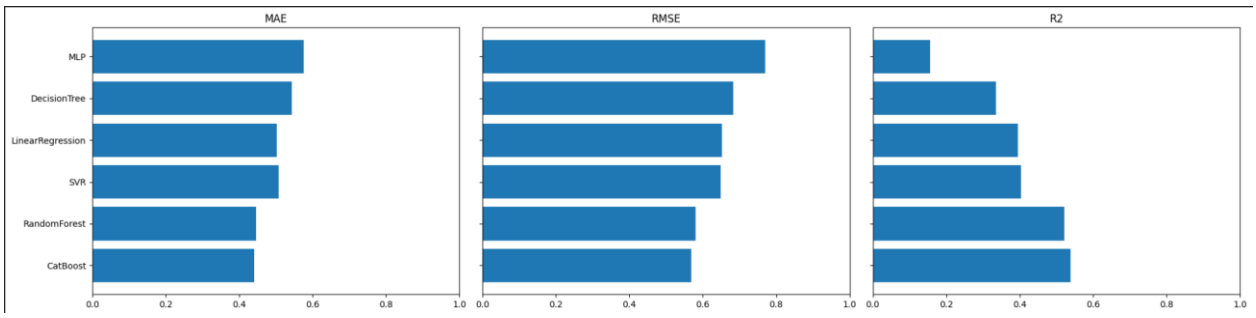


2.3. Задачи регрессии SI

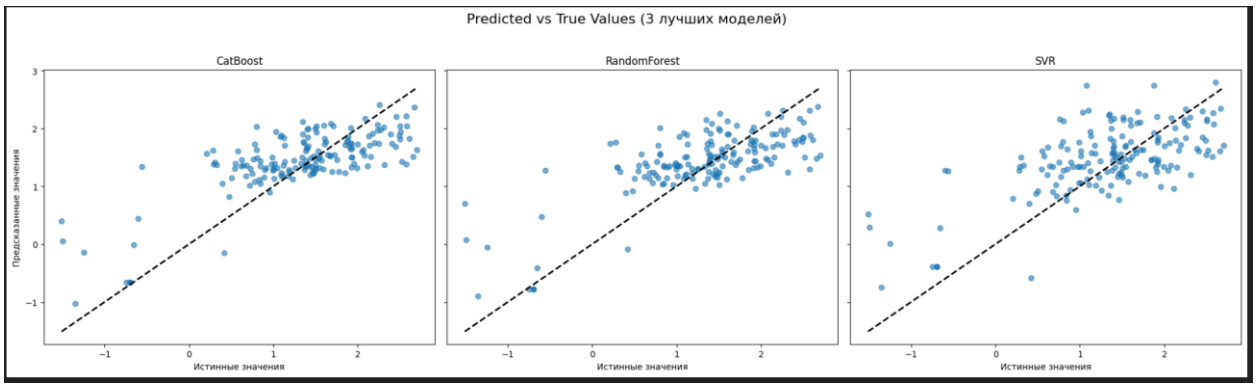
Метрики для каждой модели с наиболее удачными наборами гиперпараметров.

	Model	Best Params	MAE	RMSE	R2
3	CatBoost	{'model': <catboost.core.CatBoostRegressor obj...	0.440335	0.569124	0.539629
2	RandomForest	{'model': RandomForestRegressor(n_jobs=-1, ran...	0.446543	0.580440	0.521141
5	SVR	{'model': SVR(max_iter=2000), 'model__C': 7.89...	0.507152	0.648014	0.403155
0	LinearRegression	{'model': LinearRegression()}	0.503099	0.652469	0.394919
1	DecisionTree	{'model': DecisionTreeRegressor(random_state=4...	0.543274	0.683518	0.335961
4	MLP	{'model': MLPRegressor(max_iter=500, random_st...	0.575479	0.770097	0.157083

Графики сравнения моделей по метрикам



Графики предсказанных против истинных значений 3х лучших моделей



3. Задачи бинарной классификации

Для решения задач бинарной классификации будут использоваться такие модели:

- **Логистическая регрессия** (Logistic Regression)
- **Дерево решений** (Decision Tree Classifier)
- **Случайный лес** (Random Forest Classifier)
- **Градиентный бустинг деревьев решений (**CatBoost Classifier)
- **Многослойный перцептрон (**MLP Classifier**)**
- **Метод опорных векторов** (SVC / Support Vector Classifier)

Сравнивать данные модели я буду по следующим метрикам:

- **Accuracy** - простая метрика, % правильных ответов. Но может быть обманчива, если классы не сбалансированы.
- **Precision / Recall / F1** - стандарт для несбалансированных данных.
- **ROC-AUC** - полезно для бинарной классификации, показывает, насколько модель хорошо ранжирует вероятности.

Для подбора лучших гиперпараметров будет использоваться следующий словарь

```python
# словарь моделей и их гиперпараметров
param_grids = {
    "LogisticRegression": {
        "model": [LogisticRegression(max_iter=1000, solver="liblinear")],
        "model__C": uniform(0.01, 10),
        "model__penalty": ["l1", "l2"]
    },
    "DecisionTree": {
        "model": [DecisionTreeClassifier(random_state=42)],
        "model__max_depth": randint(3, 20),
        "model__min_samples_split": randint(2, 30),
        "model__min_samples_leaf": randint(1, 15)
    },
    "RandomForest": {
        "model": [RandomForestClassifier(random_state=42, n_jobs=-1)],
        "model__n_estimators": randint(200, 800),
        "model__max_depth": randint(5, 30),
        "model__max_features": ["sqrt", "log2", None],
        "model__min_samples_split": randint(2, 20)
    },
    "CatBoost": {
        "model": [CatBoostClassifier(verbose=0, random_state=42)],
        "model__depth": randint(4, 10),
        "model__learning_rate": uniform(0.01, 0.2),
        "model__iterations": randint(200, 800),
        "model__l2_leaf_reg": uniform(1.0, 9.0)
    },
    "MLP": {
        "model": [MLPClassifier(max_iter=600, random_state=42, early_stopping=True)],
        "model__hidden_layer_sizes": [(64,32), (128,64), (256,128)],
        "model__alpha": uniform(1e-4, 1e-2),
        "model__learning_rate_init": uniform(1e-3, 5e-2)
    },
    "SVC": {
        "model": [SVC(probability=True, max_iter=2000)],
        "model__kernel": ["rbf", "linear"],
        "model__C": uniform(0.1, 10.0),
        "model__gamma": ["scale", "auto"]
    }
}
```

✓ 0.0s

3.1. Предсказание: IC50 > медианы

Метрики для каждой модели с наиболее удачными наборами гиперпараметров.

	Model	Best Params	Accuracy	Precision	Recall	F1	ROC-AUC
3	CatBoost	{'model': <catboost.core.CatBoostClassifier ob...	0.701493	0.672414	0.78	0.722222	0.791535
5	SVC	{'model': SVC(max_iter=2000, probability=True)...	0.691542	0.663793	0.77	0.712963	0.749653
4	MLP	{'model': MLPClassifier(early_stopping=True, m...	0.676617	0.647059	0.77	0.703196	0.743713
0	LogisticRegression	{'model': LogisticRegression(max_iter=1000, so...	0.681592	0.660714	0.74	0.698113	0.750644
2	RandomForest	{'model': RandomForestClassifier(n_jobs=-1, ra...	0.666667	0.646018	0.73	0.685446	0.755495
1	DecisionTree	{'model': DecisionTreeClassifier(random_state=...	0.621891	0.593750	0.76	0.666667	0.673168

Графики сравнения моделей по метрикам

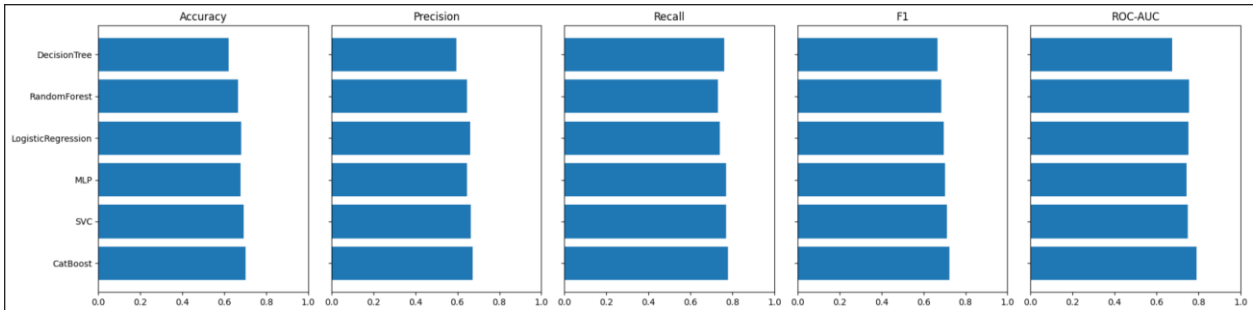
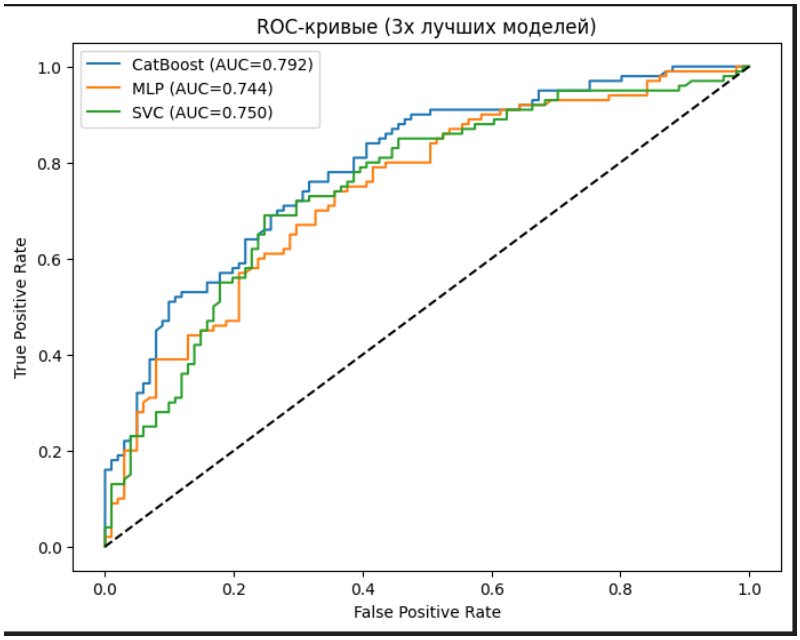


График ROC кривых



3.2. Предсказание: CC50 > медианы

Метрики для каждой модели с наиболее удачными наборами гиперпараметров.

	Model	Best Params	Accuracy	Precision	Recall	F1	ROC-AUC
0	LogisticRegression	{'model': LogisticRegression(max_iter=1000, so...	0.731343	0.694915	0.82	0.752294	0.821485
5	SVC	{'model': SVC(max_iter=2000, probability=True)...	0.736318	0.711712	0.79	0.748815	0.823069
4	MLP	{'model': MLPClassifier(early_stopping=True, m...	0.716418	0.677686	0.82	0.742081	0.832772
2	RandomForest	{'model': RandomForestClassifier(n_jobs=-1, ra...	0.726368	0.710280	0.76	0.734300	0.845941
1	DecisionTree	{'model': DecisionTreeClassifier(random_state=...	0.726368	0.718447	0.74	0.729064	0.795693
3	CatBoost	{'model': <catboost.core.CatBoostClassifier ob...	0.696517	0.663866	0.79	0.721461	0.825248

Графики сравнения моделей по метрикам

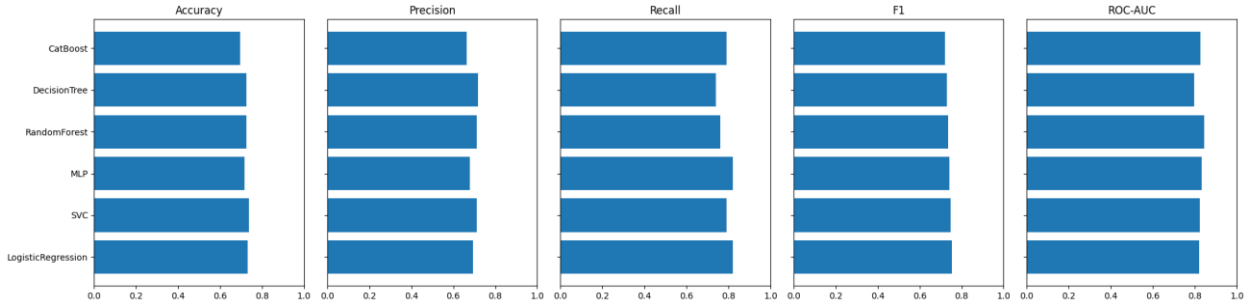
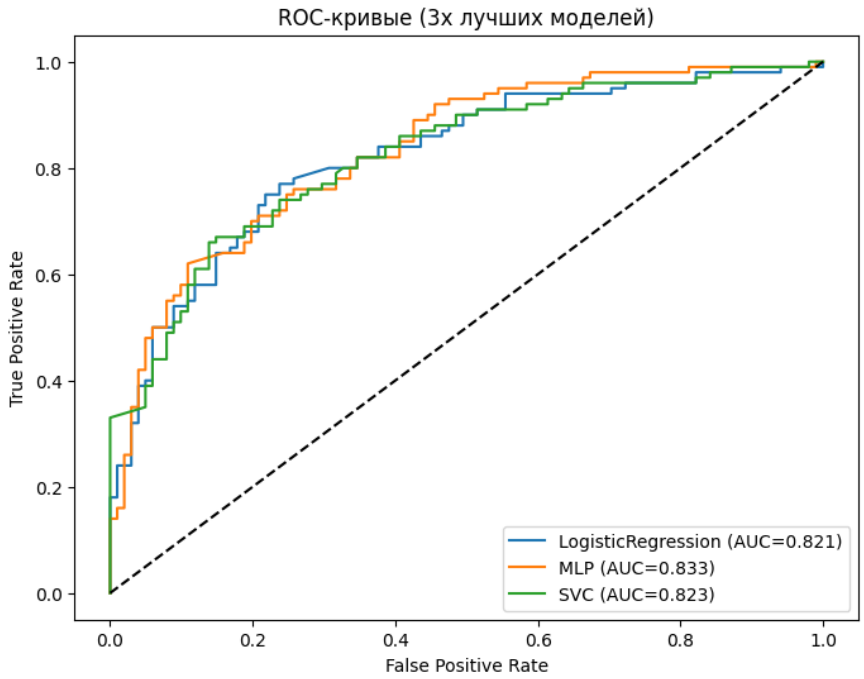


График ROC кривых



3.3. Предсказание: SI > медианы

Метрики для каждой модели с наиболее удачными наборами гиперпараметров.

	Model	Best Params	Accuracy	Precision	Recall	F1	ROC-AUC
5	SVC	{'model': SVC(max_iter=2000, probability=True)...	0.626866	0.621359	0.64	0.630542	0.677871
2	RandomForest	{'model': RandomForestClassifier(n_jobs=-1, ra...	0.631841	0.641304	0.59	0.614583	0.682624
3	CatBoost	{'model': <catboost.core.CatBoostClassifier ob...	0.611940	0.617021	0.58	0.597938	0.657772
0	LogisticRegression	{'model': LogisticRegression(max_iter=1000, so...	0.592040	0.588235	0.60	0.594059	0.632921
1	DecisionTree	{'model': DecisionTreeClassifier(random_state=...	0.587065	0.585859	0.58	0.582915	0.609802
4	MLP	{'model': MLPClassifier(early_stopping=True, m...	0.616915	0.709091	0.39	0.503226	0.677178

Графики сравнения моделей по метрикам

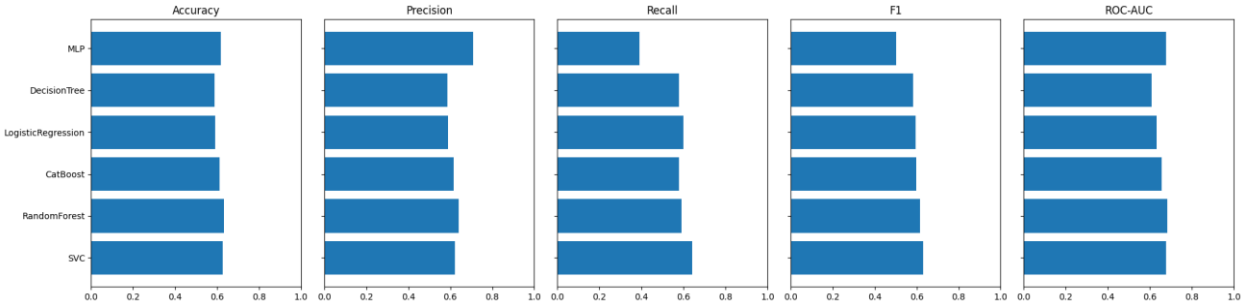
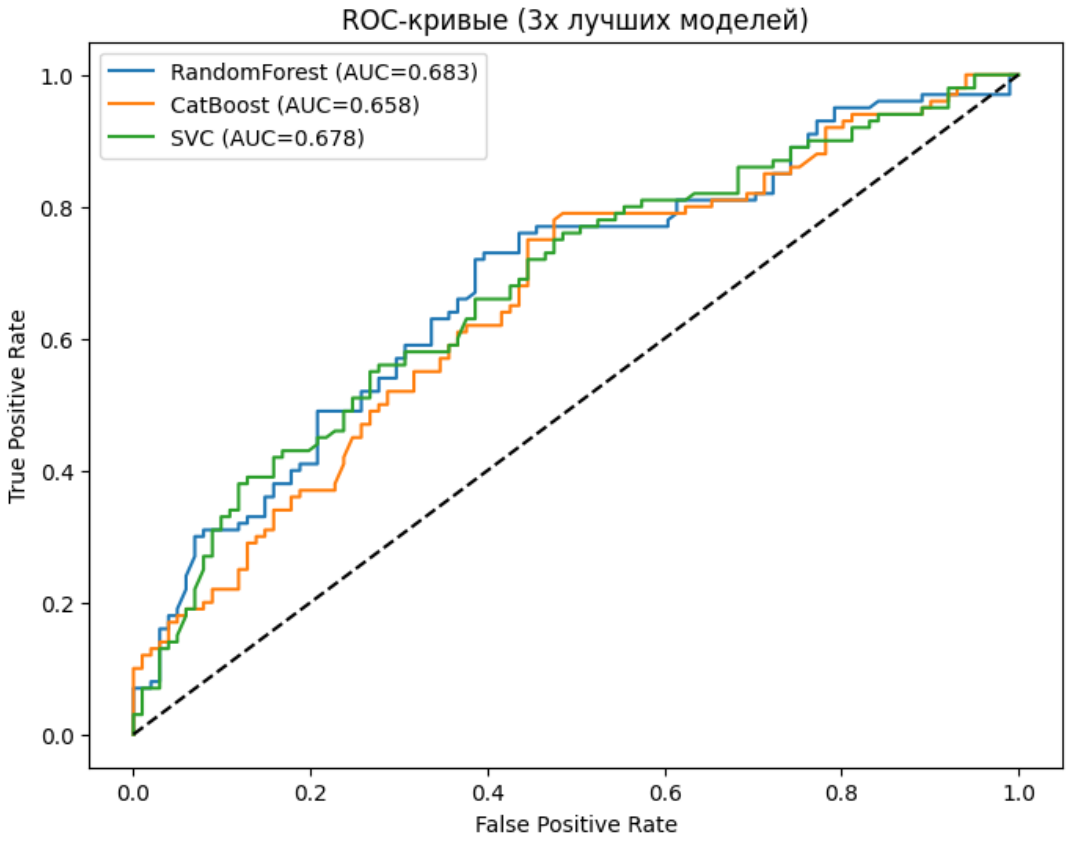


График ROC кривых



3.4. Предсказание: SI > 8

Метрики для каждой модели с наиболее удачными наборами гиперпараметров.

	Model	Best Params	Accuracy	Precision	Recall	F1	ROC-AUC
1	DecisionTree	{'model': DecisionTreeClassifier(random_state=...	0.721393	0.629032	0.541667	0.582090	0.724752
3	CatBoost	{'model': <catboost.core.CatBoostClassifier ob...	0.706468	0.603175	0.527778	0.562963	0.708710
5	SVC	{'model': SVC(max_iter=2000, probability=True)...	0.691542	0.580645	0.500000	0.537313	0.687823
2	RandomForest	{'model': RandomForestClassifier(n_jobs=-1, ra...	0.701493	0.607143	0.472222	0.531250	0.728090
4	MLP	{'model': MLPClassifier(early_stopping=True, m...	0.676617	0.553846	0.500000	0.525547	0.680233
0	LogisticRegression	{'model': LogisticRegression(max_iter=1000, so...	0.676617	0.559322	0.458333	0.503817	0.662952

Графики сравнения моделей по метрикам

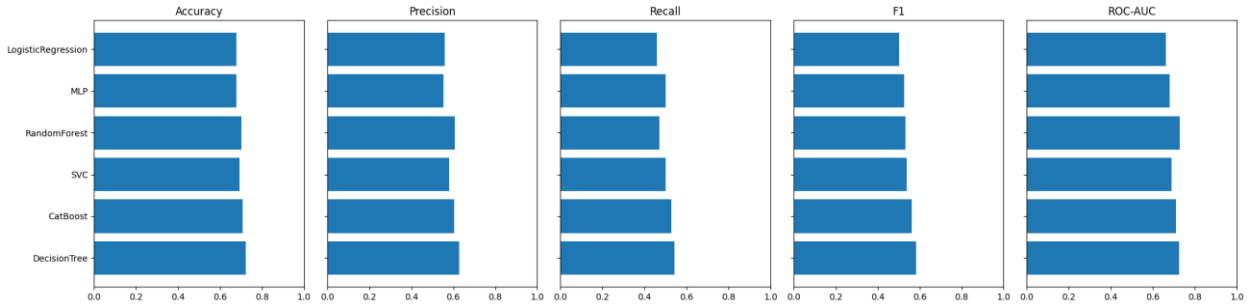
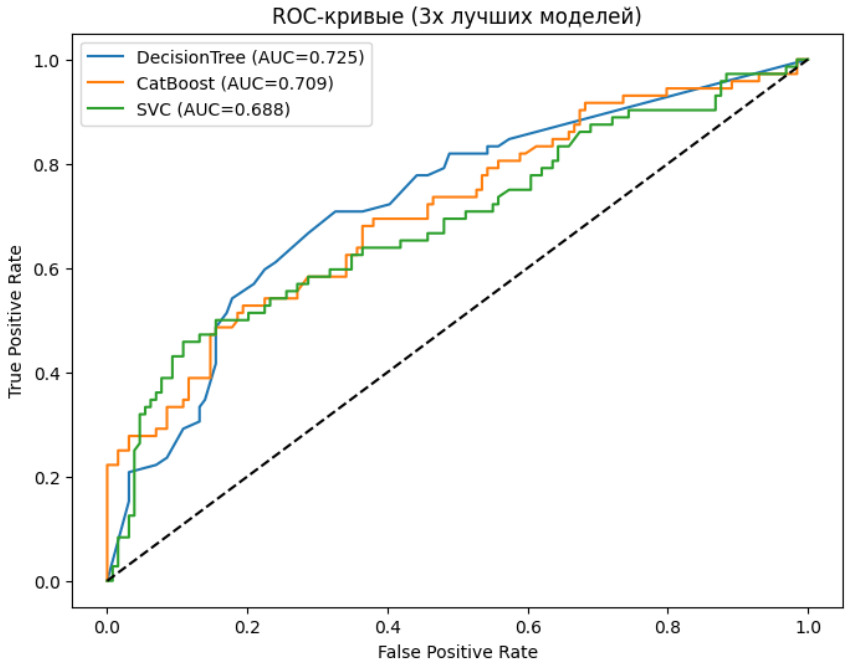


График ROC кривых



4. Вывод

Как для задач классификации, так и для задач регрессии CatBoost показывает хорошие результаты по большинству метрик (часто лучшие).

Для дальнейшего улучшения можно выполнить следующие шаги:

- Расширить гиперпараметрический поиск для моделей с лучшими результатами по каждой конкретной задаче.
- Попробовать использовать ансамблирование.
- протестировать дополнительные бустинговые алгоритмы (например, LightGBM, XGBoost) для сравнения.
- Побробовать построить новые производные признаки, чтобы усилить сигнал в данных.

На данном этапе, сложно выявить однозначно лучшую модель.

Однако можно подчеркнуть, что простые модели, такие как линейная регрессия (LinearRegression) и дерево решений (DecisionTree) показали неспособность выявить сложные зависимости в данных. Это касается задач регрессии.

Для задач бинарной классификации простые модели (DecisionTree и LogisticRegression) зачастую уступают более сложным методам.

Также, для задач классификации стоит попробовать ансамблирование SVC и CatBoost, т.к. они показывают хорошие результаты.