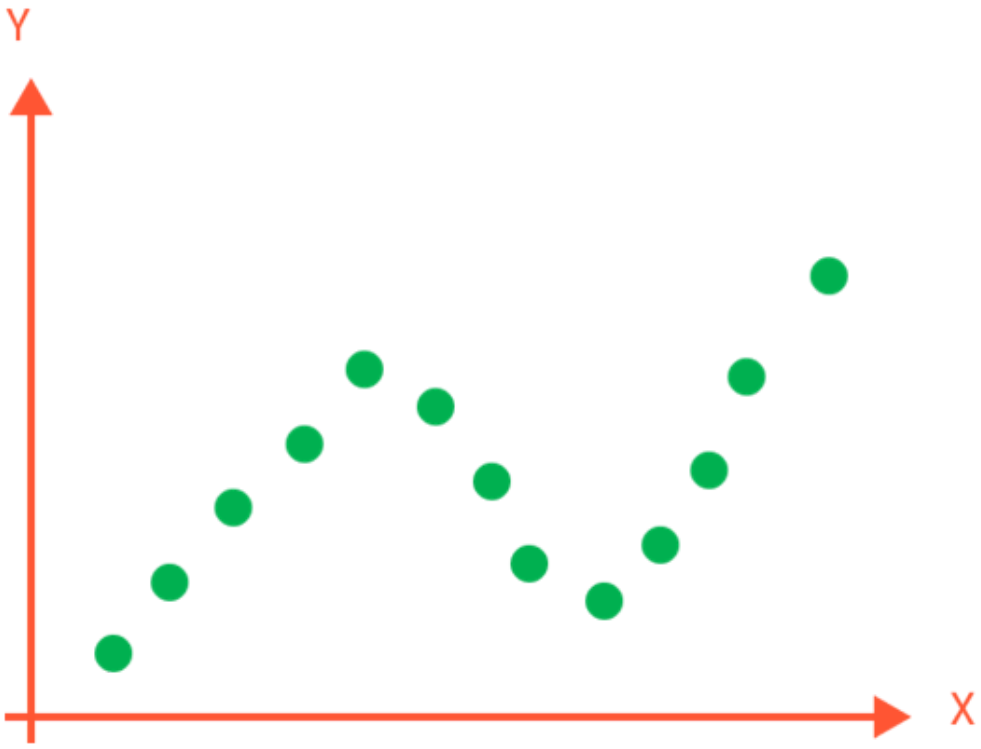
KARPOV.COURSES >>>
КОНСПЕКТ

Конспект > 18 урок > Решающее дерево: проблемы с обобщающей способностью и подбор гиперпараметров

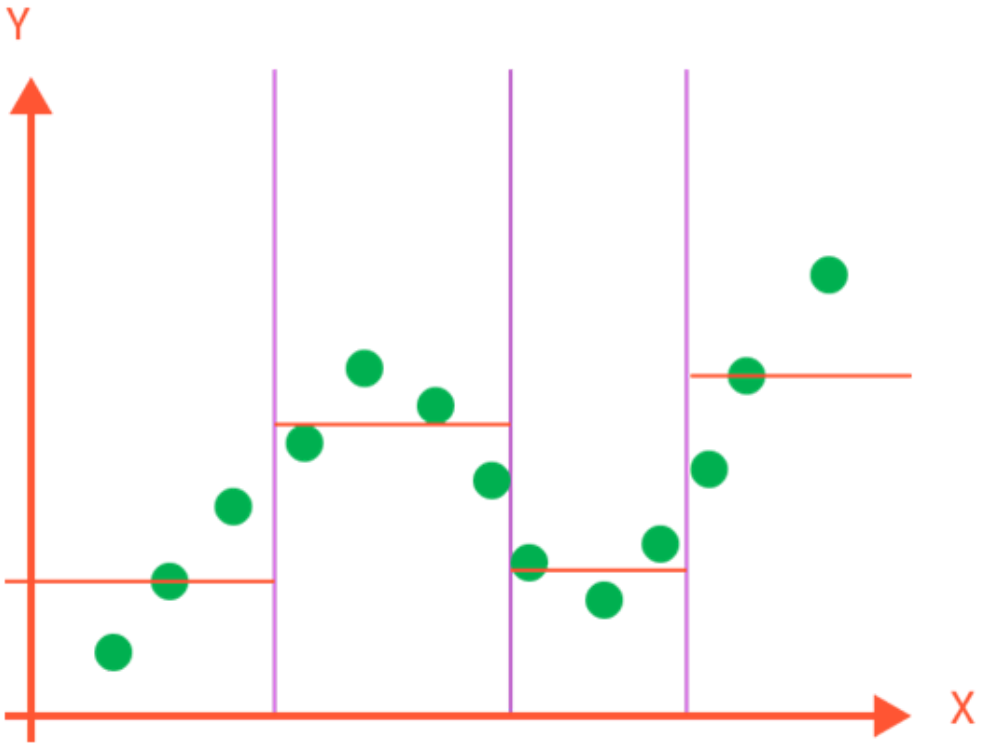
>Сила решающих деревьев

Решающие деревья - мощные модели, способные добиваться идеального качества на тренировочной выборке.

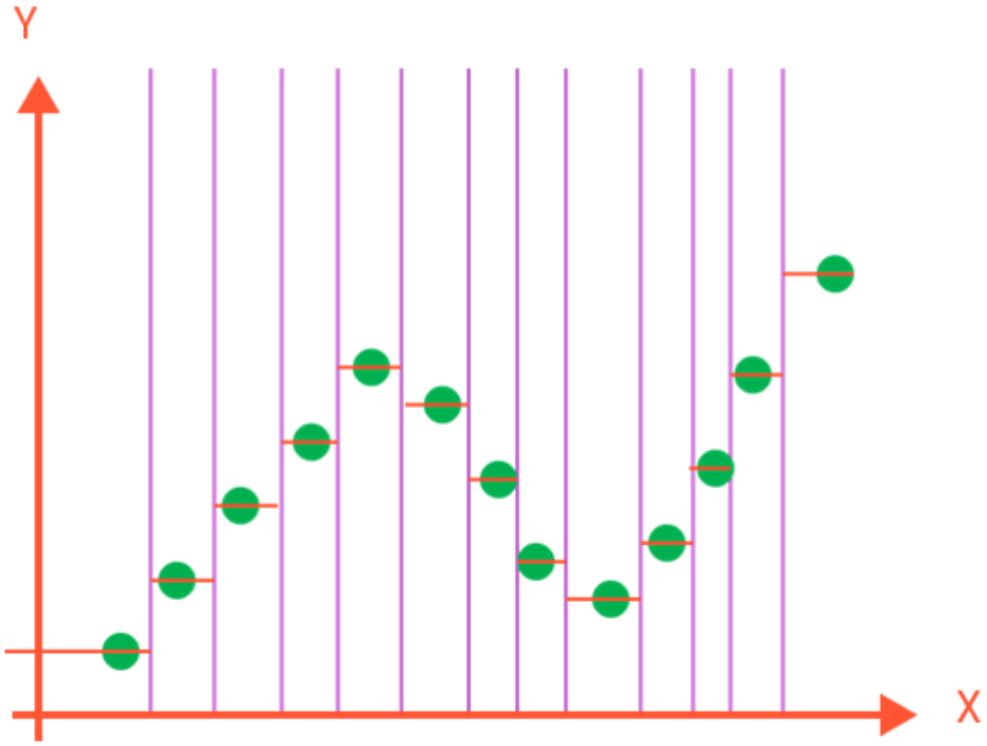
-Пусть имеем выборку, где у каждого объекта всего один признак и нелинейная зависимость от таргета



-Предикаты для бинарного дерева на такой выборке будут вида $[X \leq t]$

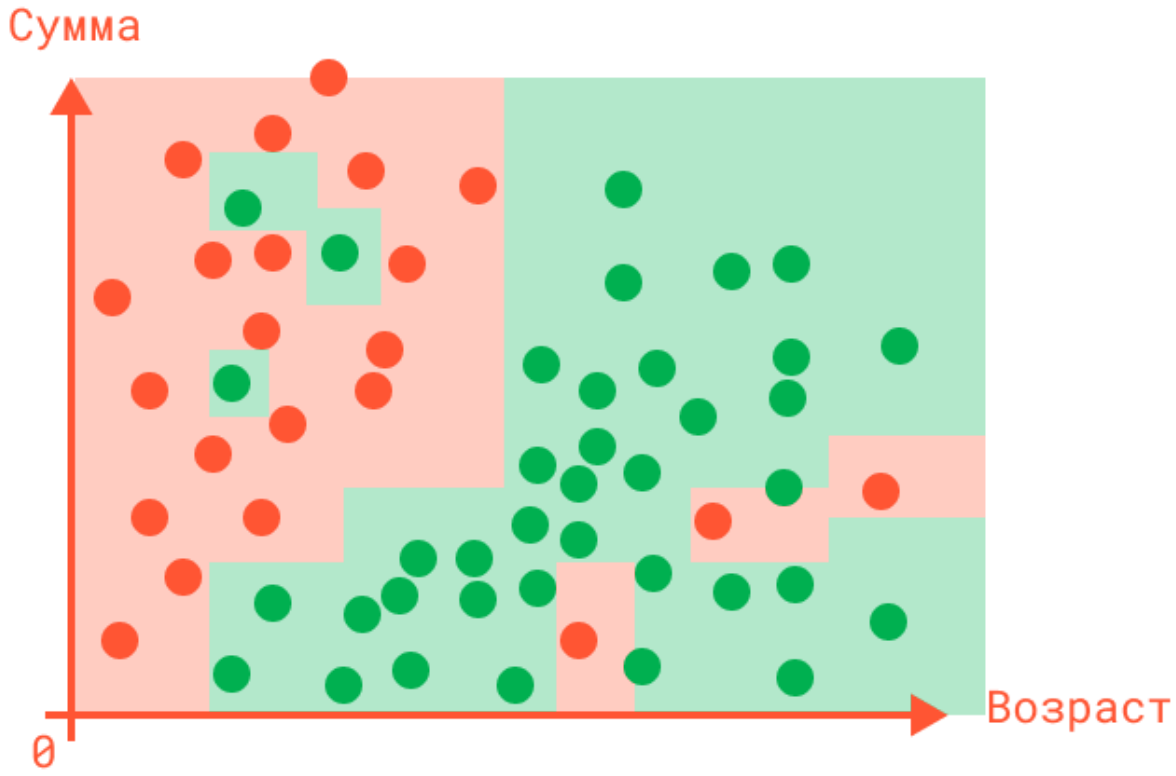


-Если в тренировочной выборке m точек, то максимум m на таких предикатах $\{[X \leq t_i]\}_i^m$ хватит для идеального прогноза

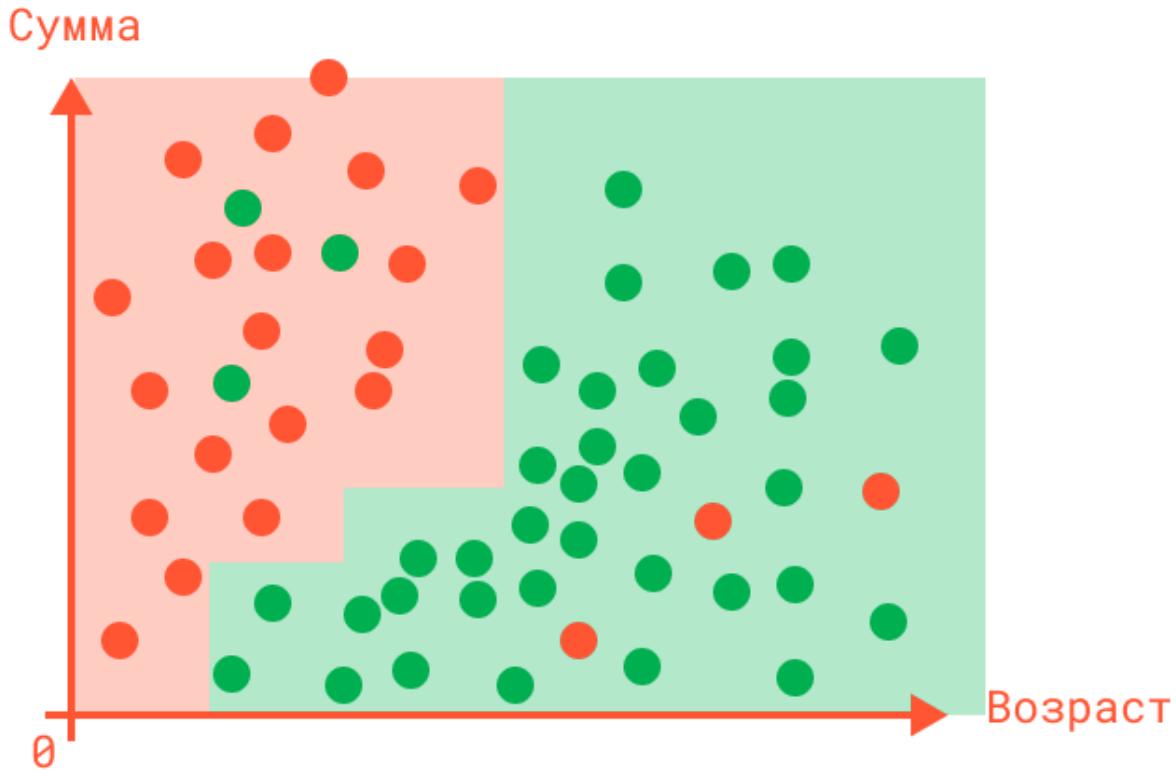


При этом основная проблема таких моделей - это их склонность к переобучению.

Поэтому необходимо как-то их сдерживать и регулировать.



Переобученная модель



Оптимальная модель

>Методы борьбы с переобучением решающих деревьев

Критерии останова

Основные признаки переобученного дерева решений есть ряд признаков, которые можно использовать в качестве гиперпараметров.

Гиперпараметр 1: Максимальная глубина дерева- у переобученного дерева очень большая глубина.

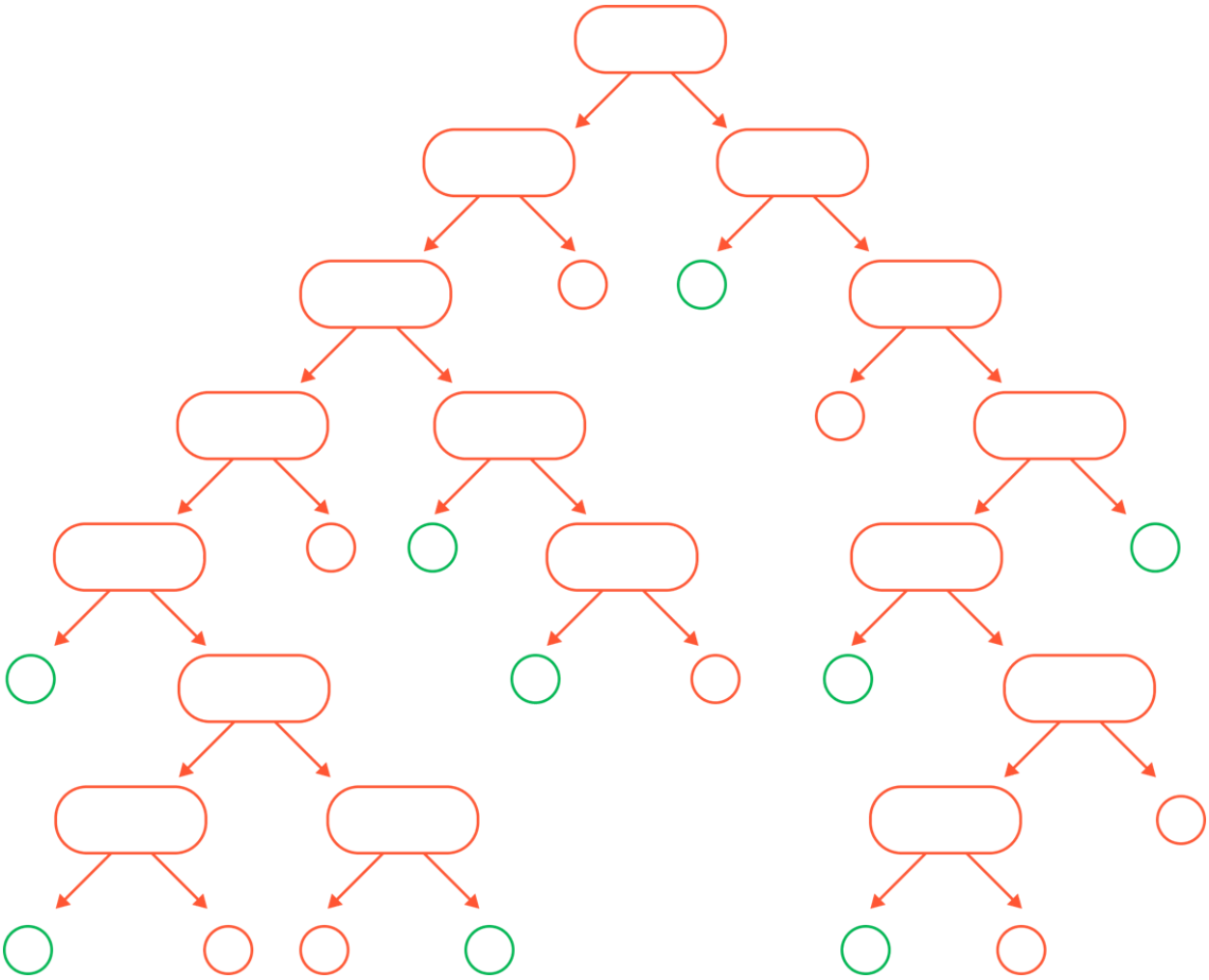
Гиперпараметр 2: Число объектов-признаков во внутренних вершинах- в переобученном дереве мало объектов во внутренних вершинах.

Гиперпараметр 3: Число объектов в листовой вершине - в переобученном дереве мало объектов в листовых вершинах.

Гиперпараметр 4: Максимальное количество листов - в переобученнном дереве как правило огромное количество листов.

Гиперпараметр 5: Минимальный прирост качества. Требование, что функционал качества при дроблении улучшался как минимум на n процентов. Если с последующим разбиением на новые вершины энтропия уменьшается незначительно, то можно останавливать обучение.

Задача - найти лучшие комбинации этих гиперпараметров с помощью кросс-валидации.



Стрижка деревьев

Другой не столь часто используемый подход к борьбе с переобучением -

стрижка дерева. При использовании стрижки сначала строится переобученное дерево, после чего производится оптимизация его структуры с целью улучшения обобщающей способности

В настоящее время данный подход редко применяется и слабо реализован в библиотеках машинного обучения.

Одним из методов стрижки является cost-complexity pruning.

>Связь решающих деревьев с линейными моделями

Вспомним как деревья осуществляют прогноз:

- При построении дерева пространство признаков разбивается на множество непересекающихся областей $D_1, D_2, \ldots D_k$
- Каждой области соответствует определенный прогноз w_j.
- Когда модели поступает новый объект, он определяется в ту или иную область размеченного пространства.

В общем виде формула данной модели выглядит следующим образом:

$$a(x) = w_1 \cdot [x \in D_1] + w_2 \cdot [x \in D_2] + \ldots + w_k \cdot [x \in D_k]$$

,где $w_1, w_2, \ldots w_k$ - веса-прогнозы каждой области , $x \in D_k$ - индикатор того, принадлежит ли объект данной области пространства (1 - принадлежит, 0 - не принадлежит)

Теперь вспомним как строятся линейные модели. Имеются некоторые признаки , каждый признак домножается на какой-либо вес, результат суммируется. По сути тут происходит то же самое, только здесь мы не строим модель над базовыми признаками, а используем новые бинарные признаки принадлежности к определенным областям пространства, в зависимости от соответствия ряду правил.

>Практика: трансформация данных

`class CustomFunctionTransformer(BaseEstimator,TransformerMixin)` - помогает значительно сократить этап предобработки данных, позволяет использовать fit и transform минуя fit_transform, а также применять встроенные методы, такие как `get_params`.

Подробнее о методе в документации

Пример реализации класса:

```
from sklearn.base import BaseEstimator, TransformerMixin
import itertools

class CustomFunctionTransformer(BaseEstimator, TransformerMixin):

    def __init__(self,
```

```
                object_columns=[],
                target_name='unit_sales'):

        self.object_columns = object_columns
        self.target_name = target_name


    def fit(self, X,y):

        X_fit = X.copy()
        y_fit = y.copy()

        self.numeric_columns = [x for x in X_fit.columns if x not in self.object_columns]

        X_with_target = pd.concat((X_fit, y_fit), axis=1)

        ### Сгенерим колонки к которым применим One-Hot-Encoding
        self.cols_for_ohe = [col for col in self.object_columns
                             if
                             X_with_target[col].nunique() <= 10]

        ### Запомним все ohe колонки и их названия!
        self.ohe_names = {col : sorted([f"{col}_{value}" for value in X_with_target[col].unique()])
                          for col in self.cols_for_ohe}


        ### Сгенерим колонки к которым применим Mean-Target-Encoding
        self.cols_for_mte = [col for col in self.object_columns
                             if X_with_target[col].nunique() > 10]

        ### Посчитаем на валидации средние значения таргета
        self.dict_of_means = {col : X_with_target.groupby(col)[self.target_name].mean()
                              for col in self.cols_for_mte}


        return self

    def transform(self,X,y=None):

        X_ = X.copy()

        data_part = pd.get_dummies(X_[self.cols_for_ohe],
                                   prefix=self.cols_for_ohe)

        data_part_cols = data_part.columns

        X_ = X_.drop(self.cols_for_ohe, axis=1)
        X_ = pd.concat((X_, data_part), axis=1)


        for col in self.cols_for_mte:
                X_[col] = X_[col].map(self.dict_of_means[col])

                mean_value = transformer.dict_of_means[col].values.mean()

                X_[col] = X_[col].fillna(mean_value)
```

```
        all_ohe = list(itertools.chain(*list(transformer.ohe_names.values())))

        missing_columns = [x
                           for x in all_ohe
                           if x not in X_.columns
                           and
                           x not in self.numeric_columns]

        extra_columns = [x
                         for x in data_part_cols
                         if x not in all_ohe]

        ### Новые категории необходимо убрать
        X_ = X_.drop(extra_columns, axis=1)

        ### Отсутствующие категории (бинарные колонки)
        ### необходимо добавить: заполним их просто нулями

        if len(missing_columns) != 0:

            zeros = np.zeros((X_.shape[0], len(missing_columns)))
            zeros = pd.DataFrame(zeros,
                                 columns=missing_columns,
                                 index=X_.index)

            X_ = pd.concat((X_, zeros), axis=1)

        return X_[sorted(X_.columns)]
```

>Практика: Обучение решающего дерева. Кросс-валидация и подбор лучших параметров на данных с временной структурой.

Как помним, при обучении модели на данных, которые имеют временную структуру, нужно соблюдать временную консистентность, то есть не допускать, чтобы при обучении данные из будущего просачивались в прошлое. Поэтому обычные методы кросс-валидации здесь не подойдут. Для разбивки таких данных используется аналогичный метод TimeSeriesSplit.

Документация

Сначала обучим дерево:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.pipeline import Pipeline

pipe = Pipeline([("custom_transformer",
                  CustomFunctionTransformer(object_columns=object_cols,
                                            target_name='unit_sales')),


                 ("decision_tree",
                  DecisionTreeRegressor())])

pipe.fit(X_train, y_train)
```

Найдем лучшие параметры на валидации:

```
from sklearn.model_selection import TimeSeriesSplit

splitter = TimeSeriesSplit(n_splits=3)

from sklearn.model_selection import GridSearchCV

param_grid = {
    "decision_tree__max_depth": [3, 5, 10],
    "decision_tree__min_samples_split": [10, 200, 50000],
    "decision_tree__min_impurity_decrease": [0, 0.1],
    "decision_tree__max_leaf_nodes": [100, 1000]
}

### Передадим в GridSearchCV

search = GridSearchCV(pipe,
                      param_grid,
                      cv=splitter,
                      scoring='neg_mean_squared_error',
                      verbose=10)

search.fit(X_train, y_train)

print(f"Best parameter (CV score={search.best_score_:.5f}):")
print(search.best_params_)

print(f"Качество лучшей модели на финальном тесте: {search.score(X_test, y_test)}")
```

С помощью метода `feature_importances_` можно визуально оценить, насколько тот или иной признак оказались важным:

```
fig = plt.figure()
fig.set_size_inches(16, 20)
```

```
sns.barplot(x=search.best_estimator_['decision_tree'].feature_importances_,
            y=check_test.columns)


plt.show()
```