Лаба 4. Задание: 1) Выберите набор данных (датасет) для решения задачи классификации или регресии. 2) В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков. 3) С использованием метода train_test_split разделите выборку на обучающую и тестовую. 4) Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью трех подходящих для задачи метрик. 5) Постройте модель и оцените качество модели с использованием кросс-валидации. Проведите эксперименты с тремя различными стратегиями кросс-валидации. 6) Произведите подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации. 7) Повторите пункт 4 для найденного оптимального значения гиперпараметра K. Сравните качество полученной модели с качеством модели, полученной в пункте 4. 8) Постройте кривые обучения и валидации.

In [1]:
```
#!pip install lightgbm
```

In [2]:
```
import pandas as pd
import numpy as np
import lightgbm # сожрет все сырым и построит регрессионную модель, которая покажет важные фичи
                # чтобы дальше делать лабу только на них
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
```

In [3]:
```
store = pd.read_csv('./data/googleplaystore.csv')
```

In [4]:
```
store.head()
```

Out[4]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Conter Ratin
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyon
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyon
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyon

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Conter Ratin
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyon

Пусть Rating - целевая фича Пропуски в данных есть. Это хорошо. Сделаем сразу две части задания

In [5]:
```
# Все колонки, которые не являются числами, делаем категориальными:
for column in store.select_dtypes(include = ['object']).columns.tolist():
    store[column] = store[column].astype('category')
```

In [6]:
```
lgbm_regressor = lightgbm.LGBMRegressor().fit(store.loc[:, store.columns != 'Rating'], store['Rating'])
lgbm_regressor # построили сырую и простую модель, вставив на X все кроме целевой, а на y - "Rating"
```

Out[6]:
```
LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
       importance_type='split', learning_rate=0.1, max_depth=-1,
       min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
       n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
       random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
       subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

In [7]:
```
list_of_importances = list(zip(store.loc[:, store.columns != 'Rating'].columns.tolist(),
                               lgbm_regressor.feature_importances_))
list_of_importances = sorted(list_of_importances, key= lambda x: x[1], reverse= True) # список фич, отсортированных по важности
```

In [8]:
```
important_features = [x[0] for x in list_of_importances if x[1] > 20]
#important_features # оставим только важные фичи
```

In [9]:
```
important_features.extend(['Rating'])
store = store[important_features]
```

In [10]:
```
# заполнение пропусков
for column in store.select_dtypes(include = ['int64', 'float64']).columns.tolist():
    store[column] = store[column].fillna(store[column].mean())
for column in store.select_dtypes(include = ['category']).columns.tolist():
    store[column] = store[column].fillna(store[column].describe(include= ['category'])['top'])
```

In [11]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for column in store.select_dtypes(include = ['category']).columns.tolist():
    le.fit(store[column])
    store[column] = le.transform(store[column])
#кодирование категориальных признаков
store.head()
```

Out[11]:

	Current Ver	Reviews	Type	Rating
0	118	1182	1	4.1
1	1018	5923	1	3.9
2	464	5680	1	4.7
3	2765	1946	1	4.5
4	277	5923	1	4.3

In [12]:
```
store.loc[:, store.columns != 'Rating'] = store.loc[:, store.columns != 'Rating'].apply(lambda x: x/x.max(), axis=0)
# нормирование или масштабирование данных
```

In [13]:
```
#train
X_train, X_test, y_train, y_test = train_test_split(store.loc[:, store.columns != 'Rating'],
                                                    store['Rating'],
                                                    test_size= 0.1,
                                                    random_state= 42)
```

In [14]:
```
# подбор лучшего параметра по кросс-валидации
random_search = GridSearchCV(estimator= KNeighborsRegressor(),
                             param_grid= {'n_neighbors': [5,10,20,50]}, #несколько вариантов парам
етра k
                             scoring= 'neg_mean_absolute_error') #средняя абсолютная ошибка по мод
ели
random_search.fit(store.loc[:, store.columns != 'Rating'], store['Rating']) #обучаем fit
```

Out[14]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=5, p=2,
          weights='uniform'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_neighbors': [5, 10, 20, 50]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_mean_absolute_error', verbose=0)
```

In [15]:
```
random_search.best_params_ #лучшее с 20 соседями
```

Out[15]: {'n_neighbors': 50}

In [16]:
```
random_search.best_score_ #выведем получившуюся абсолютную ошибку
```

Out[16]: -0.32989175073647925