

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
им. Н.Э. Баумана

Факультет «Информатика и системы управления»  
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

**Лабораторная работа № 4**  
по дисциплине «Методы машинного обучения»

Тема: «Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей.»

ИСПОЛНИТЕЛЬ:  
группа ИУ5-22М

Егоров С.А.  
ФИО

подпись

"\_\_" \_\_\_\_ 2020 г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.  
ФИО

подпись

"\_\_" \_\_\_\_ 2020 г.

Москва - 2020

---

## Задание

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
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.

## Реализация задания

```
#Подключаем данные
data = pd.read_csv('SolarPrediction.csv', sep=",")
```

```
# Список колонок с типами данных
data.dtypes
```

```
UNIXTime          int64
Data              object
Time              object
Radiation         float64
Temperature       int64
Pressure          float64
Humidity          int64
WindDirection(Degrees) float64
Speed             float64
TimeSunRise       object
TimeSunSet        object
dtype: object
```

```
data.head()
```

	UNIXTime	Data	Time	Radiation	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed	TimeSunRise	TimeSunSet
0	1475229326	9/29/2016 12:00:00 AM	23:55:26	1.21	48	30.46	59	177.39	5.62	06:13:00	18:13:00
1	1475229023	9/29/2016 12:00:00 AM	23:50:23	1.21	48	30.46	58	176.78	3.37	06:13:00	18:13:00
2	1475228726	9/29/2016 12:00:00 AM	23:45:26	1.23	48	30.46	57	158.75	3.37	06:13:00	18:13:00
3	1475228421	9/29/2016 12:00:00 AM	23:40:21	1.21	48	30.46	60	137.71	3.37	06:13:00	18:13:00
4	1475228124	9/29/2016 12:00:00 AM	23:35:24	1.17	48	30.46	62	104.95	5.62	06:13:00	18:13:00

## Часть 1. Предварительная подготовка данных

Очевидно, что все эти временные характеристики в таком виде нам не особо интересны. Преобразуем все нечисловые столбцы в числовые. В целом колонка `UNIXTime` нам не интересна, дата скорее интереснее в виде дня в году. Время измерения может быть интересно в двух видах: просто секунды с полуночи, и время, нормализованное относительно рассвета и заката.

```
#Преобразуем временные колонки в соответствующий временной формат:
data["Time"] = (pd
    .to_datetime(data["UNIXTime"], unit="s", utc=True)
    .dt.tz_convert("Pacific/Honolulu")).dt.time

data["TimeSunRise"] = (pd
    .to_datetime(data["TimeSunRise"],
        infer_datetime_format=True)
    .dt.time)

data["TimeSunSet"] = (pd
    .to_datetime(data["TimeSunSet"],
        infer_datetime_format=True)
    .dt.time)

data = data.rename({"WindDirection(Degrees)": "WindDirection"},
    axis=1)
```

```
def time_to_second(t):
    return ((datetime.combine(datetime.min, t) - datetime.min)
        .total_seconds())
```

```
df = data.copy()

timeInSeconds = df["Time"].map(time_to_second)

sunrise = df["TimeSunRise"].map(time_to_second)
sunset = df["TimeSunSet"].map(time_to_second)
df["DayPart"] = (timeInSeconds - sunrise) / (sunset - sunrise)

df = df.drop(["UNIXTime", "Data", "Time",
    "TimeSunRise", "TimeSunSet"], axis=1)

df.head()
```

Результат преобразования данных:

	Radiation	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
0	1.21	48	30.46	59	177.39	5.62	1.475602
1	1.21	48	30.46	58	176.78	3.37	1.468588
2	1.23	48	30.46	57	158.75	3.37	1.461713
3	1.21	48	30.46	60	137.71	3.37	1.454653
4	1.17	48	30.46	62	104.95	5.62	1.447778

```
df.dtypes
```

```
Radiation      float64
Temperature     int64
Pressure        float64
Humidity        int64
WindDirection   float64
Speed           float64
DayPart         float64
dtype: object
```

```
df.shape
```

```
(32686, 7)
```

Проверим набор данных на наличие пустых значений:

```
# Проверим наличие пустых значений
df.isnull().sum()
```

```
Radiation      0
Temperature     0
Pressure        0
Humidity        0
WindDirection   0
Speed           0
DayPart         0
dtype: int64
```

```
# Основные статистические характеристики набора данных
df.describe()
```

	Radiation	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
count	32686.000000	32686.000000	32686.000000	32686.000000	32686.000000	32686.000000	32686.000000
mean	207.124697	51.103255	30.422879	75.016307	143.489821	6.243869	0.482959
std	315.916387	6.201157	0.054673	25.990219	83.167500	3.490474	0.602432
min	1.110000	34.000000	30.190000	8.000000	0.090000	0.000000	-0.634602
25%	1.230000	46.000000	30.400000	56.000000	82.227500	3.370000	-0.040139
50%	2.660000	50.000000	30.430000	85.000000	147.700000	5.620000	0.484332
75%	354.235000	55.000000	30.460000	97.000000	179.310000	7.870000	1.006038
max	1601.260000	71.000000	30.560000	103.000000	359.950000	40.500000	1.566061

## Часть 2. Разделение данных.

```
x = df.drop("Radiation", axis=1)
y = df["Radiation"]
print(x.head(), "\n")
print(y.head())
```

	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
0	48	30.46	59	177.39	5.62	1.475602
1	48	30.46	58	176.78	3.37	1.468588
2	48	30.46	57	158.75	3.37	1.461713
3	48	30.46	60	137.71	3.37	1.454653
4	48	30.46	62	104.95	5.62	1.447778

```
0    1.21
1    1.21
2    1.23
3    1.21
4    1.17
Name: Radiation, dtype: float64
```

```
print(x.shape)
print(y.shape)
```

```
(32686, 6)
(32686,)
```

```
columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X, columns=columns).describe()
```

	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
count	3.268600e+04	3.268600e+04	3.268600e+04	3.268600e+04	3.268600e+04	3.268600e+04
mean	5.565041e-16	2.904952e-14	1.391260e-17	6.956302e-17	-9.738822e-17	5.217226e-18
std	1.000015e+00	1.000015e+00	1.000015e+00	1.000015e+00	1.000015e+00	1.000015e+00
min	-2.758117e+00	-4.259540e+00	-2.578560e+00	-1.724255e+00	-1.78859e+00	-1.855112e+00
25%	-8.229646e-01	-4.184734e-01	-7.316829e-01	-7.366250e-01	-8.233591e-01	-8.683240e-01
50%	-1.779139e-01	1.302504e-01	3.841386e-01	5.062367e-02	-1.787376e-01	2.279483e-03
75%	6.283995e-01	6.789742e-01	8.458578e-01	4.307058e-01	4.658840e-01	8.682924e-01
max	3.208603e+00	2.508053e+00	1.076717e+00	2.602741e+00	9.814329e+00	1.797910e+00

```
#Разделим выборку на тренировочную и тестовую
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.25, random_state=346705925)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(24514, 6)
(8172, 6)
(24514,)
(8172,)
```

### Часть 3. Модель ближайших соседей для произвольно заданного гиперпараметра $K$ .

Параметр  $K$  возьмём равным 5.

```
def test_model(model):
    print("mean_absolute_error:",
          mean_absolute_error(y_test, model.predict(X_test)))
    print("median_absolute_error:",
          median_absolute_error(y_test, model.predict(X_test)))
    print("r2_score:",
          r2_score(y_test, model.predict(X_test)))
```

```
reg_5 = KNeighborsRegressor(n_neighbors=5)
reg_5.fit(X_train, y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

```
test_model(reg_5)
```

```
mean_absolute_error: 55.39857905041605
median_absolute_error: 4.0170000000000004
r2_score: 0.8677873476991447
```

Видно, что средние ошибки не очень показательны для одной модели, они больше подходят для сравнения разных моделей. В тоже время коэффициент детерминации неплох сам по себе, в данном случае модель более-менее состоятельна.

## Часть 4. Использование кросс-валидации

Проверим различные стратегии кросс-валидации.

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                          cv=KFold(n_splits=10), scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.83276085 0.5984654 0.83547149 0.75974839 0.76407458 0.81422383
 0.85420738 0.79432111 0.74927049 0.28234327]
0.7284886763598686 ± 0.16383980384698185
```

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                          cv=RepeatedKFold(n_splits=5, n_repeats=2),
                          scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.86237764 0.86301328 0.86540791 0.8721868 0.87241189 0.85995955
 0.85940263 0.86550803 0.87276485 0.86803623]
0.8661068822340072 ± 0.004814883174569271
```

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                          cv=ShuffleSplit(n_splits=10), scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.87920701 0.86987589 0.86841501 0.87145097 0.86677115 0.87814793
 0.8725625 0.85537219 0.86709691 0.87171571]
0.8700615268168288 ± 0.0063094303682732335
```

## Часть 5. Подбор гиперпараметра $K$

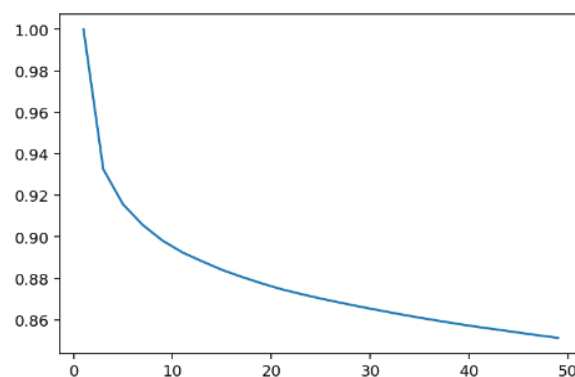
```
n_range = np.array(range(1, 50, 2))
tuned_parameters = [{'n_neighbors': n_range}]
n_range
```

```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
       35, 37, 39, 41, 43, 45, 47, 49])
```

```
gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_params_
```

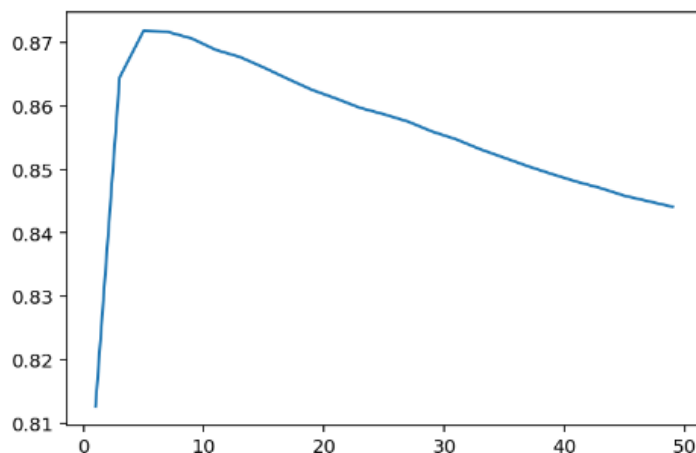
```
{'n_neighbors': 5}
```

```
plt.plot(n_range, gs.cv_results_["mean_train_score"]);
```



Очевидно, что для  $K = 1$  на тренировочном наборе данных мы находим ровно ту же точку, что и нужно предсказать, и чем больше её соседей мы берём — тем меньше точность. На тестовом наборе данных картина сильно интереснее:

```
plt.plot(n_range, gs.cv_results_["mean_test_score"]);
```



Выходит, что сначала соседей слишком мало (высоко влияние выбросов), а затем количество соседей постепенно становится слишком велико, и среднее значение по этим соседям всё больше и больше оттягивает значение от истинного.

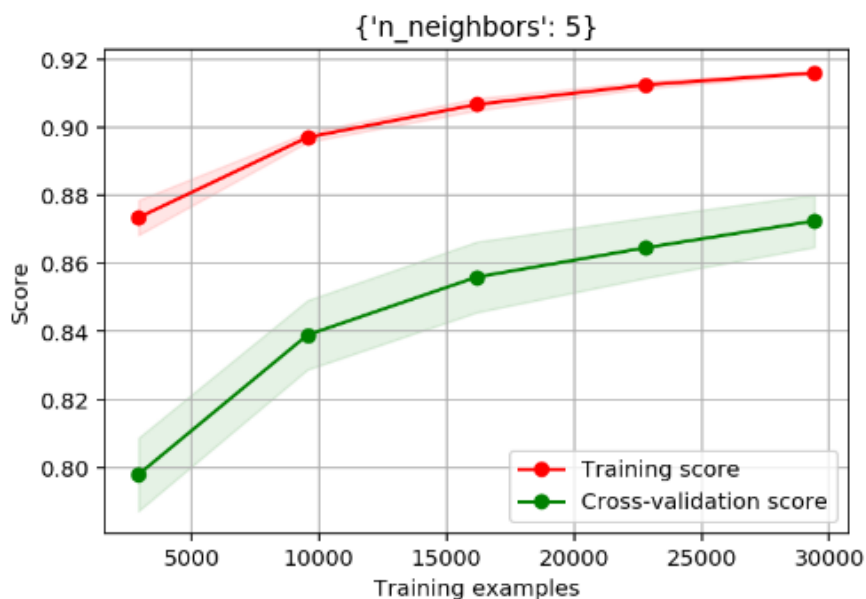
Проверим получившуюся модель:

```
reg = KNeighborsRegressor(**gs.best_params_)
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 55.39857905041605
median_absolute_error: 4.0170000000000004
r2_score: 0.8677873476991447
```

В целом получили примерно тот же результат. Очевидно, что проблема в том, что данный метод и так показал достаточно хороший результат для данной выборки.

Построим кривую обучения:



Построим кривую валидации:

