МГТУ им. Н. Э. Баумана, кафедра ИУ5

курс “Технология машинного обучения”

Лабораторная работа №5

«Линейные модели, SVM и деревья решений»

ВЫПОЛНИЛ:

Матюнин да Вейга Р.А.

Группа: ИУ5-61Б

ПРОВЕРИЛ:

Гапанюк Ю.Е.

Москва 2020

Цель лабораторной работы: изучение сложных способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей.

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 следующие модели:
   - одну из линейных моделей;
   - SVM;
   - дерево решений.
5. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

Выполненная работа

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
from sklearn import model_selection, linear_model, metrics

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

In [4]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Загрузка данных

In [5]:
```
import zipfile
import os
def fetch_data(path):
    z = zipfile.ZipFile(path, 'r')
    z.extractall(path=os.path.join(".", "dataset"))
    z.close()
```

In [6]:
```
path = 'dataset/gt.zip'
fetch_data(path)
```

In [7]:
```
raw_data = pd.read_csv('dataset/GlobalTemperatures.csv',sep=',')
```

In [8]:
```
raw_data.head()
```

Out[8]:

	dt	LandAverageTemperature	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncerta
0	1750-01-01	3.034	3.574	NaN	I
1	1750-02-01	3.083	3.702	NaN	I
2	1750-03-01	5.626	3.076	NaN	I
3	1750-04-01	8.490	2.451	NaN	I
4	1750-05-01	11.573	2.072	NaN	I

In [9]:
```
raw_data.shape
```

Out[9]:
```
(3192, 9)
```

In [10]:
```
raw_data.isnull().values.any()
```

Out[10]:
```
True
```

Предобработка данных

Типы признаков

In [11]:
```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3192 entries, 0 to 3191
Data columns (total 9 columns):
 #   Column                                     Non-Null Count  Dtype
---  ------                                     --------------  -----
 0   dt                                         3192 non-null   object
 1   LandAverageTemperature                     3180 non-null   float64
 2   LandAverageTemperatureUncertainty          3180 non-null   float64
 3   LandMaxTemperature                         1992 non-null   float64
 4   LandMaxTemperatureUncertainty              1992 non-null   float64
 5   LandMinTemperature                         1992 non-null   float64
 6   LandMinTemperatureUncertainty              1992 non-null   float64
 7   LandAndOceanAverageTemperature             1992 non-null   float64
 8   LandAndOceanAverageTemperatureUncertainty  1992 non-null   float64
dtypes: float64(8), object(1)
memory usage: 224.6+ KB
```

In [16]:
```
raw_data.dt = raw_data.dt.apply(pd.to_datetime)
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3192 entries, 0 to 3191
Data columns (total 9 columns):
 #   Column                                     Non-Null Count  Dtype
---  ------                                     --------------  -----
 0   dt                                         3192 non-null   datetime64[ns]
 1   LandAverageTemperature                     3180 non-null   float64
 2   LandAverageTemperatureUncertainty          3180 non-null   float64
 3   LandMaxTemperature                         1992 non-null   float64
 4   LandMaxTemperatureUncertainty              1992 non-null   float64
 5   LandMinTemperature                         1992 non-null   float64
 6   LandMinTemperatureUncertainty              1992 non-null   float64
 7   LandAndOceanAverageTemperature             1992 non-null   float64
 8   LandAndOceanAverageTemperatureUncertainty  1992 non-null   float64
dtypes: datetime64[ns](1), float64(8)
memory usage: 224.6 KB
```

In [20]:
```
raw_data['month'] = raw_data.dt.apply(lambda x: x.month)
raw_data['day'] = raw_data.dt.apply(lambda x: x.day)
```

In [21]:
```
raw_data.head()
```

Out[21]:

	dt	LandAverageTemperature	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncerta
0	1750-01-01	3.034	3.574	NaN	I
1	1750-02-01	3.083	3.702	NaN	I
2	1750-03-01	5.626	3.076	NaN	I
3	1750-04-01	8.490	2.451	NaN	I
4	1750-05-01	11.573	2.072	NaN	I

Обучение и отложенный тест

Обучающая выборка для создания модели и обучения ее. Тестовая выборка - для проверки качества модели

In [22]:
```
train_data = raw_data.iloc[:-1000, :]
hold_out_test_data = raw_data.iloc[-1000:, :]
```

In [23]:
```
raw_data.shape, train_data.shape, hold_out_test_data.shape
```

Out[23]:
```
((3192, 12), (2192, 12), (1000, 12))
```

In [25]:
```
print('train period from {} to {}'.format(train_data.dt.min(), train_data.dt.max()))
print('evaluation period from {} to {}'.format(hold_out_test_data.dt.min(), hold_out_test_data.dt.max
```

```
train period from 1750-01-01 00:00:00 to 1932-08-01 00:00:00
evaluation period from 1932-09-01 00:00:00 to 2015-12-01 00:00:00
```

Данные и целевая функция

In [30]:
```
## обучение
train_labels = train_data['LandAverageTemperature'].values
train_data = train_data.drop(['dt', 'LandAverageTemperature'], axis=1)
train_data.head()
```

Out[30]:

	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandMinTem
0	3.574	NaN	NaN	NaN	
1	3.702	NaN	NaN	NaN	
2	3.076	NaN	NaN	NaN	
3	2.451	NaN	NaN	NaN	
4	2.072	NaN	NaN	NaN	

In [32]:
```
## test
test_labels = hold_out_test_data['LandAverageTemperature'].values
test_data = hold_out_test_data.drop(['LandAverageTemperature', 'dt'], axis=1)
test_data.head()
```

Out[32]:

	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandMin
2192	0.268	18.216	0.213	6.340	
2193	0.232	15.424	0.160	3.900	
2194	0.186	11.551	0.204	0.014	
2195	0.196	8.888	0.215	-2.071	
2196	0.226	7.379	0.207	-3.453	

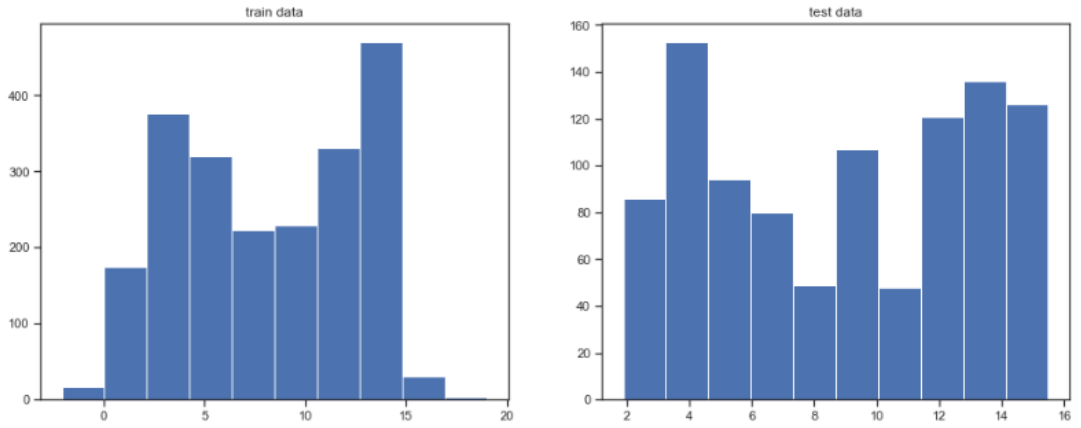
Целевая функция на обучающей выборке и на отложенном тесте

In [33]:
```
pylab.figure(figsize=(16,6))

pylab.subplot(1, 2, 1)
pylab.hist(train_labels)
pylab.title('train data')

pylab.subplot(1, 2, 2)
pylab.hist(test_labels)
pylab.title('test data')
```

Out[33]: Text(0.5, 1.0, 'test data')



Числовые признаки

In [36]:
```
inty', 'LandAndOceanAverageTemperature', 'LandAndOceanAverageTemperatureUncertainty', 'month', 'day']
```

In [37]:
```
train_data = train_data[numeric_columns]
test_data = test_data[numeric_columns]
```

In [38]:
```
train_data.head()
```

Out[38]:

	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandMinTem
0	3.574	NaN	NaN	NaN	
1	3.702	NaN	NaN	NaN	
2	3.076	NaN	NaN	NaN	
3	2.451	NaN	NaN	NaN	
4	2.072	NaN	NaN	NaN	

In [39]:
```
test_data.head()
```

Out[39]:

	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandMin
2192	0.268	18.216	0.213	6.340	
2193	0.232	15.424	0.160	3.900	
2194	0.186	11.551	0.204	0.014	
2195	0.196	8.888	0.215	-2.071	
2196	0.226	7.379	0.207	-3.453	

Модель

Так как у нас регрессия, то обучим регрессор. Моделью будет регрессор на основе стохастического градиентного спуска

In [24]:
```
regressor = linear_model.SGDRegressor(random_state=0)
```

In [25]:
```
regressor.fit(train_data, train_labels)
metrics.mean_absolute_error(test_labels, regressor.predict(test_data))
```

In [41]:
```
test_labels[:10]
```

Out[41]:
```
array([12.31 ,  9.682,  5.908,  3.527,  2.169,  2.657,  4.869,  8.125,
       11.047, 13.11 ])
```

In [42]:
```
regressor.predict(test_data)[:10]
```

Scaling

In [44]:
```
from sklearn.preprocessing import StandardScaler
```

Создаем ***scaler***

Чтобы применить наше преобразование, нужно сначала его обучить(то есть высчитать параметры μ и σ)

Обучать ***scaler*** можно только на обучающей выборке(потому что часто на практике нам неизвестна тестовая выборка)

In [45]:
```
## создаем scaler
scaler = StandardScaler()
scaler.fit(train_data, train_labels)
scaled_trained_data = scaler.transform(train_data)
scaled_test_data = scaler.transform(test_data)
```

Теперь можно снова обучить модель

In [46]:
```
regressor.fit(scaled_trained_data, train_labels)
metrics.mean_absolute_error(test_labels, regressor.predict(scaled_test_data))
```

Ошибка получилось очень маленька

Посмотрим на целефую функцию и наши прогнозы

In [47]:
```
test_labels[:10]
```

Out[47]:
```
array([12.31 ,  9.682,  5.908,  3.527,  2.169,  2.657,  4.869,  8.125,
       11.047, 13.11 ])
```

In [48]:
```
regressor.predict(scaled_test_data)[:10]
```

In [51]:
```
train_data.head()
```

Out[51]:

	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandMinTem
0	3.574	NaN	NaN	NaN	
1	3.702	NaN	NaN	NaN	
2	3.076	NaN	NaN	NaN	
3	2.451	NaN	NaN	NaN	
4	2.072	NaN	NaN	NaN	

In [52]:
```
train_labels[:10]
```

Out[52]:
```
array([ 3.034,  3.083,  5.626,  8.49 , 11.573, 12.937, 15.868, 14.75 ,
       11.413,  6.367])
```

In [53]:
```
np.all(train_data.LandMaxTemperature + train_data.registered == train_labels)
```

Pipilene

In [ ]:
```
from sklearn.pipeline import Pipeline
```

Вместо одного преобразования, ***Pipeline*** позволяет делать целую цепочку преобразований

- каждый шаг представляется ***tuple***
- первый элемент: название шага, второй элемент: объект, который способен преобразовывать данные
- ***главное, чтобы у объектов были такие методы как fit и $transform$***

In [ ]:
```
## создаем Pipeline из двух шагов: scaling и классификация
pipeline = Pipeline(steps=[('scaling', scaler), ('regression', regressor)])
```

In [ ]:
```
pipeline.fit(train_data, train_labels)
metrics.mean_absolute_error(test_labels, pipeline.predict(test_data))
```

Подбор параметров

Параметры будем подбирать по сетке

Посмотрим сначала как правильно к ним обращаться

In [ ]:
```
pipeline.get_params().keys()
```

In [ ]:
```
parameters_grid = {
    'regression__loss' : ['huber', 'epsilon_insensitive', 'squared_loss', ],
    'regression__max_iter' : [3, 5, 10, 50],
    'regression__penalty' : ['l1', 'l2', 'none'],
    'regression__alpha' : [0.0001, 0.01],
    'scaling__with_mean' : [0., 0.5],
}
```

SVM

In [ ]:
```
from sklearn.svm import LinearSVR, SVR, NuSVR
```

Объединяем отмасштабированные тренировачную и тестовую выборку в одну, чтобы показать на графике

In [ ]:
```
columns = ['LandAverageTemperatureUncertainty', 'LandMaxTemperature', 'LandMaxTemperatureUncertainty'
df_scaled_train_data = pd.DataFrame(scaled_train_data, columns=columns)
df_scaled_train_data.shape
```

In [ ]:
```
df_scaled_test_data = pd.DataFrame(scaled_test_data, columns=columns)
df_scaled_test_data.shape
```

In [ ]:
```
df_scaled_data = pd.concat((df_scaled_train_data, df_scaled_test_data))
df_scaled_data.shape
```

Объединяем метки

In [ ]:
```
labels = np.concatenate((train_labels,test_labels))
labels
```

In [ ]:
```
fig, ax = plt.subplots(figsize=(5,5))
sns.scatterplot(ax=ax, x=df_scaled_data.temp, y=labels)
```

In [ ]:
```
def plot_regr(clf):
    title = clf.__repr__
    clf.fit(df_scaled_data, labels)
    y_pred = clf.predict(df_scaled_data)
    fig, ax = plt.subplots(figsize=(5,5))
    ax.set_title(title)
    ax.plot(df_scaled_data.temp, labels, 'b.')
    ax.plot(df_scaled_data.temp, y_pred, 'ro')
    plt.show()
```


In [26]:
```
%%time
list_SVR = [LinearSVR(C=1.0, max_iter=10000),
            LinearSVR(C=1.0, loss='squared_epsilon_insensitive', max_iter=10000),
            SVR(kernel='linear', C=1.0),
           SVR(kernel='rbf', gamma=0.2, C=1.0),
           SVR(kernel='rbf', gamma=0.8, C=1.0),
           NuSVR(kernel='rbf', gamma=0.8, nu=0.1, C=1.0),
           NuSVR(kernel='rbf', gamma=0.8, nu=0.9, C=1.0),
           SVR(kernel='poly', degree=2, gamma='auto', C=1.0),
           SVR(kernel='poly', degree=3, gamma=0.2, C=1.0),
           SVR(kernel='poly', degree=4, gamma=0.2, C=1.0)]
for i, svm in enumerate(list_SVR):
    pylab.figure(figsize=(4,36))
    pylab.subplot(10, 1, i + 1)
    pylab.grid(True)
    svm.fit(df_scaled_data, labels)
    y_pred = svm.predict(df_scaled_data)
    plt.plot(df_scaled_data.temp, labels, 'b.')
    plt.plot(df_scaled_data.temp, y_pred, 'ro')
    print(i+1, ":", svm)
    pylab.title(svm.__repr__)
    plt.show()
#     plot_regr(svm)
```

In [27]:
```
def func():
    minimum = 100000.
    for i, svm in enumerate(list_SVR):
        svm.fit(scaled_train_data, train_labels)
        y_pred = svm.predict(scaled_test_data)
        print(str(i+1) + ") " + str(svm) + ":")
        MAE = metrics.mean_absolute_error(test_labels, y_pred)
        RMSE = np.sqrt(metrics.mean_squared_error(test_labels, y_pred))
        mean = (MAE + RMSE)/2.
        if float(mean) < float(minimum):
            minimum = mean
            best = (i+1, svm)
        print("MAE :", MAE)
        print("RMSE :", RMSE)
    print("The best is", best)
```

In [28]:
```
func()
```

Random Forest

In [29]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [30]:
```
regressor = RandomForestRegressor(random_state = 0, max_depth = 20, n_estimators = 50)
```

In [58]:
```
regressor.fit(train_data, train_labels)
y_pred = regressor.predict(test_data)
```

In [32]:
```
MAE = metrics.mean_absolute_error(test_labels, y_pred)
RMSE = np.sqrt(metrics.mean_squared_error(test_labels, y_pred))
print("MAE :", MAE)
print("RMSE :", RMSE)
```

In [33]:
```
pd.DataFrame([(pair[0], '{:.2f}'.format(pair[1])) for pair in zip(test_labels[:10],
                                                                  list(map(lambda x: round(x, 2), y_
```

In [34]:
```
pylab.figure(figsize=(16, 6))

pylab.subplot(1,2,1)
pylab.grid(True)
pylab.xlim(-100,1100)
pylab.ylim(-100,1100)
pylab.scatter(train_labels, grid_cv.best_estimator_.predict(train_data), alpha=0.5, color = 'red')
pylab.scatter(test_labels, grid_cv.best_estimator_.predict(test_data), alpha=0.5, color = 'blue')
pylab.title('linear model')

pylab.subplot(1,2,2)
pylab.grid(True)
pylab.xlim(-100,1100)
pylab.ylim(-100,1100)
pylab.scatter(train_labels, regressor.predict(train_data), alpha=0.5, color = 'red')
pylab.scatter(test_labels, regressor.predict(test_data), alpha=0.5, color = 'blue')
pylab.title('random forest model')
```

In [35]:
```
from sklearn.tree import DecisionTreeRegressor, export_graphviz
from IPython.display import Image
try:
    from StringIO import StringIO
except ImportError:
    from io import StringIO
import pydotplus
```

In [36]:
```
# Визуализация дерева
def get_png_tree(tree_model_param, feature_names_param):
    dot_data = StringIO()
    export_graphviz(tree_model_param, out_file=dot_data, feature_names=feature_names_param,
                    filled=True, rounded=True, special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return graph.create_png()
```

In [37]:
```
# dot_data = export_graphviz(regressor)
# graph = graphviz.Source(dot_data)
# graph
```

In [38]:
```
decision_tree = DecisionTreeRegressor(random_state=42, max_depth=8)
decision_tree.fit(train_data, train_labels)
y_pred = decision_tree.predict(test_data)
metrics.mean_absolute_error(test_labels, y_pred)
```

In [39]:
```
# Image(get_png_tree(decision_tree, train_data.columns), height="500")
```

In [40]:
```
path = os.path.join('dataset', 'GlobalTemperatures.csv')
os.remove(path)
```

Более качественной получилась

- Ноутбук с выполненной работой и отчет размещены в репозитории на github: https://github.com/Yorati/TMO