Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Отчет по ЛР №5

по курсу «Технологии машинного обучения»

«Линейные модели, SVM и деревья решений»

ИСПОЛНИТЕЛЬ:
Аушева Л.И.
Группа ИУ5-61Б

_____

"__"_____2020 г.

ПРЕПОДАВАТЕЛЬ:
Гапанюк Ю.Е.

_____

"__"_____2020 г.

Москва 2020

Цель лабораторной работы:

Изучение линейных моделей, SVM и деревьев решений.

Задание:

1. Выберите набор данных (Corruption_Perception_Index.csv) 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.

Sklearn

Bike sharing demand ¶

Задача на kaggle: https://www.kaggle.com/c/bike-sharing-demand

По историческим данным о прокате велосипедов и погодным условиям необходимо оценить спрос на прокат велосипедов.

В исходной постановке задачи доступно 11 признаков: https://www.kaggle.com/c/prudential-life-insurance-assessment/data

В наборе признаков присутсвуют вещественные, категориальные, и бинарные данные.

Библиотеки

In [2]:
```
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 [3]:
```
%pylab inline
```

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

Загрузка данных

In [4]:
```
raw_data = pd.read_csv('datasets/bike_sharing_demand.csv',sep=',')
```

In [5]:
```
raw_data.head()
```

Out[5]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

datetime - hourly date + timestamp

season - 1 = spring, 2 = summer, 3 = fall, 4 = winter

holiday - whether the day is considered a holiday

workingday - whether the day is neither a weekend nor holiday

weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

temp - temperature in Celsius

atemp - "feels like" temperature in Celsius

humidity - relative humidity

windspeed - wind speed

casual - number of non-registered user rentals initiated

registered - number of registered user rentals initiated

count - number of total rentals

In [6]:
```
raw_data.shape
```

Out[6]:
```
(10886, 12)
```

In [7]:
```
raw_data.isnull().values.any()
```

Out[7]:
```
False
```

Предобработка данных

Типы признаков

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   datetime    10886 non-null  object 
 1   season      10886 non-null  int64  
 2   holiday     10886 non-null  int64  
 3   workingday  10886 non-null  int64  
 4   weather     10886 non-null  int64  
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64  
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64  
 10  registered  10886 non-null  int64  
 11  count       10886 non-null  int64  
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

Поменяем тип у *datetime*. С object-> datetime

In [9]:
```
raw_data.datetime = raw_data.datetime.apply(pd.to_datetime)
```

Создадим два новых признака: *месяц и час, когда это происходит*

In [10]:
```
raw_data['month'] = raw_data.datetime.apply(lambda x: x.month)
raw_data['hour'] = raw_data.datetime.apply(lambda x: x.hour)
```

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

Out[11]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	month	hour
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	1	0
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	1	1
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	1	2
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13	1	3
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1	1	4

Обучение и отложенный тест

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

In [12]:
```
train_data = raw_data.iloc[:-1000, :]
hold_out_test_data = raw_data.iloc[-1000:, :]
```

In [13]:
```
raw_data.shape, train_data.shape, hold_out_test_data.shape
```

Out[13]:
```
((10886, 14), (9886, 14), (1000, 14))
```

In [14]:
```
print('train period from {} to {}'.format(train_data.datetime.min(), train_data.datetime.max()))
print('evaluation period from {} to {}'.format(hold_out_test_data.datetime.min(), hold_out_test_data.datetime.max()))
```

```
train period from 2011-01-01 00:00:00 to 2012-10-16 06:00:00
evaluation period from 2012-10-16 07:00:00 to 2012-12-19 23:00:00
```

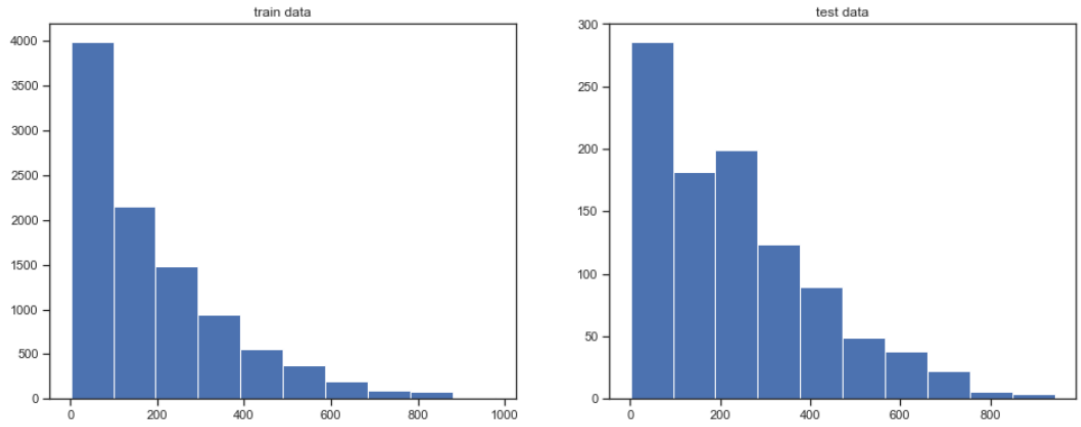
Целевая функция на обучающей выборке и на отложенном тесте

In [17]:
```
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[17]:
```
Text(0.5, 1.0, 'test data')
```



Модель

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

In [22]:
```
regressor = linear_model.SGDRegressor(random_state=0)
```

In [23]:
```
regressor.fit(train_data, train_labels)
metrics.mean_absolute_error(test_labels, regressor.predict(test_data))
```

Out[23]:
```
32678333066101.656
```

Невероятно большая ошибка.

Посмотрим на истинные метки, и наши

In [24]:
```
test_labels[:10]
```

Out[24]:
```
array([525, 835, 355, 222, 228, 325, 328, 308, 346, 446], dtype=int64)
```

In [25]:
```
regressor.predict(test_data)[:10]
```

Out[25]:
```
array([-8.15217468e+13, -1.27940348e+14, -5.15430820e+13, -2.29149119e+13,
       -2.58255957e+13, -3.85635403e+13, -3.92648763e+13, -3.03104198e+13,
       -4.03846575e+13, -5.57515848e+13])
```

У нас очень плохие значения.

Посмотрим на наши коэффициенты.

In [26]:
```
print(list(map(lambda x: round(x, 2), regressor.coef_)))
```

```
[-23028547689.43, -587204401.77, -12729175530.75, -1273594091.93, 58373018311.46, -160478942389.07, -36745954823.9, -487006372
8.49]
```

Коэффициенты просто нереальны. Это происходит из-за того, что мы не отмасштабировали признаки

Scaling

In [27]:
```
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 [28]:
```
## 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 [29]:
```
regressor.fit(scaled_trained_data, train_labels)
metrics.mean_absolute_error(test_labels, regressor.predict(scaled_test_data))
```

Out[29]:
```
0.042930483012408885
```

Ошибка получилось очень маленька

Мы ошибаемся меньше чем на 1 велосипед, это очень странно.

Посмотрим на коэффициенты

In [32]:
```
regressor.coef_
```

Out[32]:
```
array([ 4.58902678e-01, -4.51836158e-01,  6.62608792e-04, -1.40703258e-02,
        5.08590377e+01,  1.48008168e+02, -1.32281341e-03,  7.59230341e-03])
```

In [33]:
```
print(list(map(lambda x: round(x, 2), regressor.coef_)))
```

```
[0.46, -0.45, 0.0, -0.01, 50.86, 148.01, -0.0, 0.01]
```

Видно, что почти все признаки принимают маленькие коэффициенты, за исключением двух.

Посмотрим на эти признаки

In [34]:
```
train_data.head()
```

Out[34]:

	temp	atemp	humidity	windspeed	casual	registered	month	hour
0	9.84	14.395	81	0.0	3	13	1	0
1	9.02	13.635	80	0.0	8	32	1	1
2	9.02	13.635	80	0.0	5	27	1	2
3	9.84	14.395	75	0.0	3	10	1	3
4	9.84	14.395	75	0.0	0	1	1	4

Это признаки ***casual и registered***

Видно, что если сложить два эти признака, то получим целевую функцию

In [36]:
```
np.all(train_data.casual + train_data.registered == train_labels)
```

Out[36]:
```
True
```

То есть мы использовали те данные, по которым однозначно восстанавливается целевая функция

Удалим из нашей выборки эти признаки

In [37]:
```
train_data.drop(['casual', 'registered'], axis=1, inplace=True)
test_data.drop(['casual', 'registered'], axis=1, inplace=True)
```

Отмасштабируем признаки на новом наборе данных.

И обучим модель

In [38]:
```
scaler.fit(train_data, train_labels)
scaled_train_data = scaler.transform(train_data)
scaled_test_data = scaler.transform(test_data)
```

In [39]:
```
regressor.fit(scaled_train_data, train_labels)
metrics.mean_absolute_error(test_labels, regressor.predict(scaled_test_data))
```

Out[39]:
```
121.8835371361759
```

Мы построили модель. Теперь попытаемся ее улучшить

Для этого проведем кросс-валидацию. Но тут существует проблема: нам необходимо масштабировать данные. То есть нужно для каждого фолда провести масштабирование, а потом обучить его и проверить качество. Вся реализация будет очень громоздкой.

Благо, существует класс $Pipeline$

Pipilene

In [41]:
```
from sklearn.pipeline import Pipeline
```

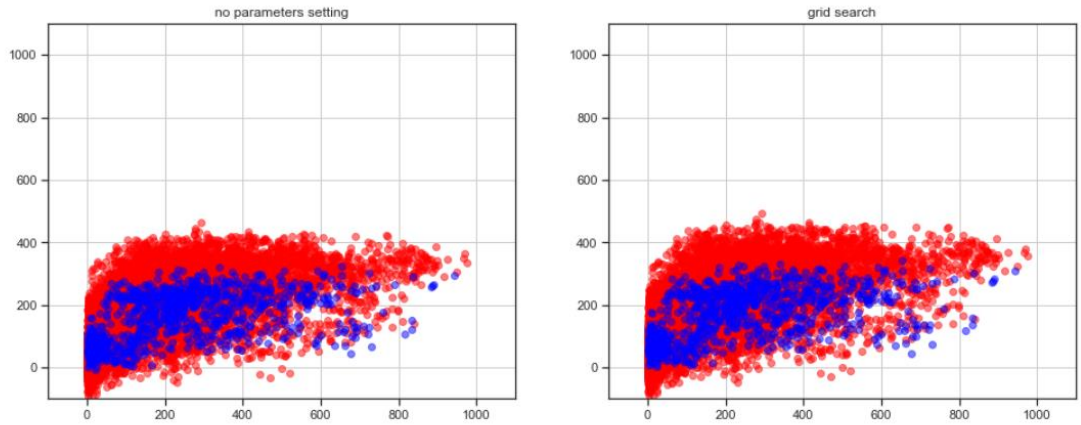
Вместо одного преобразования, $Pipeline$ позволяет делать целую цепочку преобразований

- 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 [42]:
```
## создаем Pipeline из двух шагов: scaling и классификация
pipeline = Pipeline(steps=[('scaling', scaler), ('regression', regressor)])
```

In [43]:
```
pipeline.fit(train_data, train_labels)
metrics.mean_absolute_error(test_labels, pipeline.predict(test_data))
```

Out[43]:
```
121.8835371361759
```



SVM

In [54]:
```
from sklearn.svm import LinearSVR, SVR, NuSVR
```

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

In [55]:
```
columns = ['temp', 'atemp', 'humidity', 'windspeed', 'month', 'hour']
df_scaled_train_data = pd.DataFrame(scaled_train_data, columns=columns)
df_scaled_train_data.shape
```

Out[55]:
```
(9886, 6)
```

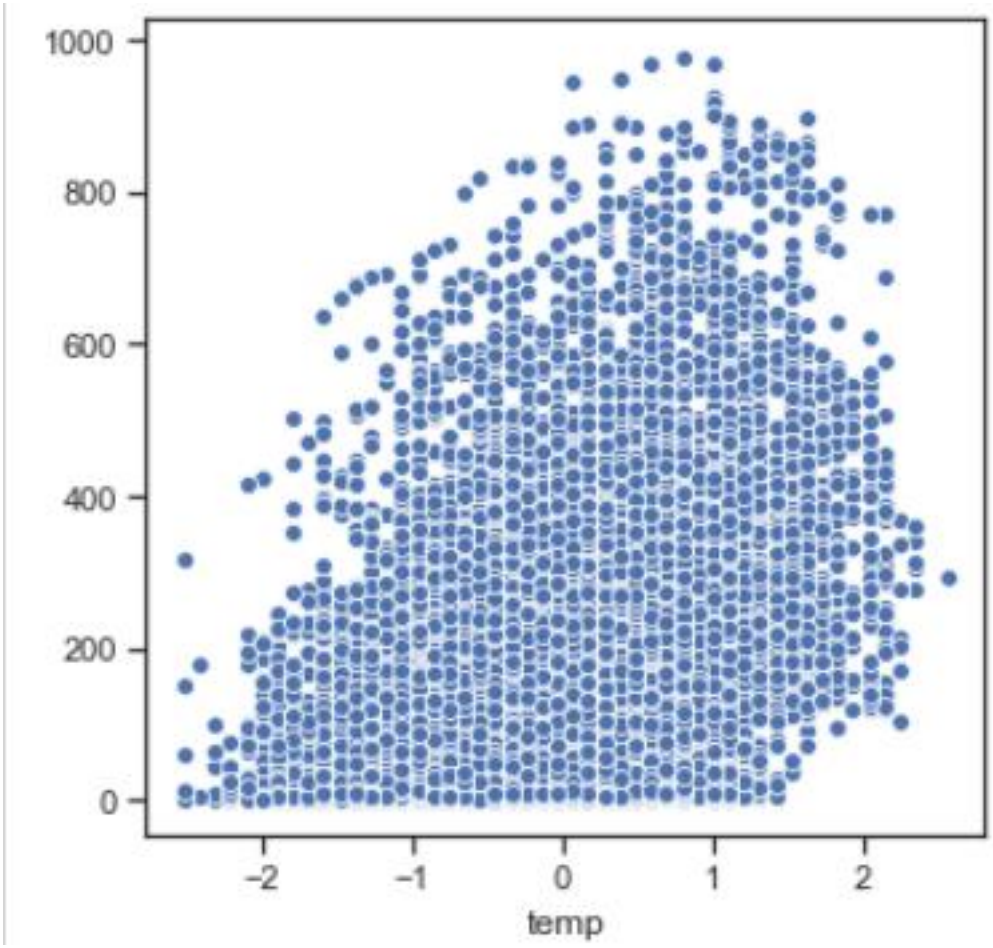
In [56]:
```
df_scaled_test_data = pd.DataFrame(scaled_test_data, columns=columns)
df_scaled_test_data.shape
```

Out[56]:
```
(1000, 6)
```

In [57]:
```
df_scaled_data = pd.concat((df_scaled_train_data, df_scaled_test_data))
df_scaled_data.shape
```

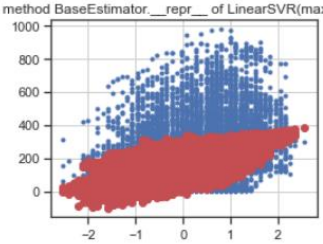
Out[57]:
```
(10886, 6)
```

Объединяем метки



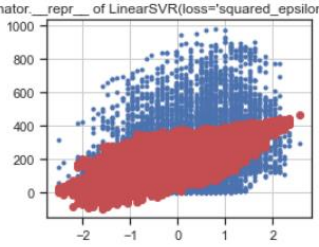

```
1 : LinearSVR(max_iter=10000)
```

```
<bound method BaseEstimator.__repr__ of LinearSVR(max_iter=10000)>
```




```
2 : LinearSVR(loss='squared_epsilon_insensitive', max_iter=10000)
```

```
<bound method BaseEstimator.__repr__ of LinearSVR(loss='squared_epsilon_insensitive', max_iter=10000)>
```



```
1) LinearSVR(max_iter=10000):
MAE : 134.62885617780657
RMSE : 198.57231537501926
2) LinearSVR(loss='squared_epsilon_insensitive', max_iter=10000):
MAE : 121.48121298033922
RMSE : 179.07353807208273
3) SVR(kernel='linear'):
MAE : 133.54396132008895
RMSE : 197.17923535229073
4) SVR(gamma=0.2):
MAE : 127.24055811808151
RMSE : 191.65174247313433
5) SVR(gamma=0.8):
MAE : 137.55045579588213
RMSE : 199.4544677227705
6) NuSVR(gamma=0.8, nu=0.1):
MAE : 156.07688650425644
RMSE : 188.5885130216648
7) NuSVR(gamma=0.8, nu=0.9):
MAE : 137.15587625226325
RMSE : 198.99307771502586
8) SVR(degree=2, gamma='auto', kernel='poly'):
MAE : 147.45771621073627
RMSE : 201.40462758624605
9) SVR(gamma=0.2, kernel='poly'):
MAE : 135.64994483200658
RMSE : 196.1028204855855
10) SVR(degree=4, gamma=0.2, kernel='poly'):
MAE : 133.0989147717965
RMSE : 194.07532401041914
The best is (2, LinearSVR(loss='squared_epsilon_insensitive', max_iter=10000))
```

При SVM мы все равно ошибаемся в среднем на 121 велосипед

Random Forest

In [64]:
```
from sklearn.ensemble import RandomForestRegressor
```

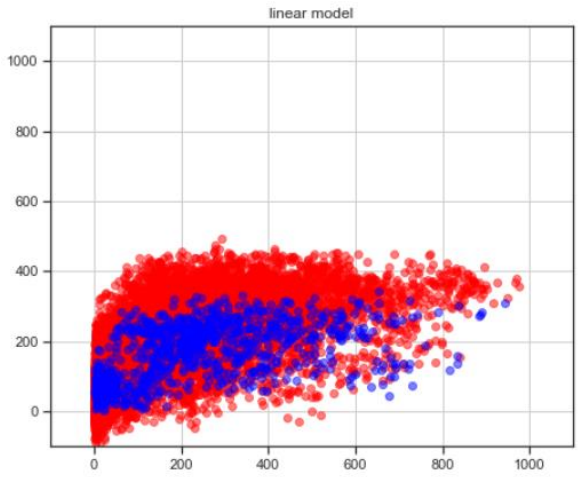
In [65]:
```
regressor = RandomForestRegressor(random_state = 0, max_depth = 20, n_estimators = 50)
```

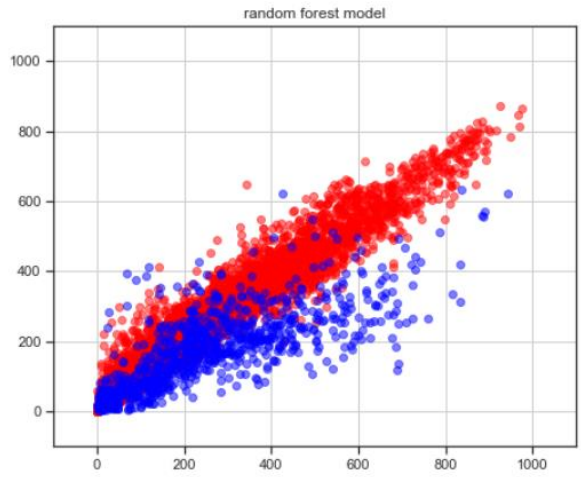
In [66]:
```
regressor.fit(train_data, train_labels)
y_pred = regressor.predict(test_data)
```

In [67]:
```
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)
```

```
MAE : 94.48785609748148
RMSE : 140.4009537401872
```

Этот результат лучше, в среднем мы ошибаемся на 94 велосипеда.





```
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)
```

```
93.38136635909738
```

Decision Tree or Random Forest - лучшие модели

Вывод:

Сравнивая качество моделей, можно сделать вывод, что метод опорных векторов (SVM) показал лучшие практические результаты.