Всё что надо знать о машинном обучении: заметки по итогам открытого курса OpenDataScience

Александр Пономаренко 17-мая 2017

Pandas - всё что нужно для обработки данных

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v'])))
```
Create DataFrame with a MultiIndex

Subset Observations (Rows)

`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

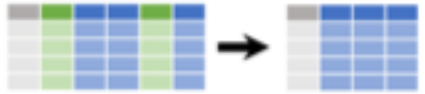
`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	`&,|,~,^,df.any(),df.all()`	Logical and, or, not, xor, any, all

Subset Variables (Columns)



`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`''^(?!Species$).*'`	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

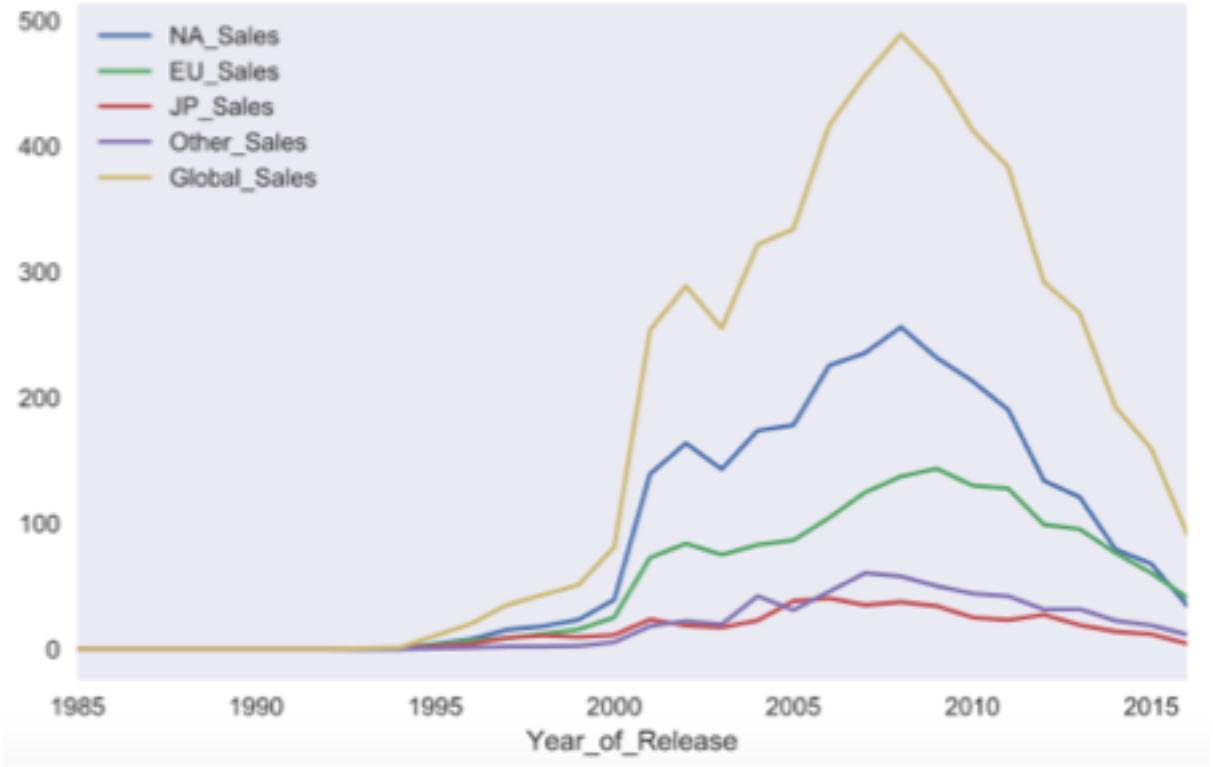
http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

https://habrahabr.ru/company/ods/blog/322626/

https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf

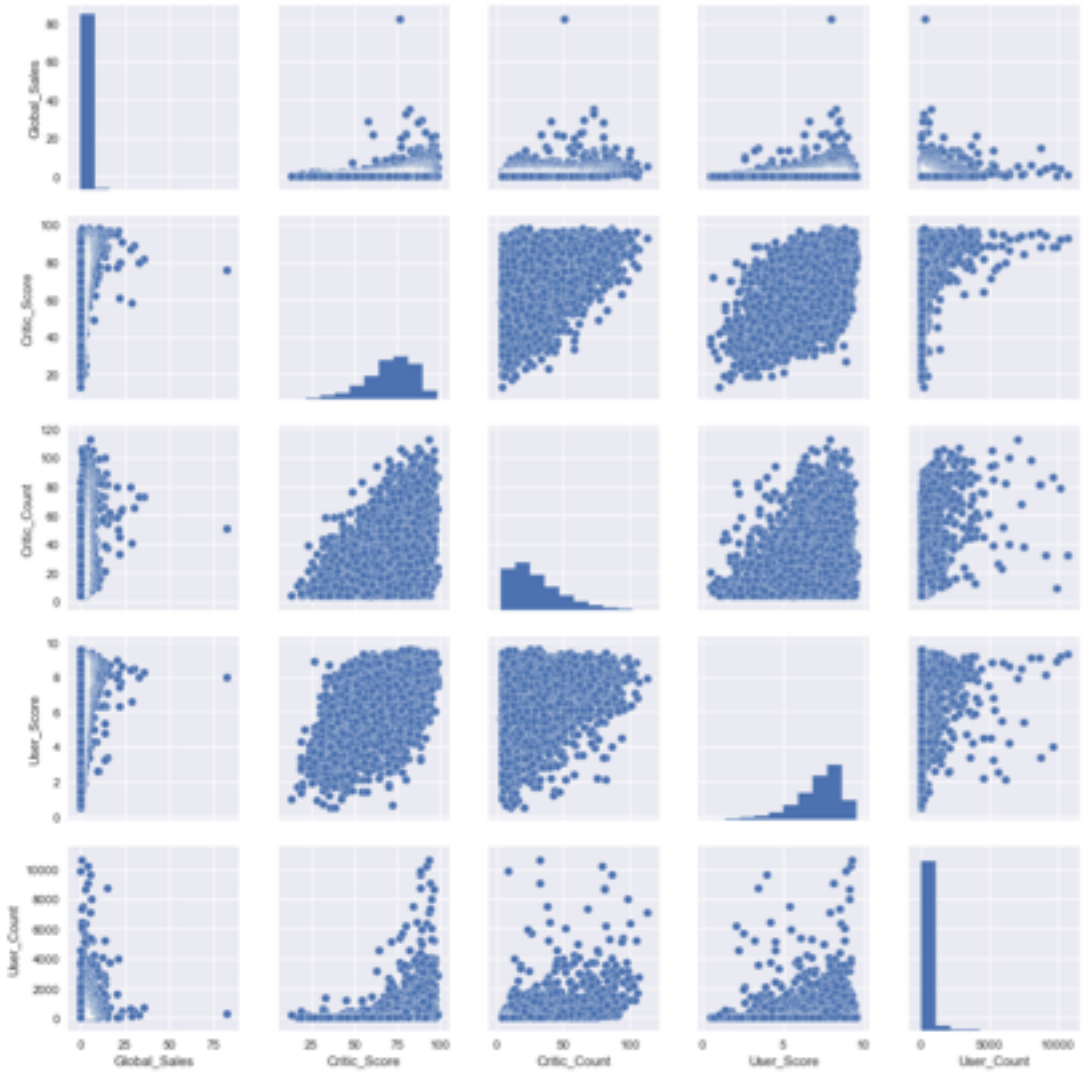
Визуализация данных

```
sales_df = df[[x for x in df.columns if 'Sales' in x] + ['Year_of_Release']]
sales_df.groupby('Year_of_Release').sum().plot()
```



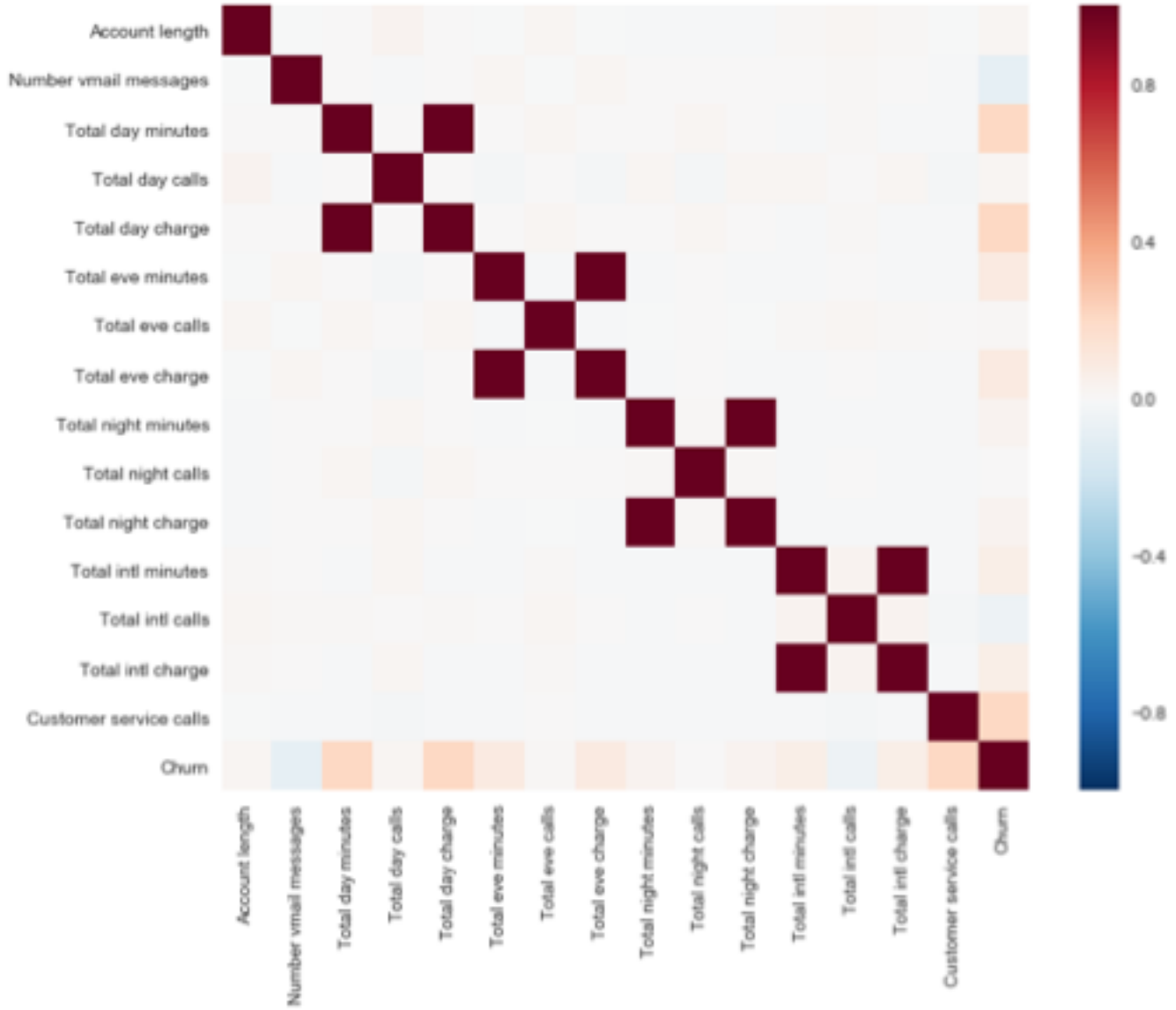
Seaborn Pairplot

```
cols = ['Global_Sales', 'Critic_Score', 'Critic_Count', 'User_Score', 'User_Count']
sns_plot = sns.pairplot(df[cols])
sns_plot.savefig('pairplot.png')
```



Seaborn heatmap

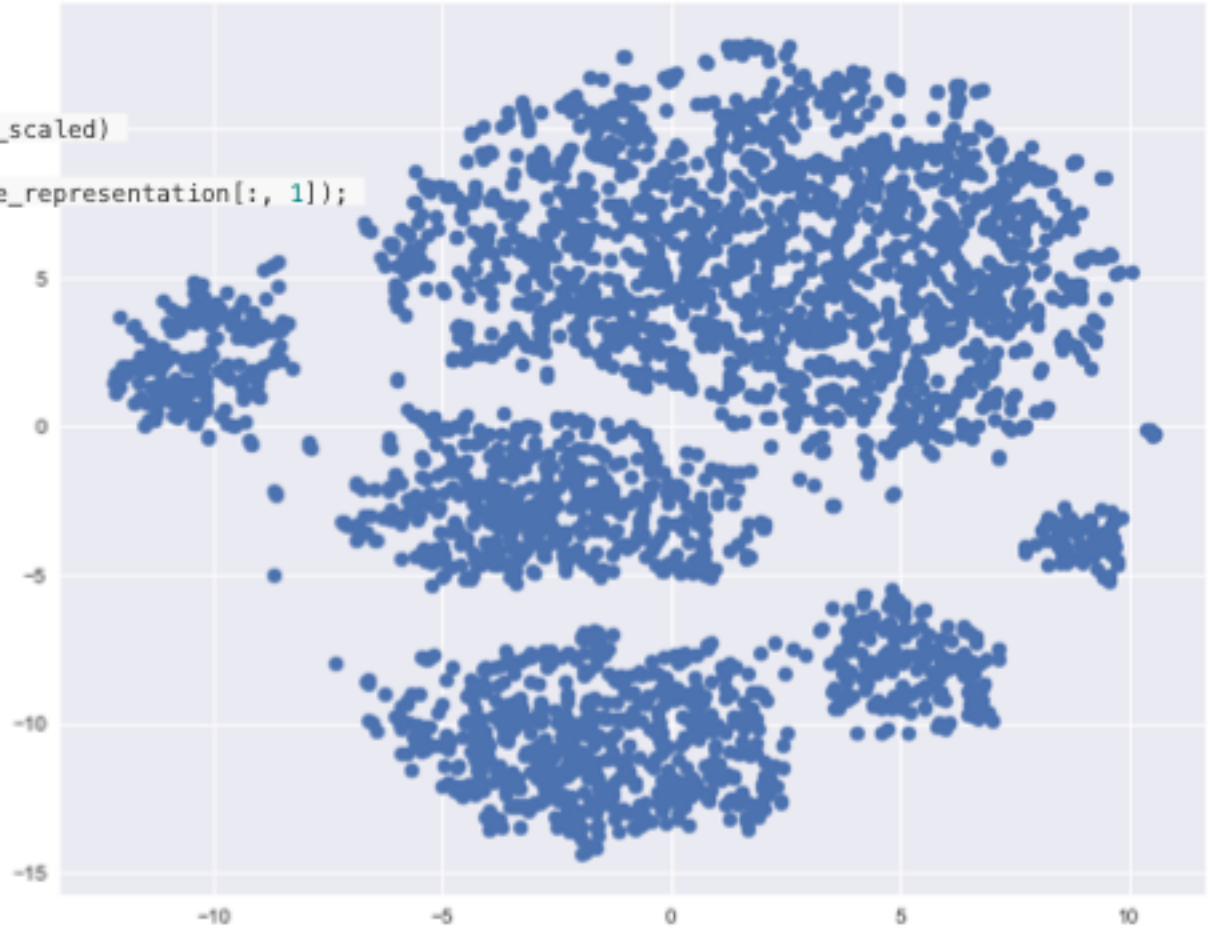
```
corr_matrix = df.drop(['State', 'International plan', 'Voice mail plan',
                       'Area code'], axis=1).corr()

sns.heatmap(corr_matrix);
```



Подглядывание в n-мерное пространство с t-SNE

```
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

# преобразуем все признаки в числовые, выкинув штаты
X = df.drop(['Churn', 'State'], axis=1)
X['International plan'] = pd.factorize(X['International plan'])[0]
X['Voice mail plan'] = pd.factorize(X['Voice mail plan'])[0]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

tsne = TSNE(random_state=17)
tsne_representation = tsne.fit_transform(X_scaled)

plt.scatter(tsne_representation[:, 0], tsne_representation[:, 1]);
```

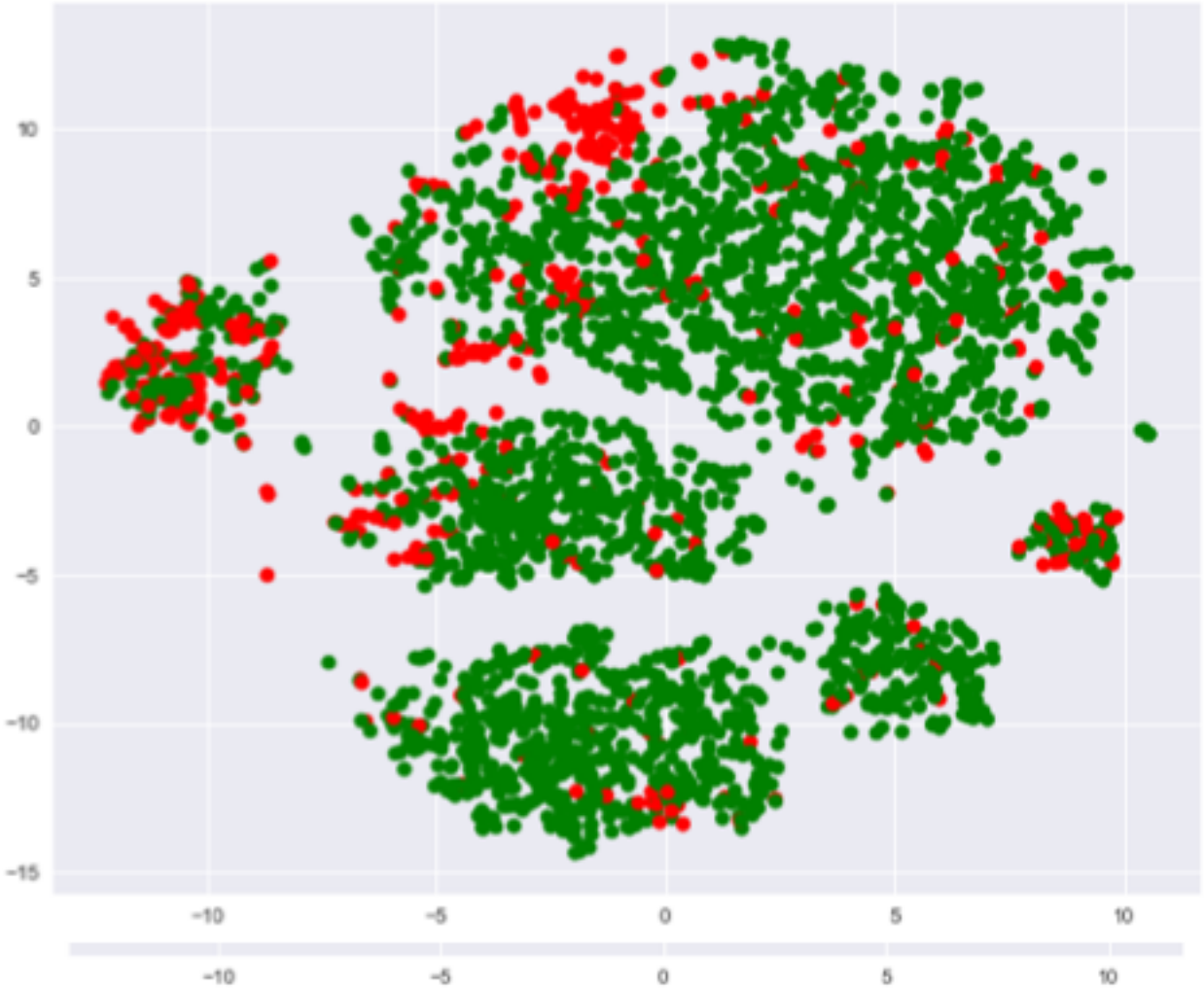
Отображение из многомерного признакового пространства на плоскость, чтоб точки, которые были далеко друг от друга, на плоскости тоже оказались удаленными, а близкие точки – также отобразились на близкие. То есть neighbor embedding – это своего рода поиск нового представления данных, при котором сохраняется соседство



Подглядывание в n-мерное пространство с t-SNE

Раскрасим полученное t-SNE представление данных по оттоку
(зеленые – лояльные, красные – ушедшие клиенты).

```
plt.scatter(tsne_representation[:, 0], tsne_representation[:, 1],
            c=df['Churn'].map({0: 'green', 1: 'red'}));
```



Работа с категориальными признаками

```
data_test.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Age	16281	73	35	461	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Workclass	15318	8	Private	11210	NaN	NaN	NaN	NaN	NaN	NaN	NaN
fnlwgt	16281	NaN	NaN	NaN	189436	105715	13492	116736	177831	238384	1.4904e+06
Education	16281	16	HS-grad	5283	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Education_Num	16281	NaN	NaN	NaN	10.0729	2.56755	1	9	10	12	16
Martial_Status	16281	7	Married-civ-spouse	7403	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Occupation	15315	14	Prof-specialty	2032	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Relationship	16281	6	Husband	6523	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Race	16281	5	White	13946	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Sex	16281	2	Male	10860	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Capital_Gain	16281	NaN	NaN	NaN	1081.91	7583.94	0	0	0	0	99999
Capital_Loss	16281	NaN	NaN	NaN	87.8993	403105	0	0	0	0	3770
Hours_per_week	16281	NaN	NaN	NaN	40.3922	12.4793	1	40	40	45	99
Country	16007	40	United-States	14662	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Target	16281	2	0	12435	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
data_test = pd.concat([data_test, pd.get_dummies(data_test['Workclass'], prefix="Workclass"),
                       pd.get_dummies(data_test['Education'], prefix="Education"),
                       pd.get_dummies(data_test['Martial_Status'], prefix="Martial_Status"),
                       pd.get_dummies(data_test['Occupation'], prefix="Occupation"),
                       pd.get_dummies(data_test['Relationship'], prefix="Relationship"),
                       pd.get_dummies(data_test['Race'], prefix="Race"),
                       pd.get_dummies(data_test['Sex'], prefix="Sex"),
                       pd.get_dummies(data_test['Country'], prefix="Country")],
                       axis=1)
```

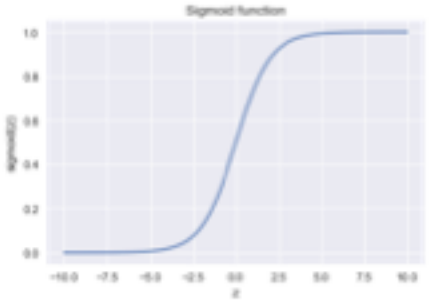
Линейные модели классификации и регрессии

$$\vec{y} = X\vec{w} + \epsilon,$$

$$a(\vec{x}) = sign(\vec{w}^T x),$$

$$p_+(x_i) = P\left(y_i = 1 \mid \vec{x_i}, \vec{w}\right) = \sigma(\vec{w}^T \vec{x_i}).$$

$$\sigma(z) = \frac{1}{1 + \exp^{-z}}$$



Плюсы:

- Хорошо изучены
- Очень быстрые, могут работать на очень больших выборках
- Практически вне конкуренции, когда признаков очень много (от сотен тысяч и более), и они разреженные (хотя есть еще факторизационные машины)
- Коэффициенты перед признаками могут интерпретироваться (при условии что признаки масштабированы) – в линейной регрессии как частные производные зависимой переменной от признаков, в логистической – как изменение шансов на отнесение к одному из классов в $\exp\beta_i$ раз при изменении признака x_i на 1 ед., подробнее тут (**upd.**)
- Логистическая регрессия выдает вероятности отнесения к разным классам (это очень ценится, например, в кредитном скоринге)
- Модель может строить и нелинейную границу, если на вход подать полиномиальные признаки

Минусы:

- Плохо работают в задачах, в которых зависимость ответов от признаков сложная, нелинейная
- На практике предположения теоремы Маркова-Гаусса почти никогда не выполняются, поэтому чаще линейные методы работают хуже, чем, например, SVM и ансамбли (по качеству решения задачи классификации/регрессии)

Обучение на гигабайтах с Vowpal Wabbit

```
'1 |text from lerxst wam umd edu where thing subject what car this nntp posting host rac3 wam umd edu organization
university maryland college park lines was wondering anyone out there could enlighten this car saw the other day was
door sports car looked from the late 60s early 70s was called bricklin the doors were really small addition the front
bumper was separate from the rest the body this all know anyone can tellme model name engine specs years production
where this car made history whatever info you have this funky looking car please mail thanks brought you your
neighborhood lerxst\n'
```

```
!vw -d ../../data/news_data/20news_train.vw \
  --loss_function hinge -f ../../data/news_data/20news_model.vw
```

```
final_regressor = ../../data/news_data/20news_model.vw
Num weight bits = 18
learning rate = 0.5
initial_t = 0
power_t = 0.5
using no cache
Reading datafile = ../../data/news_data/20news_train.vw
num sources = 1
average  since         example        example  current  current  current
loss     last          counter         weight    label  predict features
1.000000 1.000000            1            1.0  -1.0000   0.0000      157
0.911276 0.822551            2            2.0  -1.0000  -0.1774      159
0.605793 0.300311            4            4.0  -1.0000  -0.3994       92
0.419594 0.233394            8            8.0  -1.0000  -0.8167      129
0.313998 0.208402           16           16.0  -1.0000  -0.6509      108
0.196014 0.078029           32           32.0  -1.0000  -1.0000      115
0.183158 0.170302           64           64.0  -1.0000  -0.7072      114
0.261046 0.338935          128          128.0   1.0000  -0.7900      110
0.262910 0.264774          256          256.0  -1.0000  -0.6425       44
0.216663 0.170415          512          512.0  -1.0000  -1.0000      160
0.176710 0.136757         1024         1024.0  -1.0000  -1.0000      194
0.134541 0.092371         2048         2048.0  -1.0000  -1.0000      438
0.104403 0.074266         4096         4096.0  -1.0000  -1.0000      644
0.081329 0.058255         8192         8192.0  -1.0000  -1.0000      174

finished run
number of examples per pass = 8485
passes used = 1
weighted example sum = 8485.000000
weighted label sum = -7555.000000
average loss = 0.079837
best constant = -1.000000
best constant's loss = 0.109605
total feature number = 2048932
```

Обучение на гигабайтах с Vowpal Wabbit

```
!vw -i ../../data/news_data/20news_model.vw -t -d ../../data/news_data/20news_test.vw \
-p ../../data/news_data/20news_test_predictions.txt
```

```
only testing
predictions = ../../data/news_data/20news_test_predictions.txt
Num weight bits = 18
learning rate = 0.5
initial_t = 0
power_t = 0.5
using no cache
Reading datafile = ../../data/news_data/20news_test.vw
num sources = 1
average  since         example        example  current  current  current
loss     last          counter         weight    label  predict features
0.000000 0.000000            1            1.0  unknown   1.0000      349
0.000000 0.000000            2            2.0  unknown  -1.0000       50
0.000000 0.000000            4            4.0  unknown  -1.0000      251
0.000000 0.000000            8            8.0  unknown  -1.0000      237
0.000000 0.000000           16           16.0  unknown  -0.8978      106
0.000000 0.000000           32           32.0  unknown  -1.0000      964
0.000000 0.000000           64           64.0  unknown  -1.0000      261
0.000000 0.000000          128          128.0  unknown   0.4621       82
0.000000 0.000000          256          256.0  unknown  -1.0000      186
0.000000 0.000000          512          512.0  unknown  -1.0000      162
0.000000 0.000000         1024         1024.0  unknown  -1.0000      283
0.000000 0.000000         2048         2048.0  unknown  -1.0000      104

finished run
number of examples per pass = 2829
passes used = 1
weighted example sum = 2829.000000
weighted label sum = 0.000000
average loss = 0.000000
total feature number = 642215
```