Data Science Fundamentals 5

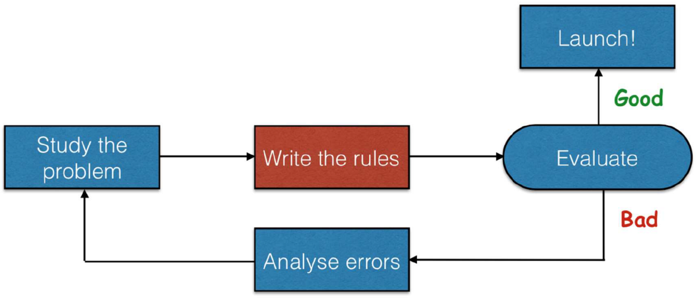
Basic introduction on how to perform typical machine learning tasks with Python.

Prepared by Mykhailo Vladymyrov & Aris Marcolongo, Science IT Support, University Of Bern, 2020

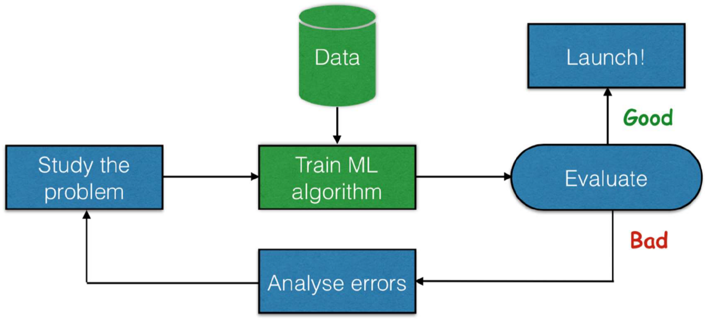
Part 1.

What is Machine learning?

Unlike classical algorithms, created by human to analyze some data:

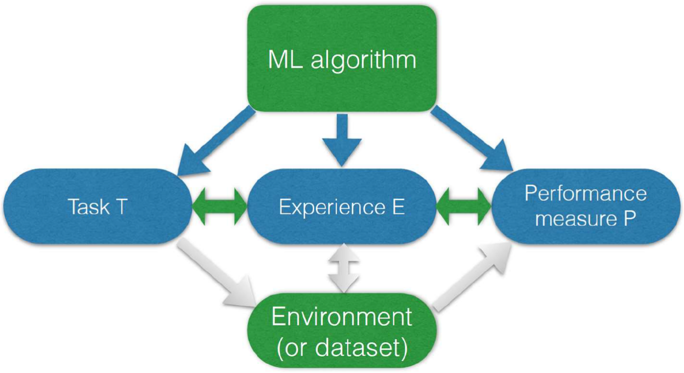


in machine learning the data itself is used for to define the algorithm:



ML:

- improves performance according to measure **P**
- on a task **T**
- with experience **E**



The boundary is a bit fuzzy. In fact when we create algorithms, the problem in hand, namely the data related to the problem, drives us to chose one or another algorithm. And we then tune it, to perform well on a task in hand. ML formalized this procedure, allowing us to automate (part) of thise process.

In this 2-day course you will get acquainted with the basics of ML, where the approach to handling the data (the algorithm) is defined, or as we say "learned" from data in hand.

Classification vs regression.

The two main tasks handled by (supervised) ML is regression and classification. In regression we aim at modeling the relationship between the system's response (dependent variable) and one or more explanatory variables (independent variables).

Examples of regression would be predicting the temperature for each day of the year, or expenses of the household as a function of the number of children and adults.

In classification the aim is to identify what class does a data-point belong to. For example the spieces or the iris plant based on the size of it's petals, or whether an email is spam or not based on it's content.

Performance measures

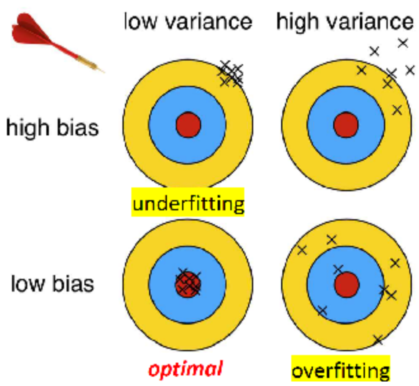
1. Regression:
2. Mean Square Error (MSE): $mse = \frac{1}{n}\sum_i (y_i - \hat{y}(\bar{x}_i))^2$
3. Mean Absolute Error (MAE): $mae = \frac{1}{n}\sum_i |y_i - \hat{y}(\bar{x}_i)|$
4. Median Absolute Deviation (MAD): $mad = median(|y_i - \hat{y}(\bar{x}_i)|)$
5. Fraction of the explained variance: $R^2 = 1 - \frac{\sum_i (y_i - \hat{y}(\bar{x}_i))^2}{\sum_i (y_i - \bar{y}_i)^2}$, where $\bar{y} = \frac{1}{n}\sum_i y_i$
6. Classification:
7. Confusion matrix

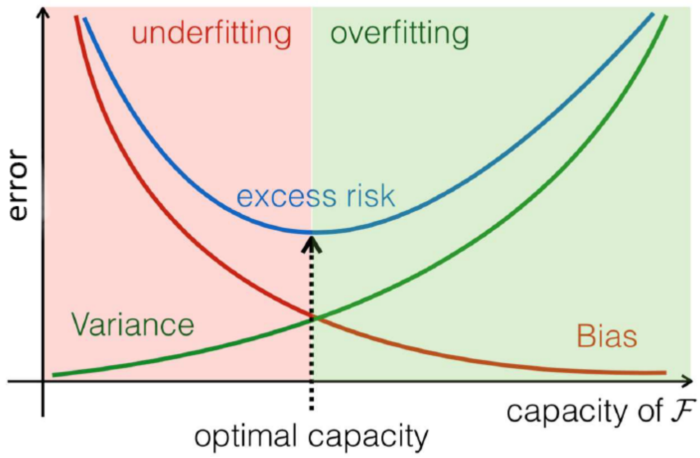
		Prediction	
		Predicted 1	Predicted 0
Ground truth	Class 1	**TP**	**FN**
	Class 0	**FP**	**TN**

- Accuracy $= \frac{TP + TN}{TP + FP + FN + TN}$
- Precision $= \frac{TP}{TP + FP}$
- Recall $= \frac{TP}{TP + FN}$
- F1 $= 2\frac{Precision \cdot Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$
- Threat score(TS), or Intersection over Union (IoU): IoU=$\frac{TP}{TP + FN + FP}$

During model optimization the used measure in most cases must be differentiable. To this end usually some measure of similarities of diestributions are employed (e.g. cross-entropy).

Actual aim: Generalization





To measure model performace in an unbiassed way, we need to use different data than the data that the model was trained on. For this we use the 'train-test' split: e.g. 20% of all available dataset is reserved for model performance test, and the remaining 80% is used for actual model training.

Load libraries

In [0]:
```
from sklearn import linear_model

from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn import metrics

from matplotlib import pyplot as plt
import numpy as np
import os
from imageio import imread
import pandas as pd
from time import time as timer

import tensorflow as tf

%matplotlib inline
from matplotlib import animation
from IPython.display import HTML
```

In [0]:
```
if not os.path.exists('data'):
    path = os.path.abspath('.')+'/colab_material.tgz'
    tf.keras.utils.get_file(path, 'https://github.com/ne
worldemancer/DSF5/raw/master/colab_material.tgz')
    !tar -xvzf colab_material.tgz > /dev/null 2>&1
```

Datasets

In this course we will use several synthetic and real-world datasets to ilustrate the behavior of the models and excercise our skills.

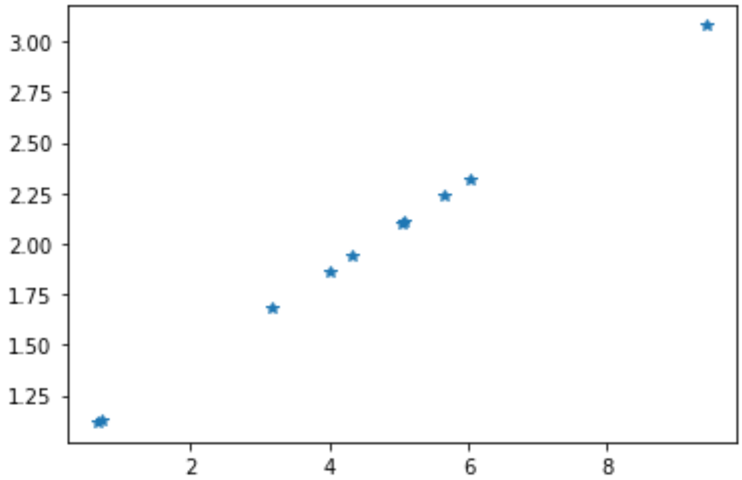
1. Synthetic linear

In [0]:
```
def get_linear(n_d=1, n_points=10, w=None, b=None, sigma
=5):
  x = np.random.uniform(0, 10, size=(n_points, n_d))

  w = w or np.random.uniform(0.1, 10, n_d)
  b = b or np.random.uniform(-10, 10)
  y = np.dot(x, w) + b + np.random.normal(0, sigma, size
=n_points)

  print('true w =', w, ';  b =', b)

  return x, y
```

In [0]:
```
x, y = get_linear(n_d=1, sigma=0)
plt.plot(x[:, 0], y, '*')
```

```
true w = [0.22340972] ;  b = 0.9711035072823542
```

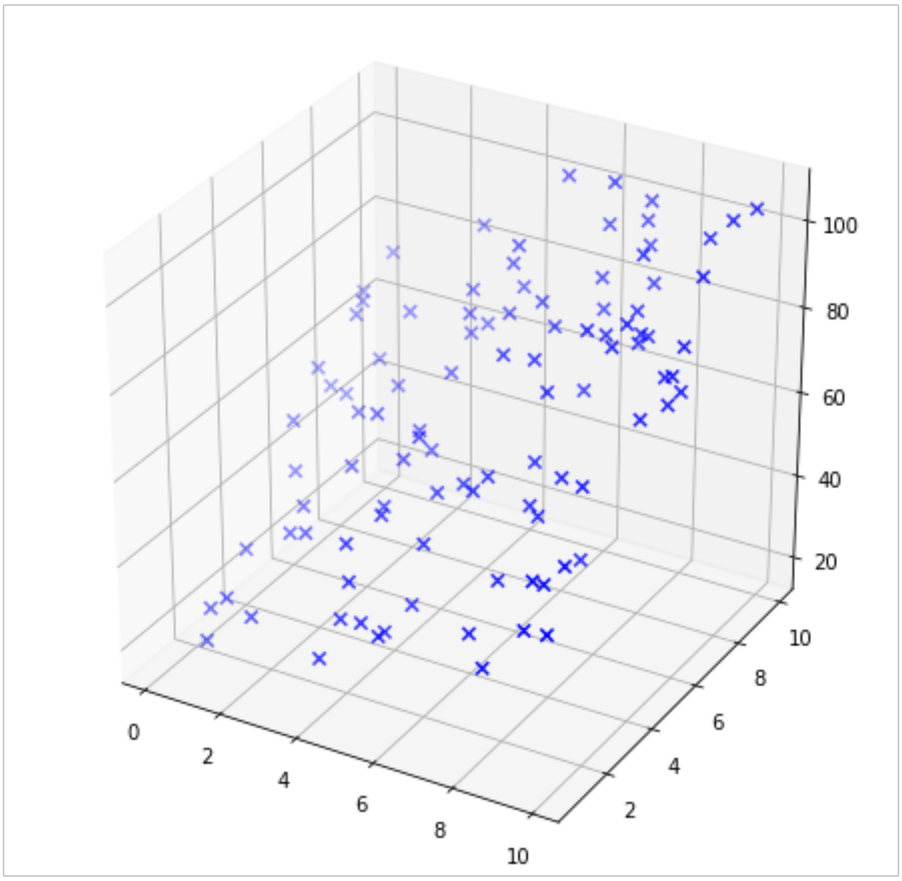
Out[0]:
```
[<matplotlib.lines.Line2D at 0x7fb2c6f56940>]
```



In [0]:
```
n_d = 2
x, y = get_linear(n_d=n_d, n_points=100)

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x[:,0], x[:,1], y, marker='x', color='b',s=4
0)
```

```
true w = [3.53409832 6.67393245] ;  b = 7.18048579361976
3
```

Out[0]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7fb2c6
a40ba8>
```



2. House prices

Subset of the the hous pricess kaggle dataset: [https://www.kaggle.com/c/house-prices-advanced-regression-techniques (https://www.kaggle.com/c/house-prices-advanced-regression-techniques)](https://www.kaggle.com/c/house-prices-advanced-regression-techniques)

In [0]:
```
def house_prices_dataset(return_df=False, price_max=4000
00, area_max=40000):
  path = 'data/train.csv'

  df = pd.read_csv(path, na_values="NaN", keep_default_n
a=False)

  useful_fields = ['LotArea',
                  'Utilities', 'OverallQual', 'OverallCo
nd',
                  'YearBuilt', 'YearRemodAdd', 'ExterQua
l', 'ExterCond',
                  'HeatingQC', 'CentralAir', 'Electrical
',
                  '1stFlrSF', '2ndFlrSF','GrLivArea',
                  'FullBath', 'HalfBath',
                  'BedroomAbvGr', 'KitchenAbvGr', 'Kitch
enQual', 'TotRmsAbvGrd',
                  'Functional','PoolArea',
                  'YrSold', 'MoSold'
                  ]
  target_field = 'SalePrice'

  cleanup_nums = {"Street":      {"Grvl": 0, "Pave": 1},
                  "LotFrontage": {"NA":0},
                  "Alley":       {"NA":0, "Grvl": 1, "Pa
ve": 2},
                  "LotShape":    {"IR3":0, "IR2": 1, "IR
1": 2, "Reg":3},
                  "Utilities":   {"ELO":0, "NoSeWa": 1,
"NoSewr": 2, "AllPub": 3},
                  "LandSlope":   {"Sev":0, "Mod": 1, "Gt
l": 3},
                  "ExterQual":   {"Po":0, "Fa": 1, "TA":
2, "Gd": 3, "Ex":4},
                  "ExterCond":   {"Po":0, "Fa": 1, "TA":
2, "Gd": 3, "Ex":4},
                  "BsmtQual":    {"NA":0, "Po":1, "Fa":
2, "TA": 3, "Gd": 4, "Ex":5},
                  "BsmtCond":    {"NA":0, "Po":1, "Fa":
2, "TA": 3, "Gd": 4, "Ex":5},
                  "BsmtExposure":{"NA":0, "No":1, "Mn":
2, "Av": 3, "Gd": 4},
                  "BsmtFinType1":{"NA":0, "Unf":1, "Lw
Q": 2, "Rec": 3, "BLQ": 4, "ALQ":5, "GLQ":6},
                  "BsmtFinType2":{"NA":0, "Unf":1, "Lw
Q": 2, "Rec": 3, "BLQ": 4, "ALQ":5, "GLQ":6},
                  "HeatingQC":   {"Po":0, "Fa": 1, "TA":
2, "Gd": 3, "Ex":4},
                  "CentralAir":  {"N":0, "Y": 1},
                  "Electrical":  {"NA":0, "Mix":1, "Fuse
P":2, "FuseF": 3, "FuseA": 4, "SBrkr": 5},
                  "KitchenQual": {"Po":0, "Fa": 1, "TA":
2, "Gd": 3, "Ex":4},
                  "Functional":  {"Sal":0, "Sev":1, "Maj
```

In [0]:
```
x, y, df = house_prices_dataset(return_df=True)
print(x.shape, y.shape)
df.head()
```

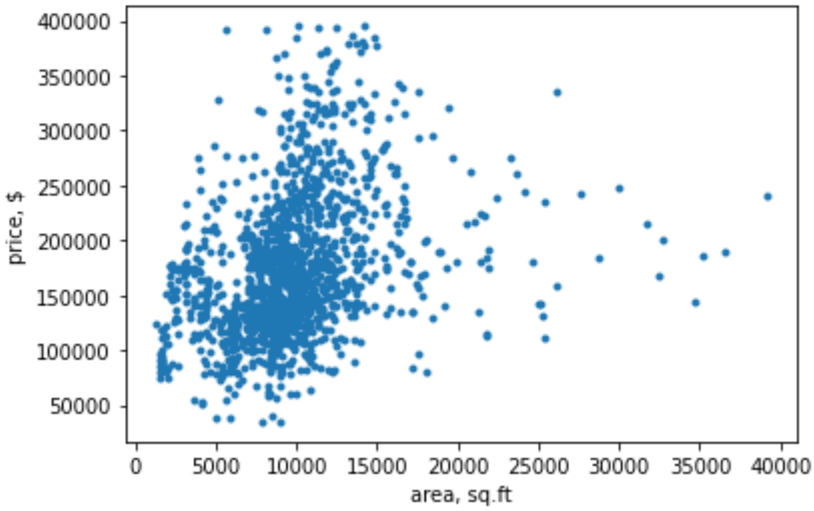
```
(1420, 24) (1420,)
```

Out[0]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotSha
0	1	60	RL	65	8450	Pave	NA	Reg
1	2	20	RL	80	9600	Pave	NA	Reg
2	3	60	RL	68	11250	Pave	NA	IR1
3	4	70	RL	60	9550	Pave	NA	IR1
4	5	60	RL	84	14260	Pave	NA	IR1

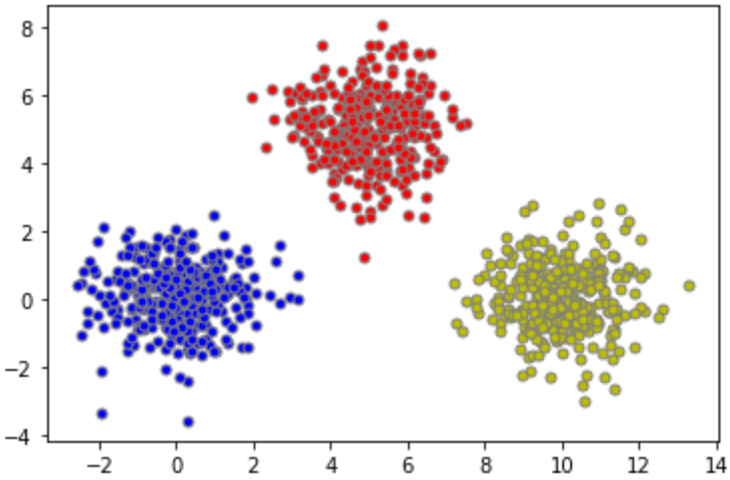
5 rows × 81 columns

In [0]:
```
plt.plot(x[:, 0], y, '.')
plt.xlabel('area, sq.ft')
plt.ylabel('price, $');
```



3. Blobs

In [0]:
```
x, y = make_blobs(n_samples=1000, centers=[[0,0], [5,5],
[10, 0]])
colors = "bry"
for i, color in enumerate(colors):
    idx = y == i
    plt.scatter(x[idx, 0], x[idx, 1], c=color, edgecolor
='gray', s=25)
```



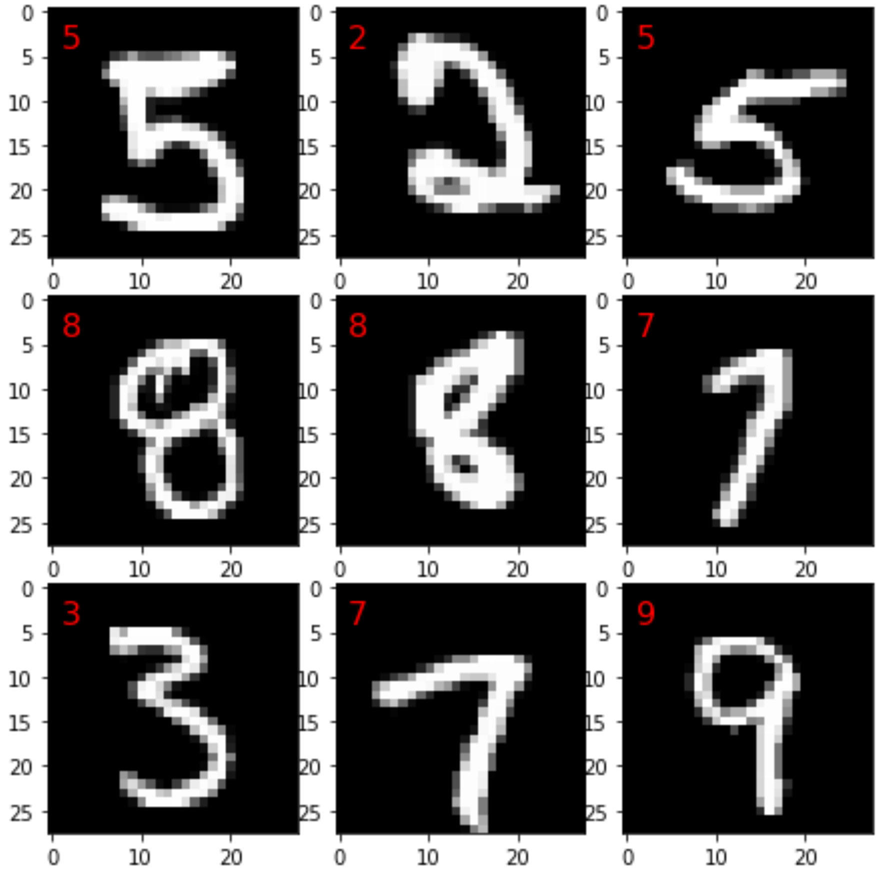
4. MNIST

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. (taken from http://yann.lecun.com/exdb/mnist/ (http://yann.lecun.com/exdb/mnist/)) . Each example is a 28x28 grayscale image and the data-set can be readily downloaded from Tensorflow.

In [0]:
```
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels)
= mnist.load_data()
```

Let's chech few samples:

In [0]:
```
n = 3
fig, ax = plt.subplots(n, n, figsize=(2*n, 2*n))
ax = [ax_xy for ax_y in ax for ax_xy in ax_y]
for axi, im_idx in zip(ax, np.random.choice(len(train_images), n**2)):
  im = train_images[im_idx]
  im_class = train_labels[im_idx]
  axi.imshow(im, cmap='gray')
  axi.text(1, 4, f'{im_class}', color='r', size=16)
plt.tight_layout(0,0,0)
```



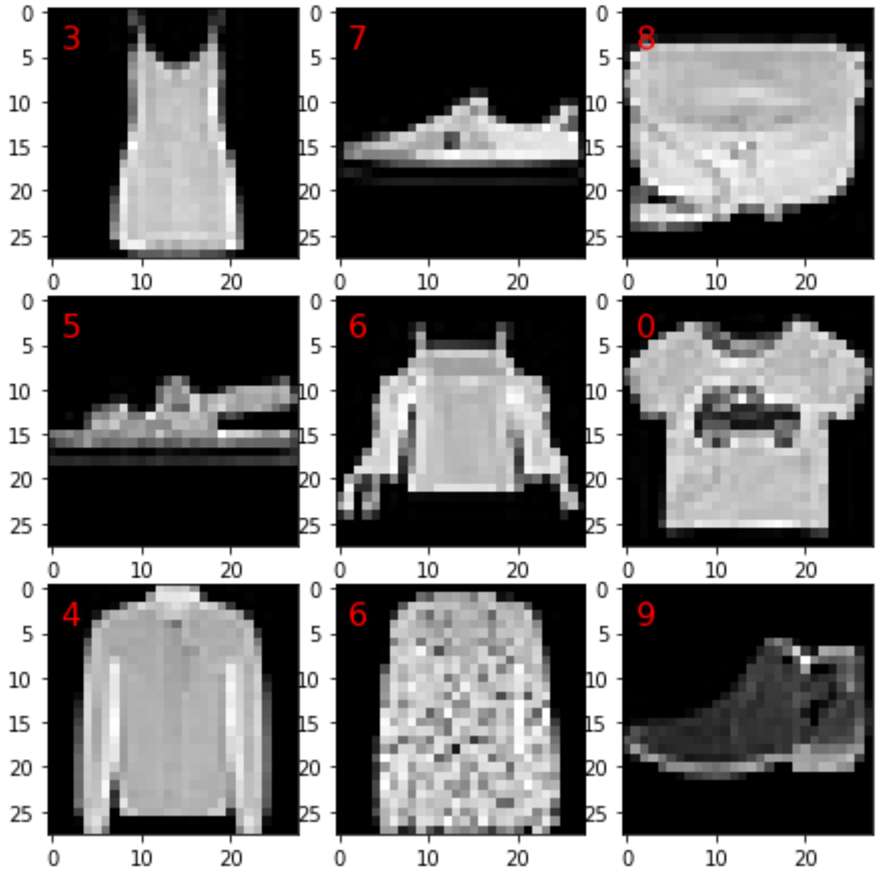
5. Fashion MNIST

`Fashion-MNIST` is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. (from https://github.com/zalandoresearch/fashion-mnist (https://github.com/zalandoresearch/fashion-mnist))

In [0]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels)
= fashion_mnist.load_data()
```

Let's chech few samples:

In [0]:
```
n = 3
fig, ax = plt.subplots(n, n, figsize=(2*n, 2*n))
ax = [ax_xy for ax_y in ax for ax_xy in ax_y]
for axi, im_idx in zip(ax, np.random.choice(len(train_images), n**2)):
  im = train_images[im_idx]
  im_class = train_labels[im_idx]
  axi.imshow(im, cmap='gray')
  axi.text(1, 4, f'{im_class}', color='r', size=16)
plt.tight_layout(0,0,0)
```



Each training and test example is assigned to one of the following labels:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

`scikit-learn` interface

In this course we will primarily use the `scikit-learn` module. You can find extensive documentation with exmples in the user guide (https://scikit-learn.org/stable/user_guide.html)

The module contains A LOT of diffenet mashine learning methods, and here we will cover only few of them. What is great about `scikit-learn` is that it has a uniform and consistent interface.

All the different ML approaches are implemented as classes with a set of same main methods:

1. `fitter = ...`: Create object.
2. `fitter.fit(x, y[, sample_weight])`: Fit model.
3. `y_pred = fitter.predict(X)`: Predict using the linear model.
4. `s = score(x, y[, sample_weight])`: Return an appropriate measure of model performace.

This allows one to easily replace one approach with another, and find the best one for the probkem at hand, by simply using another regression/classification object, while the rest of the code can remain the same.

1.Linear models

In many cases the the scalar value of interest - dependent variable - is (or can be aproximated as) linear combination of the independent variables.

In linear regression the estimator is searched in the form:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$$

The parameters $w = (w_1, \ldots, w_p)$ and w_0 are designated as `coef_` and `intercept_` in `sklearn`.

Reference: https://scikit-learn.org/stable/modules/linear_model.html (https://scikit-learn.org/stable/modules/linear_model.html)

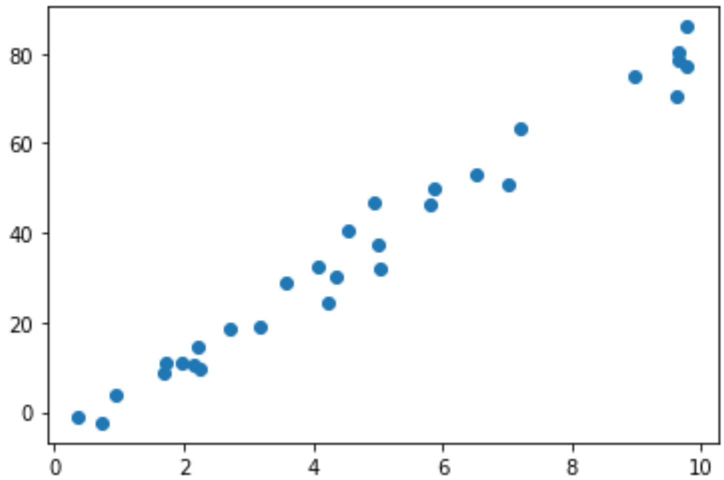
1. Linear regression

LinearRegression fits a linear model with coefficients $w = (w_1, \ldots, w_p)$ and w_0 to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_{w} ||Xw - y||_2^2$$

In [0]:
```
x, y = get_linear(n_d=1, sigma=3, n_points=30)  # p==1, 1D input
plt.scatter(x, y);
```

```
true w = [8.45446474] ;  b = -3.1145687950426026
```



In [0]:
```
reg = linear_model.LinearRegression()
reg.fit(x, y)
```

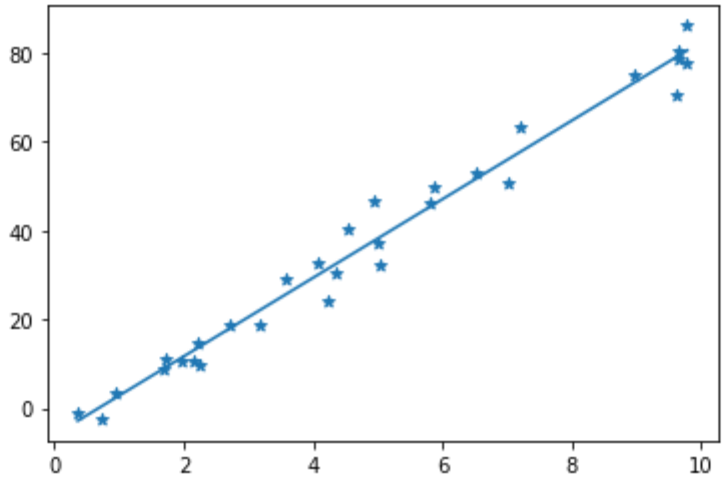
Out[0]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs
=None, normalize=False)
```

In [0]:
```
w, w0 = reg.coef_, reg.intercept_
print(w, w0)
```

```
[8.83547846] -5.992466828806293
```

In [0]:
```
plt.scatter(x, y, marker='*')
x_f = np.linspace(x.min(), x.max(), 10)
y_f = w0 + w[0] * x_f
plt.plot(x_f, y_f)
```

Out[0]:
```
[<matplotlib.lines.Line2D at 0x7fb2c253cd30>]
```



In [0]:
```
# mse
np.std(y - reg.predict(x))  # or use metrics.mean_square
d_error(..., squared=False)
```

Out[0]:
```
3.957683648148916
```

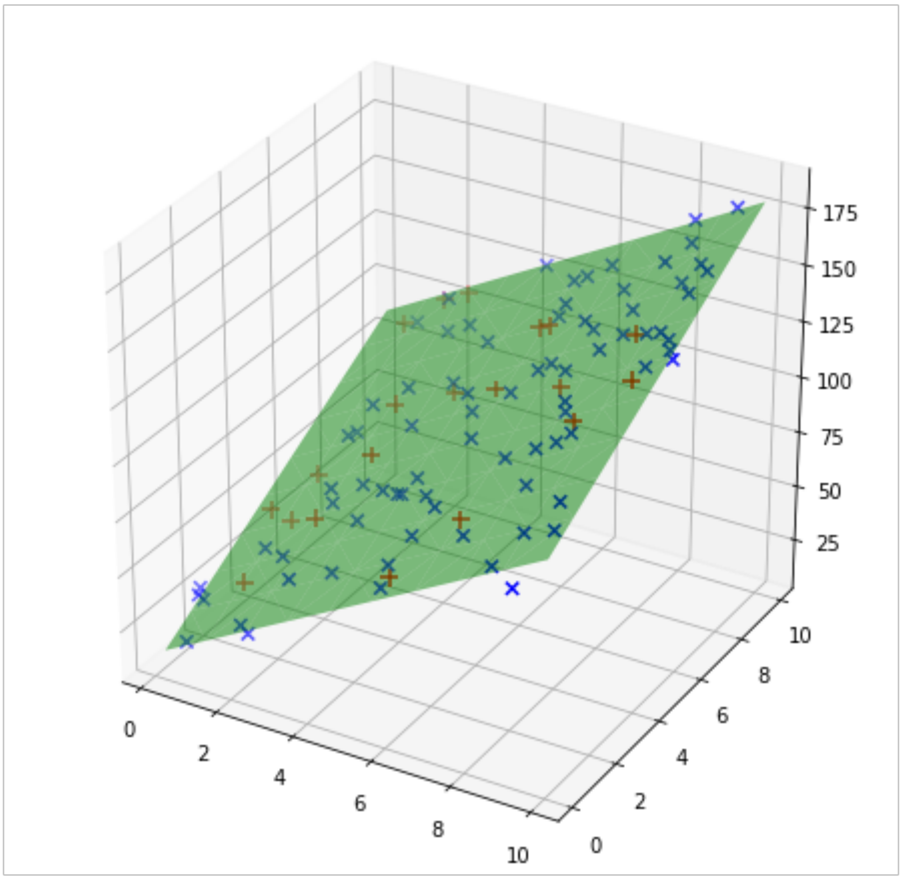
In [0]:
```
# R2
reg.score(x, y)
```

Out[0]:
```
0.9774962341905981
```

Let's try 2D input. Aditionally here we will split the whole dataset into training and test subsets using the `train_test_split` function:

In [0]:
```
n_d = 2
x, y = get_linear(n_d=n_d, n_points=100, sigma=5)

# train test split
x_train, x_test, y_train, y_test = train_test_split(x,
y, test_size=0.2)

reg = linear_model.LinearRegression()
reg.fit(x_train, y_train)


fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_train[:,0], x_train[:,1], y_train, marker='
x', color='b',s=40)
ax.scatter(x_test[:,0], x_test[:,1], y_test, marker='+',
color='r',s=80)

xx0 = np.linspace(x[:,0].min(), x[:,0].max(), 10)
xx1 = np.linspace(x[:,1].min(), x[:,1].max(), 10)
xx0, xx1 = [a.flatten() for a in np.meshgrid(xx0, xx1)]
xx = np.stack((xx0, xx1), axis=-1)
yy = reg.predict(xx)
ax.plot_trisurf(xx0, xx1, yy, alpha=0.5, color='g');
```

```
true w = [9.67781514 7.94527226] ;  b = 7.37399277228391
6
```



In [0]:
```
# mse
print('train mse =', np.std(y_train - reg.predict(x_trai
n)))
print('test mse =', np.std(y_test - reg.predict(x_tes
t)))
```

```
train mse = 4.825382240735499
test mse = 5.946155392271906
```

In [0]:
```
# R2
print('train R2 =', reg.score(x_train, y_train))
print('test R2 =', reg.score(x_test, y_test))
```

```
train R2 = 0.9839542964836011
test R2 = 0.9619708379511813
```

EXERCISE 1.

Use linear regression to fit house prices dataset.

In [0]:
```
x, y = house_prices_dataset()

# 1. make train/test split

# 2. fit the model

# 3. evaluate MSE, MAE, and R2 on train and test dataset
s

# 4. plot y vs predicted y for test and train parts
```

2. Logistic regression

Logistic regression, despite its name, is a linear model for classification rather than regression. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

In logistic regression the probability p of a point belonging to a class is modeled as:

$$\frac{p}{1-p} = e^{w_0 + w_1 x_1 + \ldots + w_p x_p}$$

The binary class ℓ_2 penalized logistic regression minimizes the following cost function:

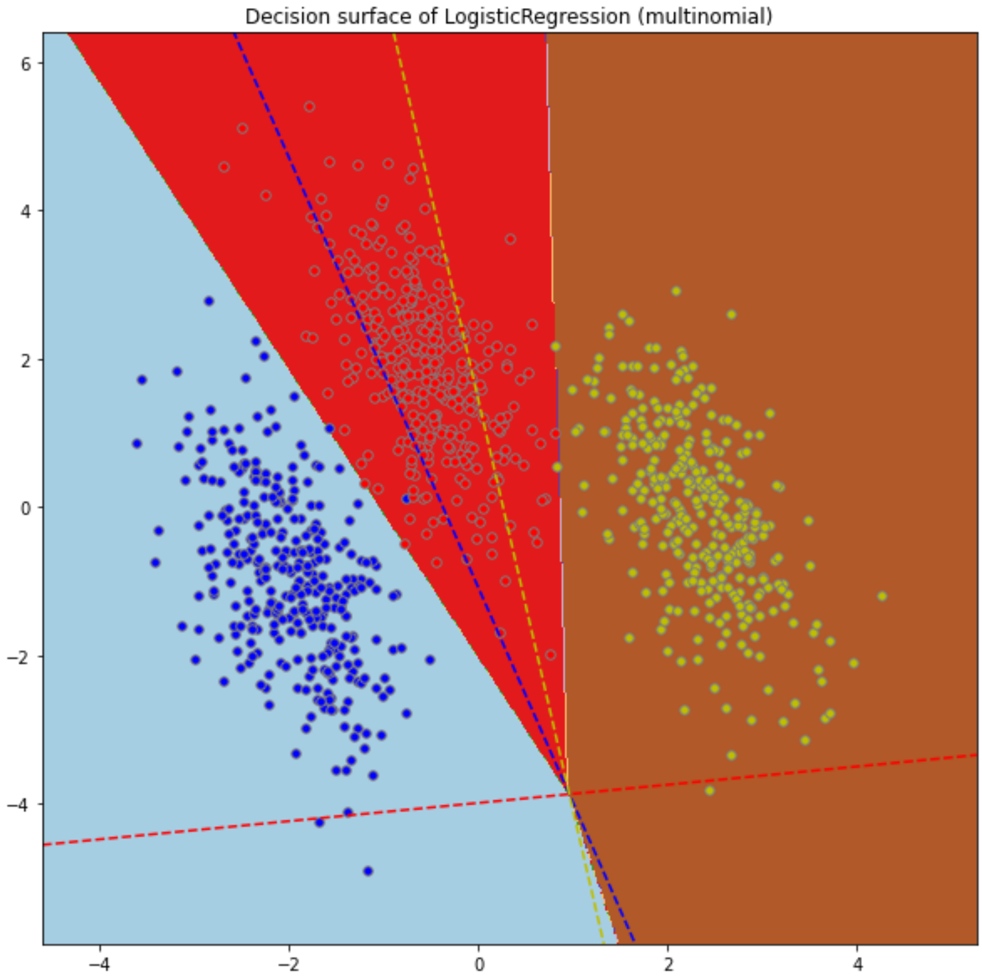
$$\min_{w,c} \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1) + \lambda \frac{1}{2} w^T w$$

.

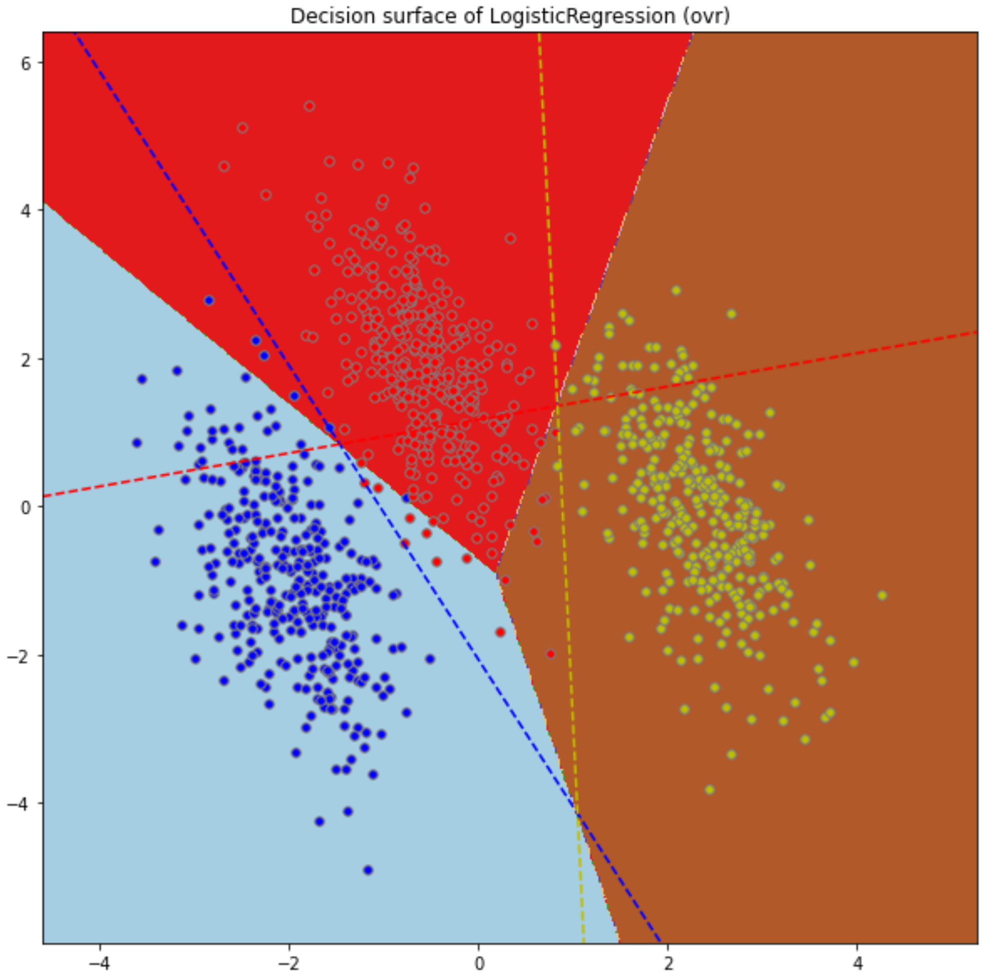
In [0]:

```
# make 3-class dataset for classification
centers = [[-5, 0], [0, 1.5], [5, -1]]
x, y = make_blobs(n_samples=1000, centers=centers, rando
m_state=40)
transformation = [[0.4, 0.2], [-0.4, 1.2]]
x = np.dot(x, transformation)

for multi_class in ('multinomial', 'ovr'):
    clf = linear_model.LogisticRegression(solver='sag',
max_iter=100,
                             multi_class=multi_class)
    clf.fit(x, y)

    # print the training scores
    print("training accuracy : %.3f (%s)" % (clf.score
(x, y), multi_class))

    # create a mesh to plot in
    h = .02  # step size in the mesh
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # Plot the decision boundary. For that, we will assi
gn a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    # Put the result into a color plot
    z = z.reshape(xx.shape)
    plt.figure(figsize=(10,10))
    plt.contourf(xx, yy, z, cmap=plt.cm.Paired)
    plt.title("Decision surface of LogisticRegression (%
s)" % multi_class)
    plt.axis('tight')

    # Plot also the training points
    colors = "bry"
    for i, color in zip(clf.classes_, colors):
        idx = np.where(y == i)
        plt.scatter(x[idx, 0], x[idx, 1], c=color, cmap=
plt.cm.Paired,
                    edgecolor='gray', s=30)

    # Plot the three one-against-all classifiers
    xmin, xmax = plt.xlim()
    ymin, ymax = plt.ylim()
    coef = clf.coef_
    intercept = clf.intercept_

    def plot_hyperplane(c, color):
        def line(x0):
            return (-(x0 * coef[c, 0]) - intercept[c]) /
coef[c, 1]
        plt.plot([xmin, xmax], [line(xmin), line(xmax)],
```

```
training accuracy : 0.995 (multinomial)
```




```
training accuracy : 0.976 (ovr)
```



EXERCISE 2.

In [0]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels)
= fashion_mnist.load_data()
```

We will reshape 2-d images to 1-d arrays for use in scikit-learn:

In [0]:
```
n_train = len(train_labels)
x_train = train_images.reshape((n_train, -1))
y_train = train_labels

n_test = len(test_labels)
x_test = test_images.reshape((n_test, -1))
y_test = test_labels
```

Now use a multinomial logistic regression classifier, and measure the accuracy:

In [0]:
```
# 1. Create classifier

# 2. fit the model

# 3. evaluate accuracy on train and test datasets
```