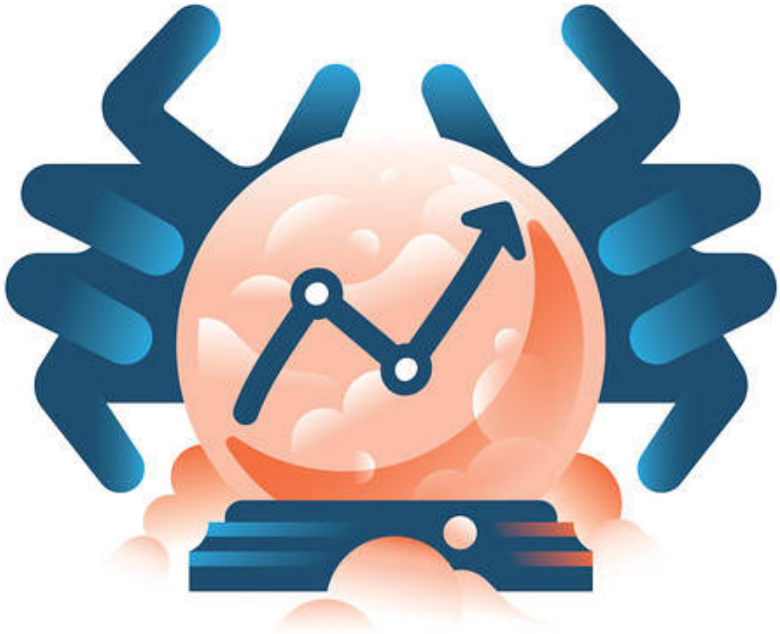
Predicting with supervised learning

What we will see today:

- Classification
- Regression
- Model selection and evaluation
- Brief Introduction to Deep Learning



I - Introduction

What is supervised learning?

- Problem statement
- Model and Hyperparameters
- K-fold cross-validation

Example with Young People Dataset

In [11]:
```
import pandas as pd
import numpy as np
from sklearn.svm import SVC

young_dataset = pd.read_csv('data/responses.csv').dropna()

likes_np = young_dataset.iloc[:,:50].to_numpy()
social_np = young_dataset.iloc[:,113].factorize()[0]

X_train = likes_np[ :int(len(likes_np)*0.8 ), : ]
y_train = social_np[ :int(len(social_np)*0.8 ), ]

X_test = likes_np[ int(len(likes_np)*0.8):, :]
y_test = social_np[ int(len(social_np)*0.8):, ]

classifier = SVC(gamma = 'auto').fit(X_train, y_train)

print(classifier.score( X_test, y_test), ' vs chance level :', 1/5.0)
```

```
0.35294117647058826  vs chance level : 0.2
```

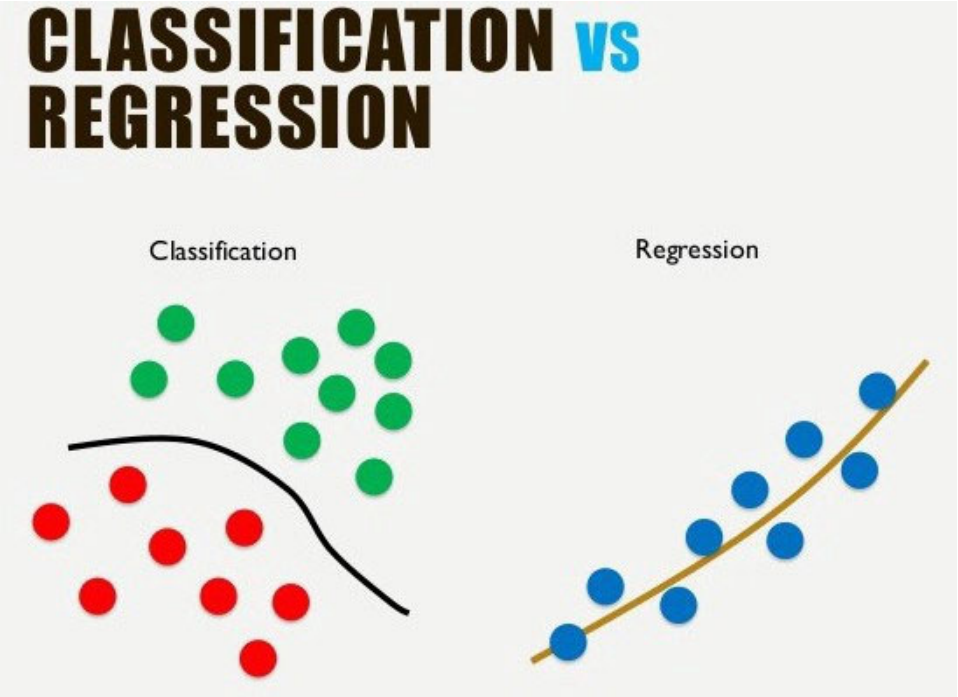
In [22]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, classifier.predict(X_test))

print(cm)
```

```
[[23  1  0  1  0]
 [10  0  1  0  0]
 [ 7  1  0  0  0]
 [14  2  0  1  0]
 [ 7  0  0  0  0]]
```

Different sub-fields of supervised learning

The point of supervised learning is to predict something:

- a quantity (regression)
- a property (classification)



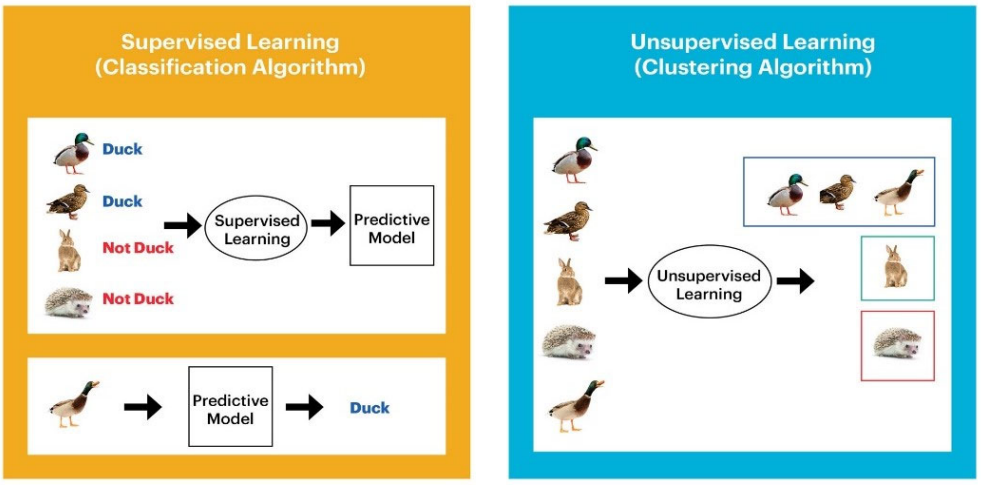
Other related problems:

- reinforcement learning (making decisions on how to act)
- recommender systems (based on sparse dataset, recommend products, movies ...)
- AGI: self-supervised (predicting the future of an agent)

Description of supervised learning

A supervised learning problem consists of:

- a dataset $D = \{x_i, y_i\}$
- x_i raw input data or compressed / feature engineered
- y_i target data: what we want to predict
- a model that links inputs to outputs
- a Loss / Objective to optimize / train our model
- an evaluation criterion



Importance for Datascience

Datascience without Supervised learning can only give intuitions: no quantitative insights.

Models can be used to make business decisions: Predict, anticipate

Data-driven decisions, Big data, the new oil, ...

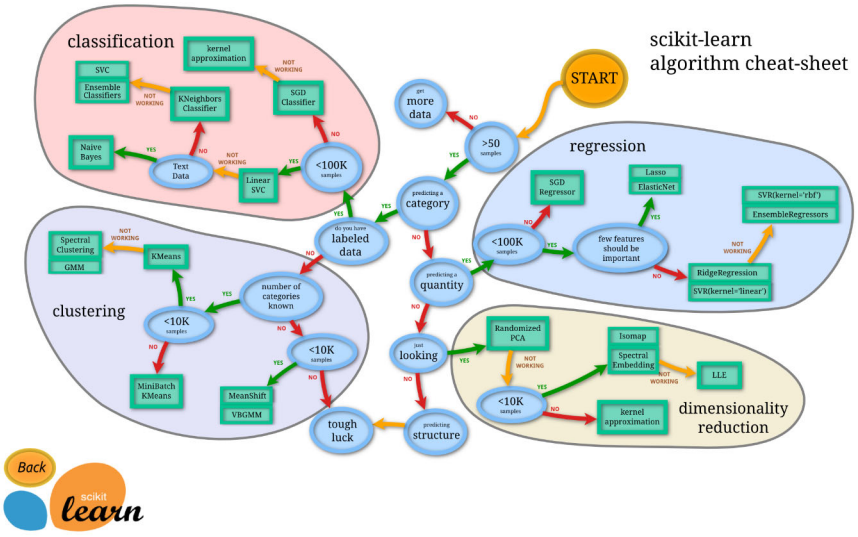
But you have to do it right...

Decisions based on a wrong model can have adverse effects.

New recent topics:

- bias in machine learning
- privacy preservation
- explainability

II - How to choose a model?



A huge amount of models, and their associated hyperparameters, are available to solve prediction problems.

We must chose a model selection and hyperparameters:

- We can use automated tools
- We can limit the search space through intuition and experience

Predicting with 100% accuracy?

1. Train your model on your dataset.
2. Compare the score of different models, pick the best.
3. Compare different model parameters, pick the best.
4. Evaluate, have 100% accuracy
5. Become rich

Why am I going to loose all my money?

Everybody will do this mistake at some point!

The problem is well known. Everybody did it (me included).

Basically, you just trained your model on the same data you are testing on. As your model optimizes to get the best result on the training set, it will get 100% accuracy when you test it.

That is why we split training and testing sets.

Be wary of very good results.

Train / test split

Now we split the dataset D into:

- a training set D_{train}
- a testing set D_{test}

Train our model on D_{train}, evaluate on D_{test}

Now it is fine, we split training and testing!

Do you see any issue?

Selection bias

Your results will depend on how your data was split into test/train sets.

Most of the time, it is a random shuffling, but you coud get lucky, or unlucky, in the way the data was shuffled.

These effects are lessened in the case of very big datasets.

Particular case of temporal data

Take a lot of care when using time series, or temporal data: the behavior of data can change over time

Split the data intelligently!

K-fold cross validation

In order to leverage all of your data, split the dataset into K folds. Run your model and evaluate it K times, by holding out a different fold each time for evalutation.

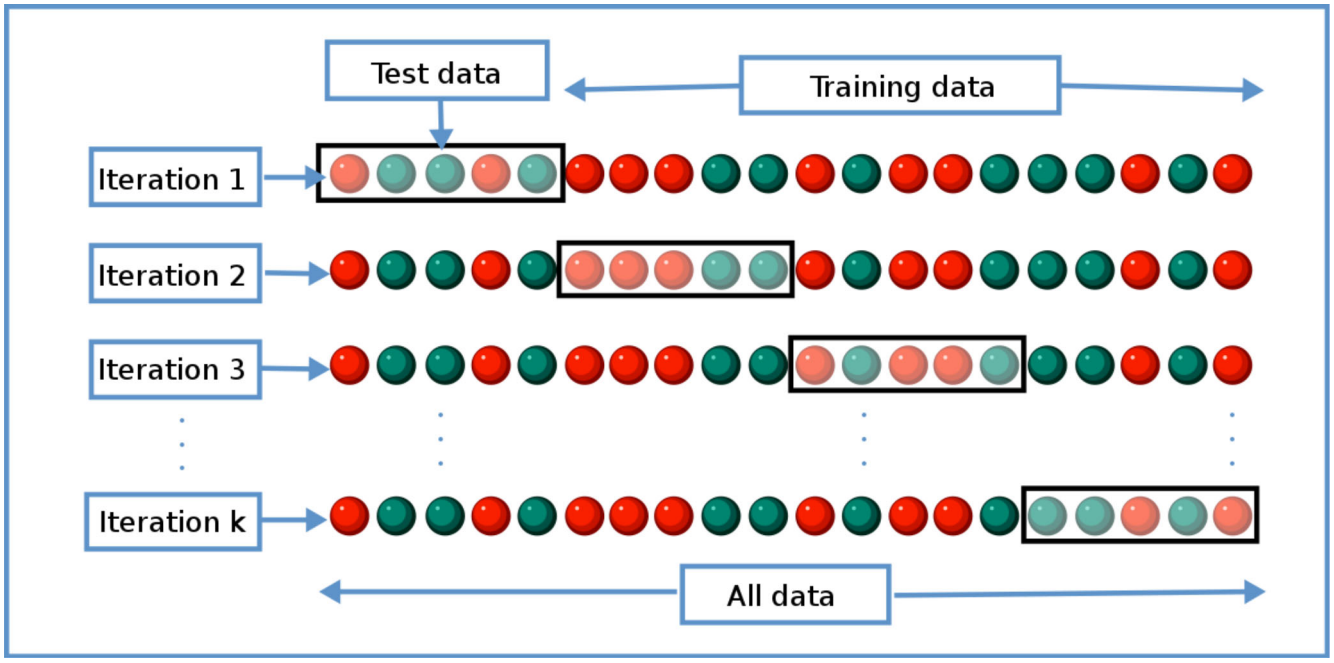
We split the dataset D into K folds.

For each iteration k:

- the k^{th} fold is set aside for testing
- the remaining (k-1) folds are merged together and used for training

Draw statistics over your whole dataset.

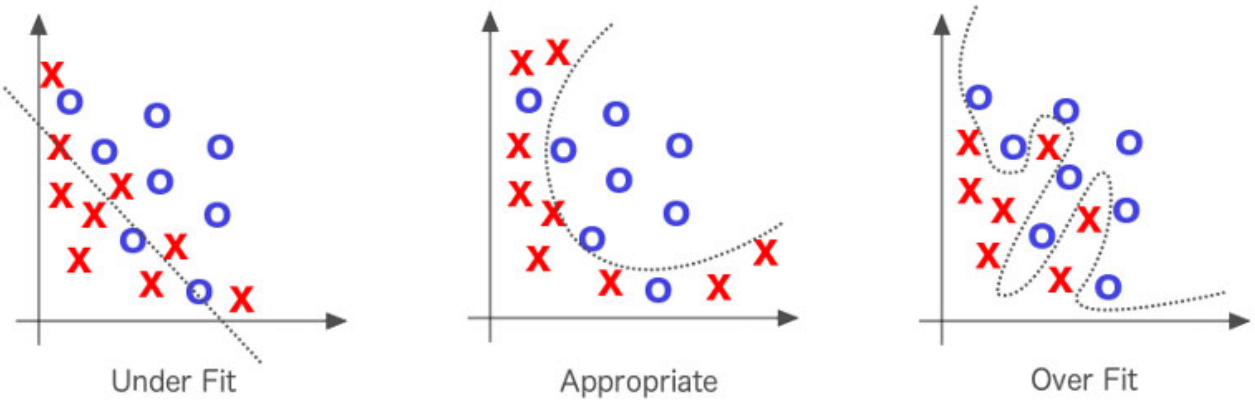
The larger the number of folds, the more precise the evaluation.



Overfitting

When optimized on the training set, given enough modelling capacity, the model will converge by itself to a solution that fits all the training data perfectly.

But the model will not generalize to new data!

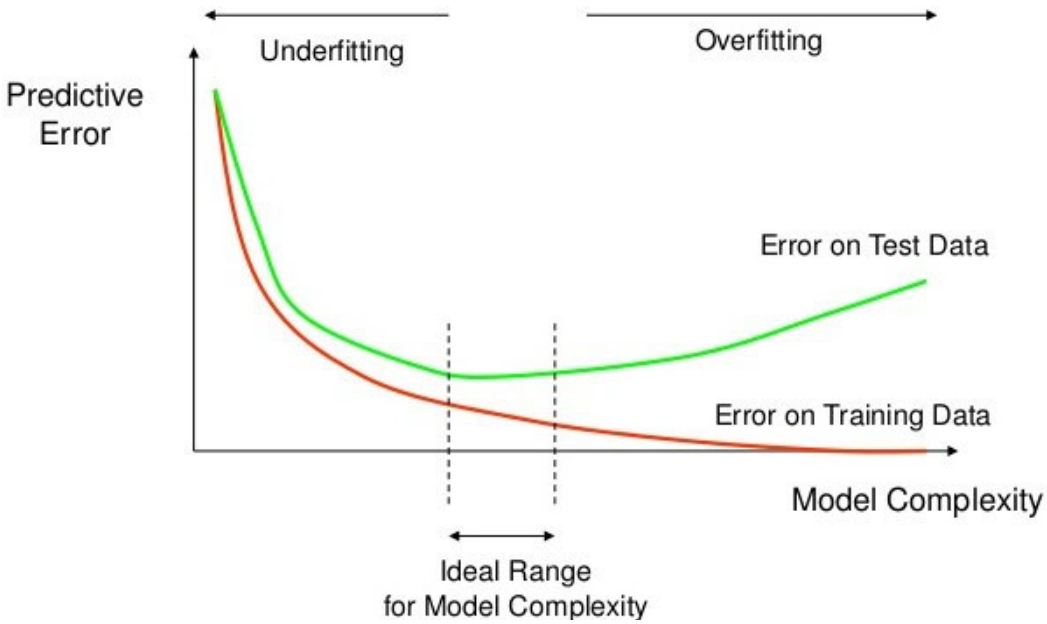


Stopping criterion

Several strategies to prevent overfitting:

- decreasing learning rate
- early stopping
- regularization

Machile Learning Libraries do it automatically, and set aside a Validation Set within the training set.



Model selection

We have access to a huge quantity of potential model, each of them with hyperparameters.

Remember the no free lunch theorem!

How do we choose one on particular?

Classical approach to model selection

1. Plan loops in loops to to test all possible models and hyperparameters
2. Start the process
3. Go on holiday (your code is working for you)
4. Come back, realize that you made a mistake somewhere
5. Iterate

There are more clever ways to do it

Model selection:

- intuition
- first try with broad range of parameters to get a feeling of what might work

Automated hyperparameter search:

- grid search
- bayesian parameter search

In the lab, we will just try to iterate over models and parameters because, in our case, it is not computationally prohibitive.

Final step-by-step approach

For each model that you want to evaluate:

1 - Split your dataset into k folds.

2 - Evaluate your model on the whole dataset:

- train on the training set (with a potential validation set)
- evaluate on the testing set
- do it for every fold

3 - Select the model with best performance over the whole dataset

4 - You can draw statistics (mean and variance) on the evaluation by running your K-fold shuffling multiple times

II - Predicting a value: regression

Examples:

- Predicting the future value of Bitcoins
- Estimating the price of an appartment
- Predicting a life expectancy

Also known as: forecasting

Definitions

A regression problem is composed of :

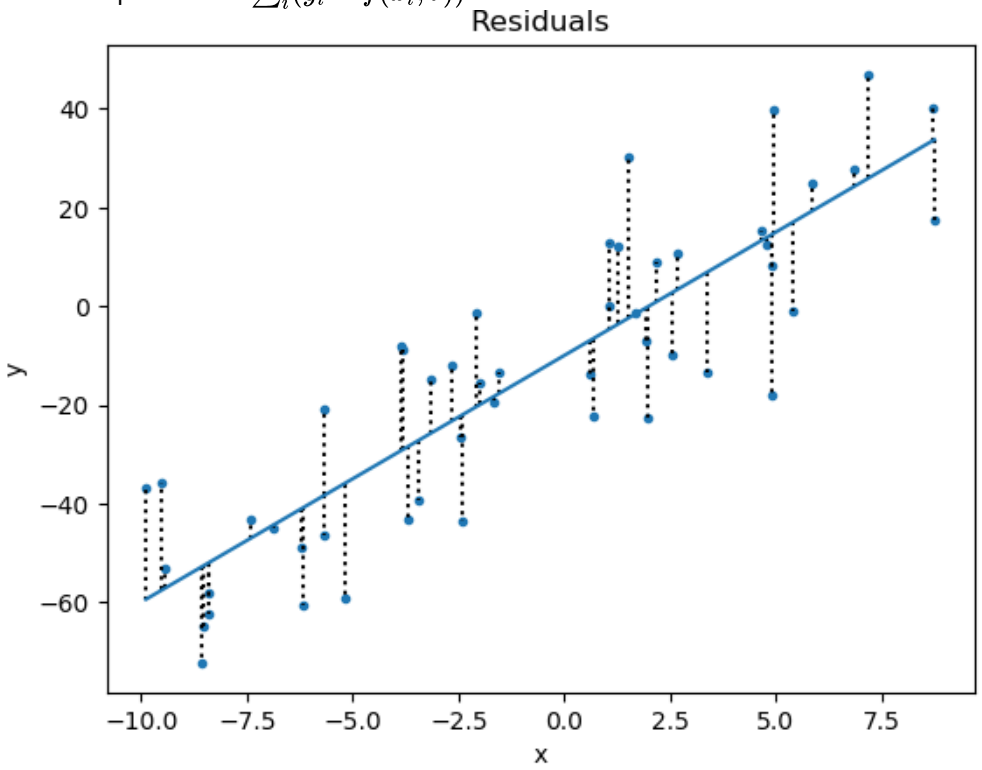
- input data x_i
- target output data y_i
- a model $f(x_i, \theta)$ parameterized by θ
- an objective to optimize the parameters to get $f(x_i, \theta)$ as close as possible to y_i

Linear Regression

A linear regression is linear in terms of the parameters.

The model is a linear combination of the features (that can be expanded).

We minimize the sum of squared error $\sum_i (y_i - f(x_i, \theta))^2$



Simple Linear Regression

One input, one output, and the model is a straight line.

The predictive model is:

$$y^* = \alpha + \beta x$$

We want to minimize the sum of squared errors:

$$\sum_i (y_i - (\alpha + \beta x_i))^2$$

We can solve it analytically: Ordinary Least Square method.

Intuition about the least square method

The sum of squares S is :

$$\sum_i (y_i - (\alpha + \beta x_i))^2$$

The minimum of S corresponds to the gradient of S being 0.

We can use this to find the parameters.

In [2]:
```
# From sklearn website

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = datasets.load_diabetes()

# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.show()
```

```
Coefficients:
 [938.23786125]
Mean squared error: 2548.07
Variance score: 0.47
```


Multiple Linear Regression

Multiple inputs, one output.

The predictive model is:

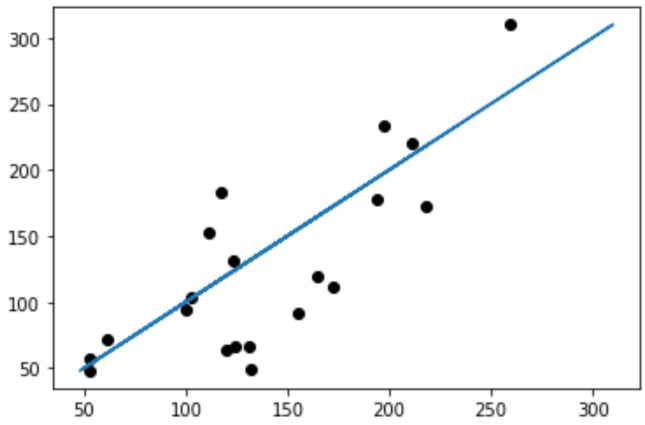
$$y^* = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_{n-1} x_{n-1}$$

We can also solve it analytically.

In [3]:
```
# Split the data into training/testing sets
diabetes_X_train = diabetes.data[:-20]
diabetes_X_test = diabetes.data[-20:]
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# linear regression object
regr = linear_model.LinearRegression()
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
```

In [4]:
```
print('Coefficients: \n', regr.coef_)
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
print('Variance score: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_y_pred, diabetes_y_test,  color='black')
plt.plot(diabetes_y_test,diabetes_y_test)
plt.show()
```

```
Coefficients:
 [ 3.03499549e-01 -2.37639315e+02  5.10530605e+02  3.27736980e+02
 -8.14131709e+02  4.92814588e+02  1.02848452e+02  1.84606489e+02
  7.43519617e+02  7.60951722e+01]
Mean squared error: 2004.57
Variance score: 0.59
```



General Linear Regression

In this model, we don't try to predict a single value, but a vector of values.

In matrix form: $Y = XB + U$ with:

- Y multivariate measurements we want to predict (output / target)
- X observations (input)
- B parameters
- U errors / noise

Polynomial Regression

This model is a linear combination of polynomial expansions of the input data.

$$y^* = \beta_0 + \beta_1 x^1 + \beta_2 x^2 + \ldots + \beta_{n-1} x^{n-1}$$

Still linear in the parameters, so Least Square Analysis is still valid.

We can always consider the polynomial to be new features created through polynomial expansion

Goodness of fit: Coefficient of determination R^2

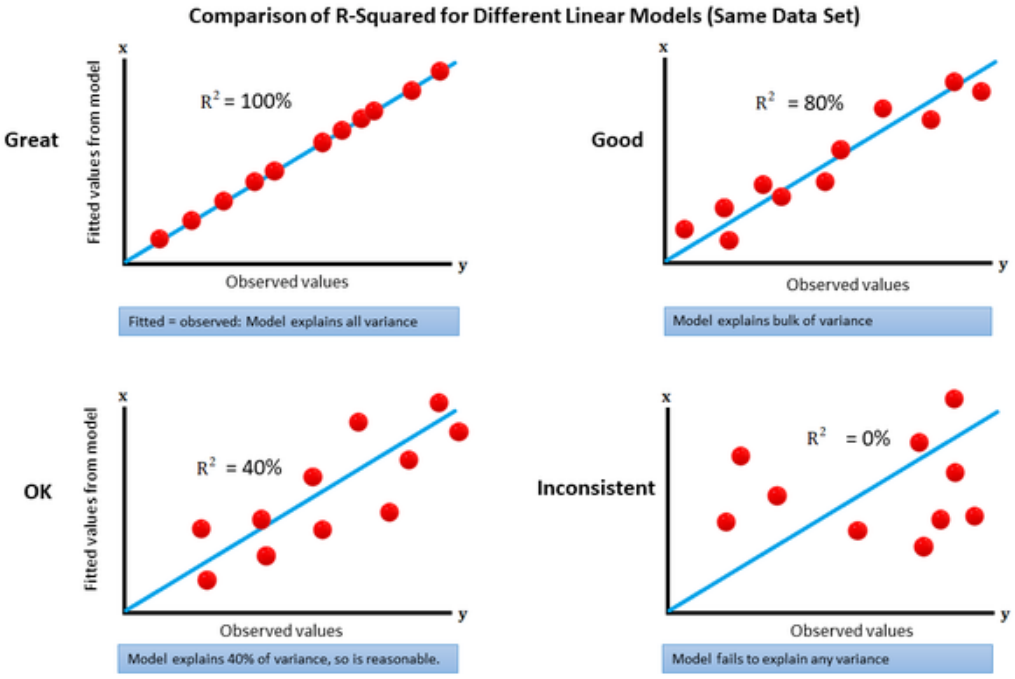
Mean of the observed data: $\bar{y} = \frac{1}{n} \sum_i yi$

Total sum of squares: $SS_{tot} = \sum_i (y_i - \bar{y})^2$

Residual sum of squares: $SS_{res} = \sum_i (y_i - y_i^*)^2$

Then, we can calculate :

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$



Interpretation:

- $R^2 = 1$ perfect fit
- closer to 1, the better

Idea: plot the residuals

Other topics:

Regression using basis functions:

$$y^* = \beta_0 + \beta_1\phi_1(x) + \beta_2\phi_2(x) + \ldots + \beta_{n-1}\phi_{n-1}(x)$$

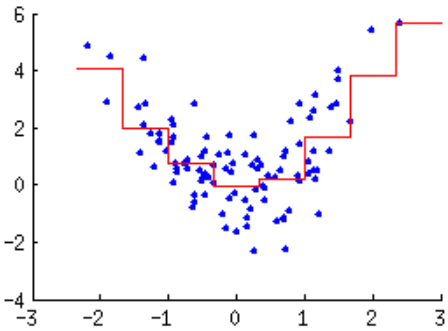
Regularization to avoid Overfitting

Support Vector Regression

Neural Networks (Deep Learning)

A regression problem can always be transformed into a classification problem

- predicting a change in price: going up or down
- predicting a price: binning, k-means, ...



III - Classification

In Classification, we want to predit a class given an input vector.

Different classification problems:

- binary classification (yes/no, cat/dog)
- multi-class classification (voting in France)

Automatic marking of a student?

- Input: report, code, previous marks
- Output: grade for the Coursework

Is it classification or regression?

Binary Classification with logistic regression

Logistic regression models the probability of a class in the binary setting.

Linear Regression:

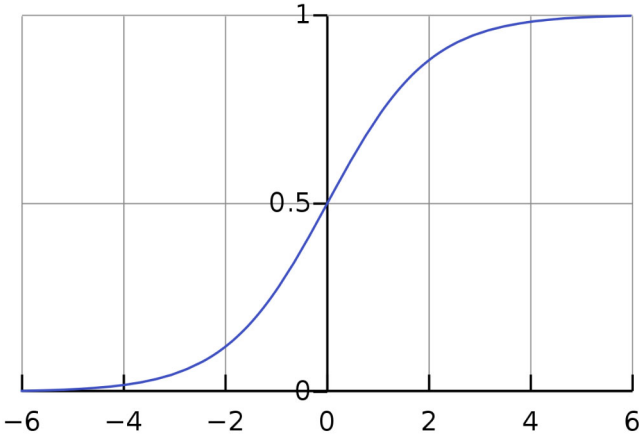
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_{n-1} x_{n-1}$$

Logistic regression:

$$log(\frac{p}{1-p}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_{n-1} x_{n-1}$$

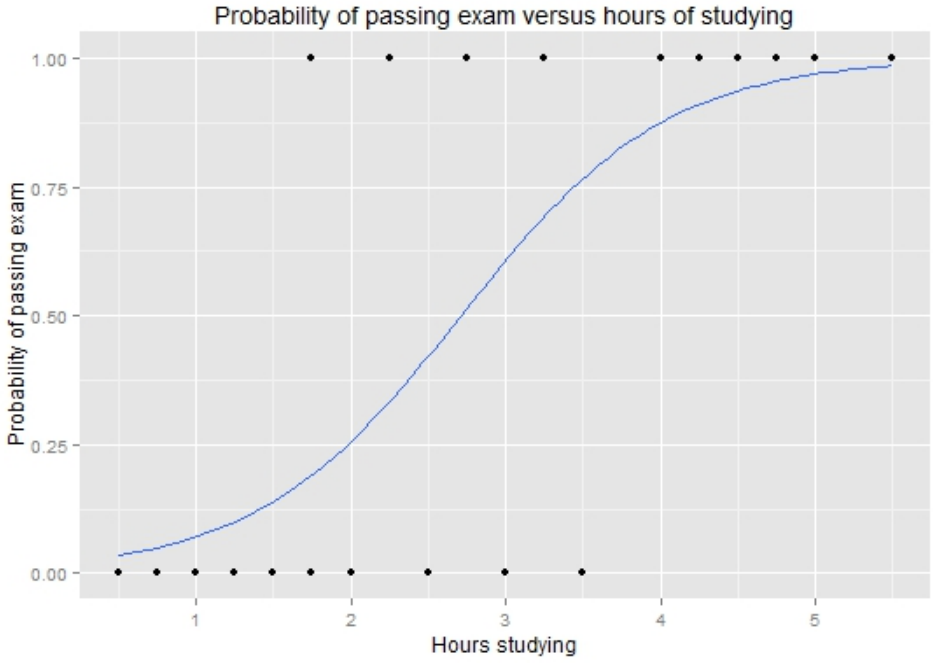
We can write:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_{n-1} x_{n-1})}}$$



No close form solution (no least square), so we use Maximum Likelihood Estimation and Gradient Descent.

In the end, Logistic regression tries to find the hyperplan that splits the dataset into class = 0 and class = 1.



In [27]:
```
# Example of binary classification

from sklearn.linear_model import LogisticRegression

young_np = young_dataset.iloc[:,:135].apply(lambda x: pd.factorize(x)[0]).to_num
py()

target = young_dataset.iloc[:,144].factorize()[0] #gender
#target = young_dataset.iloc[:,145].factorize()[0] #lefthanded
#target = young_dataset.iloc[:,147].factorize()[0] #only child

X_train = young_np[ :int(len(young_np)*0.9 ), : ]
y_train = target[ :int(len(target)*0.9 ), ]

X_test = young_np[ int(len(young_np)*0.9):, :]
y_test = target[ int(len(target)*0.9):, ]

classifier = SVC(gamma = 'auto').fit(X_train, y_train)

print('score: ', classifier.score( X_test, y_test))
```

```
score:  0.8235294117647058
```

In [28]:
```
cm = confusion_matrix(y_test, classifier.predict(X_test))

print(cm)
```

```
[[38  6]
 [ 6 18]]
```

Extensions of binary logistic regression

Multinomial logistic regression

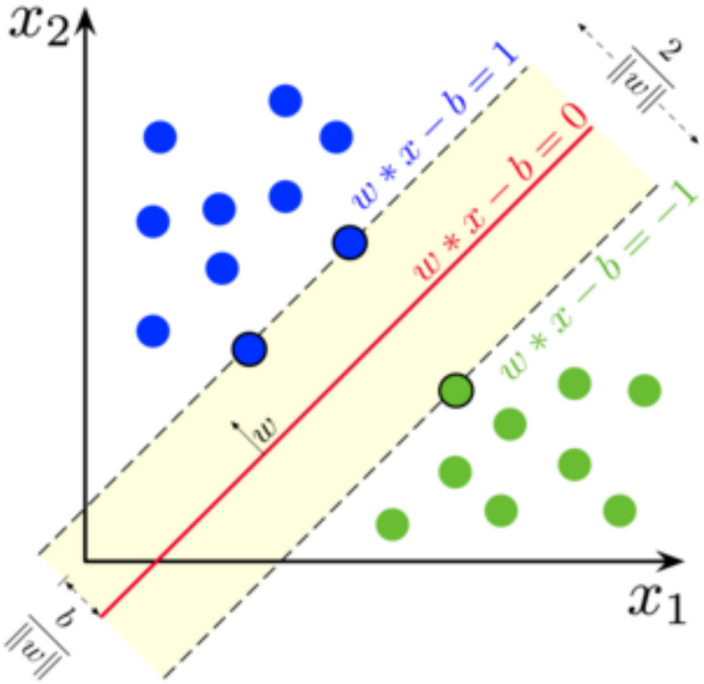
Ordered Logistic Regression

SVM

Used for (originally) binary classification (regression versions exist).

Main idea:

- represent the input points in a new space
- so that there is a clear separation between classes
- find the highest margin possible

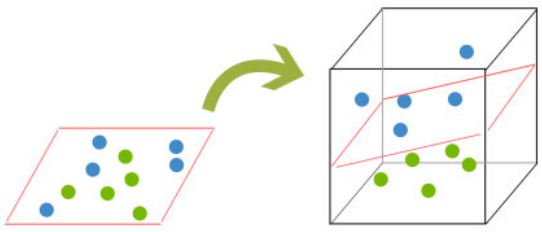


Different flavours of SVMs

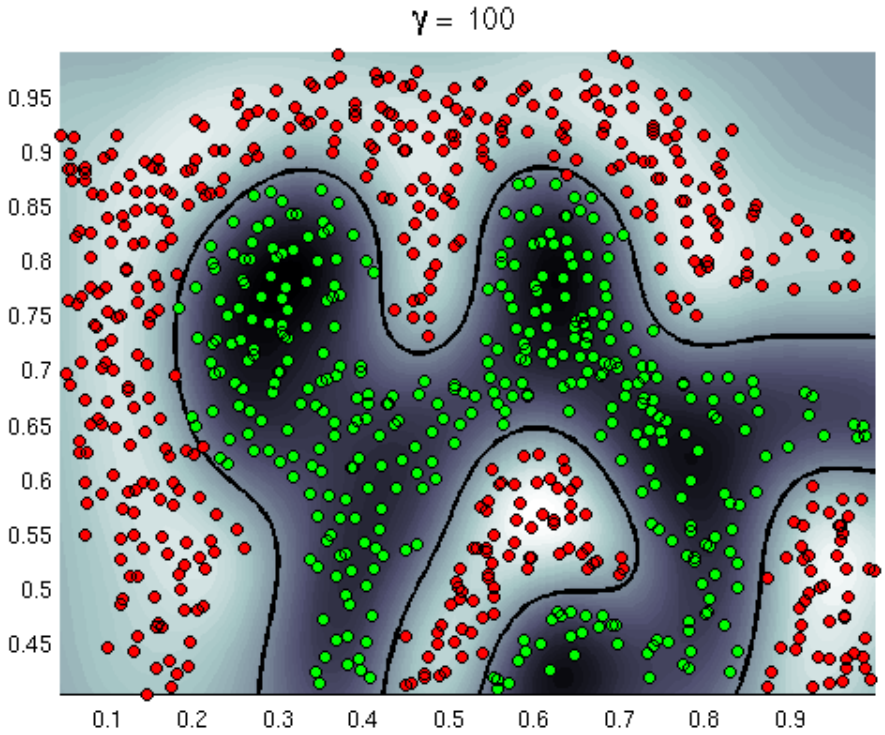
Hard margin / Soft margin : do we allow wrong classification during training?

Kernel trick:

- original version uses linear combination of features (linear svm)
- improved versions use different projections:
  - polynomial
  - rbf



Goal: project/map features to a high dimensional feature space where classes become linearly separable



Standard measures

We can evaluate on our whole testing set, for a particular value of τ our decision threshold, if a prediction is :

True Positive (TP) Patient has pneumonia Model predicts: pneumonia Number of occurrences: 1	**False Positive (FP)** Patient is healthy Model predicts: pneumonia Number of occurrences: 1
False Negative (FN) Patient has pneumonia Model predicts: healthy Number of occurrences: 8	**True Negative (TN)** Patient is healthy Model predicts: healthy Number of occurrences: 90

Based on these measures, we can describe the quality of our classifier

Accuracy of Binary classifier

Our model can be evaluated by looking into different metrics.

The first one is Accuracy:

$$ACC = \frac{TP + TN}{P + N}$$

Most classifiers will return accuracy as a score.

Advanced Evaluation of Binary classifier

Most classifiers can return a probability that a class is predicted.

They output $p = P(Y = 1|x)$ for each x in the dataset. Based on that, we can make a decision with a threshold τ:

- if $p > \tau$, we decide that the model predicts the class 1
- else, we decide that the model predicts the class 0

By varrying this threshold, we can influence the sensitivity of our classifier.

If τ is very high, then our model only predicts the class 1 when it is very certain.

Similarly, if τ is very low, our model predicts the class 0 when it is very certain.

Sensitivity

Also known as True Positive Rate, Recall:

$$TPR = \frac{TP}{P}$$

Proportion of actual positives that are classified as such.

Important when you don't want to miss a positive. Example: medical diagnostic.

Specificity

Also known as True Negative Rate, Selectivity:

$$TNR = \frac{TN}{N}$$

Miss Rate

Also known as False Negative Rate:

$$FNR = \frac{FN}{P}$$

Fall-out

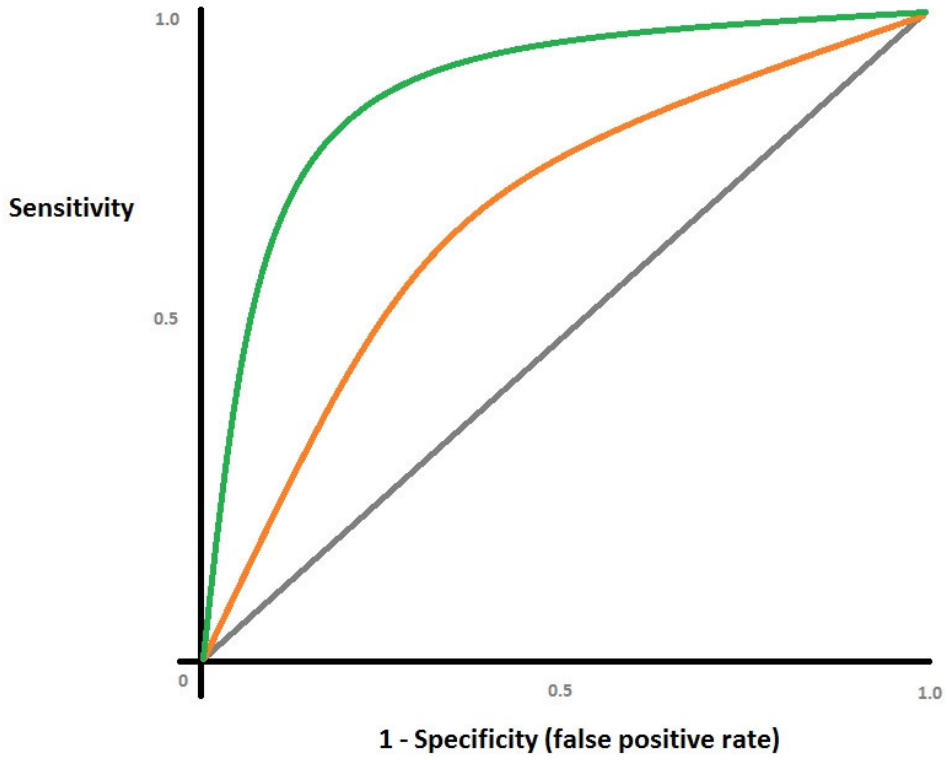
Also known as False Positive Rate:

$$FPR = \frac{FP}{N}$$

Probability of false alarm.

ROC curve

Now that we have a measure of TPR and FPR, for different values of τ, we can plot a ROC curve.



From binary to muticlass classification

Any binary classifier can be transformed to multiclass classifier:

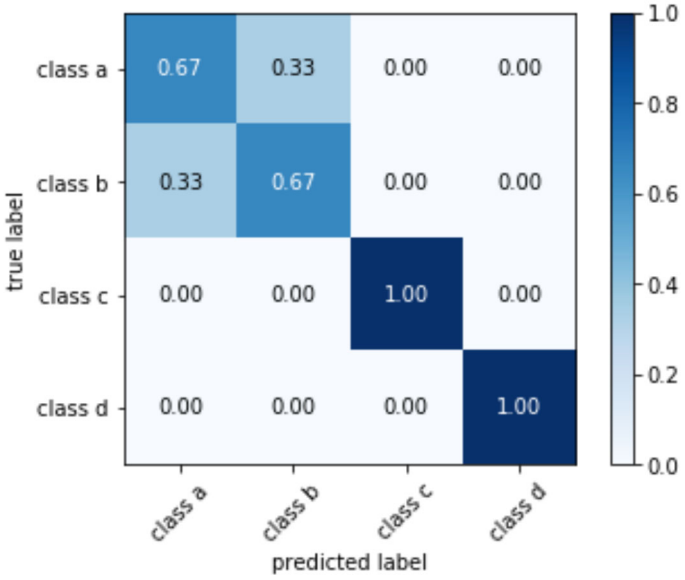
- One vs Rest: Run K classifiers, prediction is the highest confidence
- One vs One: Run K(K-1)/2 classifiers, voting scheme

Else, algorithms can be modified to take into account multiclass (for example, SVM)

How to evaluate multiclass?

One option is to perform pairwise comparisons and ROCs.

Very good and visual way: Confusion matrix



Conclusion on classification:

- find the right metric for your problem
- if possible, start with binary classification
- take care of class imbalance

III - Introduction to Deep Learning

Why?

- your friends talk about it
- your colleagues talk about it
- your parents watched a documentary about it
- your boss read an article about it and wants you to do it

And here you are with your linear regressions...

More seriously, today we collect enormous multidimensional databases.

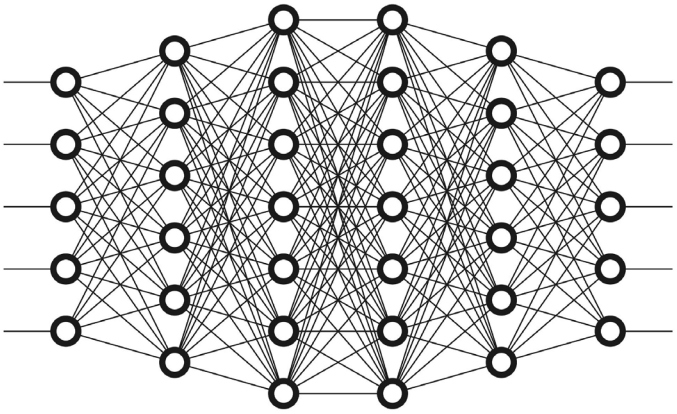
Classical approaches are limited in terms of computational power and expressivity.

Deep Learning solves this, and actually thrives on Gigabytes of data and huge computing farms.

Neural Networks

Machine Learning model formed by associating numerous neurons with each other, organized in layers:

- distributed computing
- learn intermediary representations
- used for any kind of Machine Learning problem: general function approximator



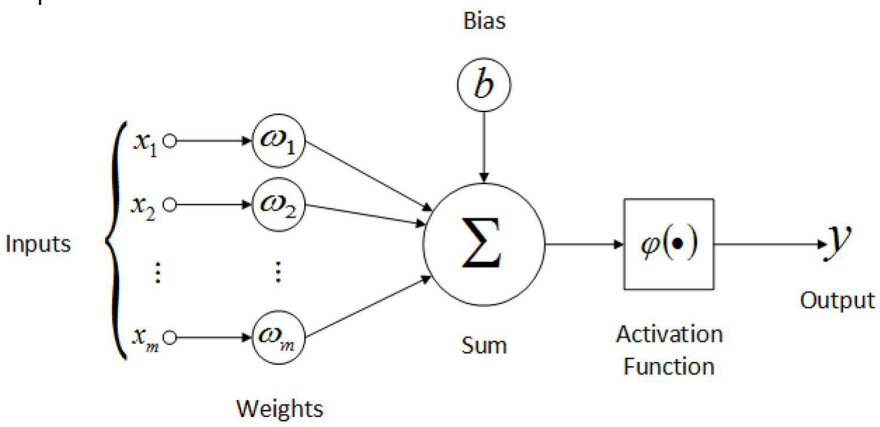
Inspiration from human brain (a little): layers of neurons.

Neurons and Activation functions

A neuron takes a group of inputs, performs a weighted sum which is passed through an activation function:

$$y = \phi(\sum_j w_j x_j)$$

It is the elementary computation unit of a Neural Network

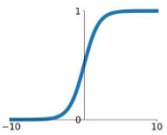


In Deep Learning, we different types of activation functions:

- Rectifier linear unit: represent a feature
- sigmoid: represent a class

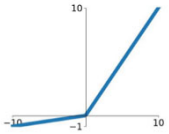
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



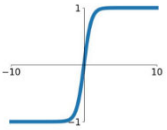
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

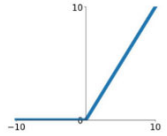


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

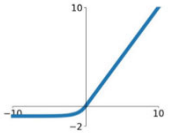
ReLU

$\max(0, x)$

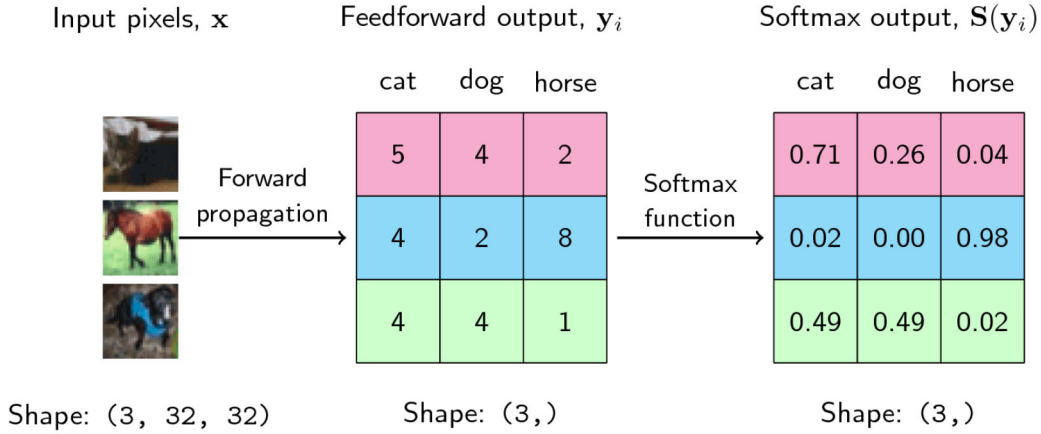


ELU

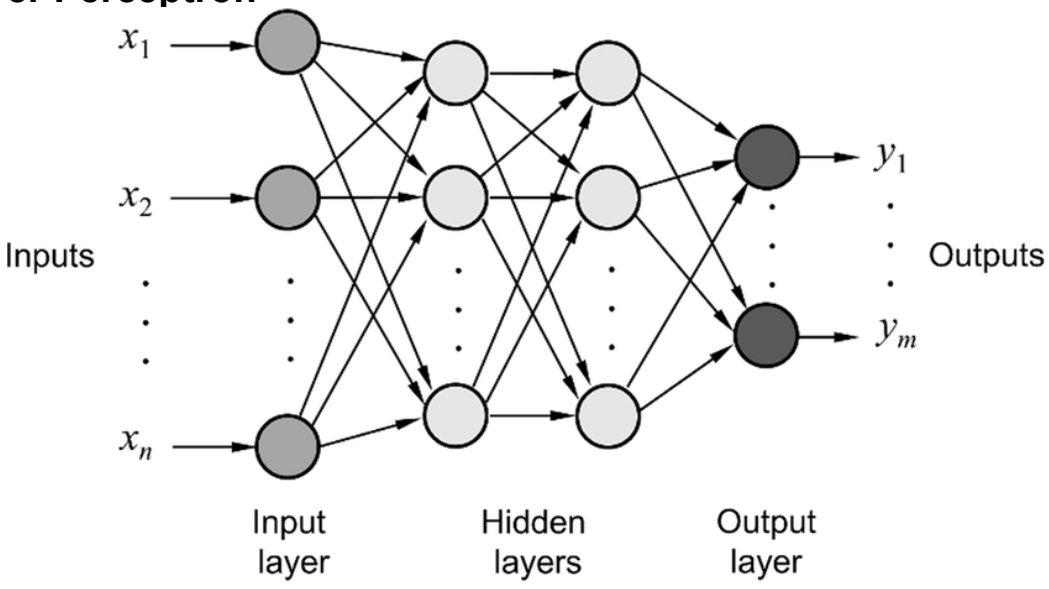
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Softmax: sigmoid normalized across a layer: represent probability in the multiclass case



Multilayer Perceptron



Convert our objective into a loss function

Define a loss function to optimize your network.

Regression:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2$$

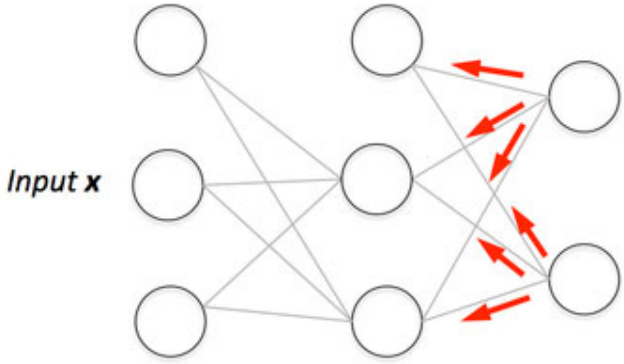
Binary Classification with Binary Cross Entropy:

$$J = -\frac{1}{N}\sum_{i=1}^{N}[y_n log(\hat{y}_n) + (1-y_n) log(1-\hat{y}_n)]$$

For multiclass: we calculate a separate loss for each class.

Optimization

Gradient descent and backpropagation



All the weights can be adapted in a single pass (differentiable architectures).

Optimization: SGD, Adam, ...

Mini-batch learning: Adapt the weights step by step.

Why is deep learning so successful?

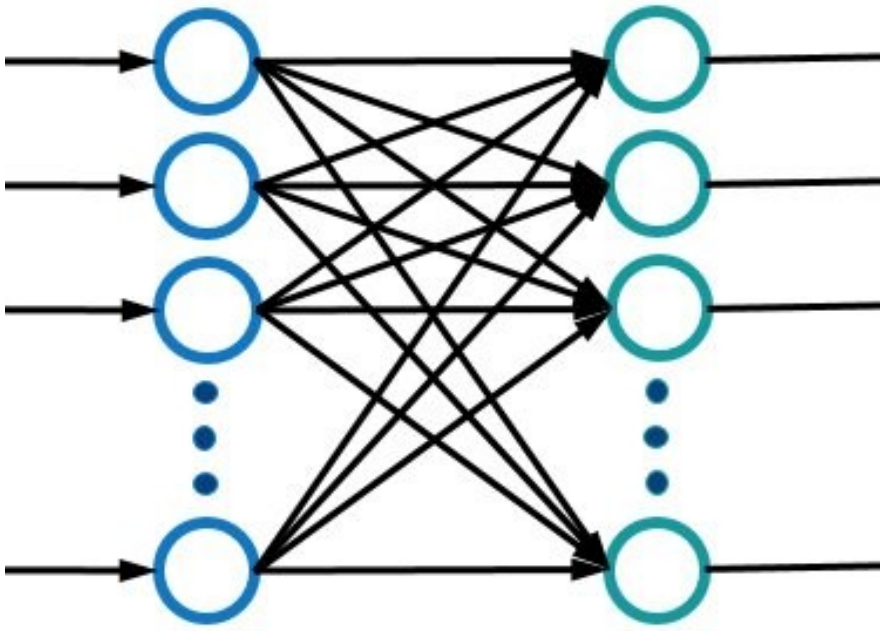
All units perform very similar operations. It is possible to parallelize a lot!

Through raw speed and computing power, we can build huge models and leverage enormous quantities of data.

Think about it if your data is: very high dimensional + lots of samples.

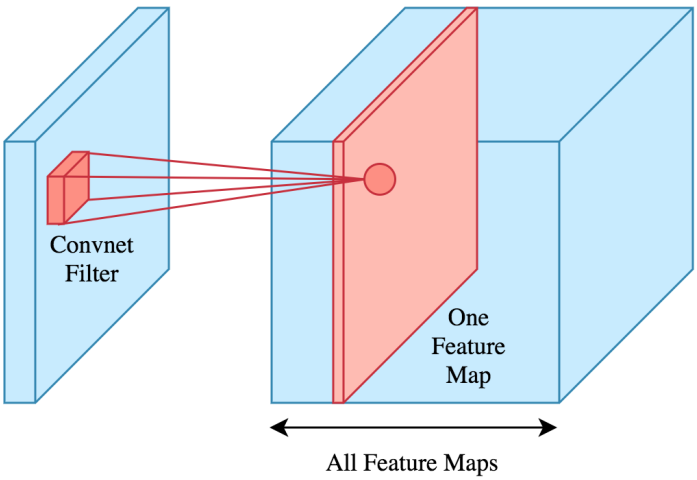
Dense layers

All the neurons of the layer are connected to all the neurons of the previous layers.



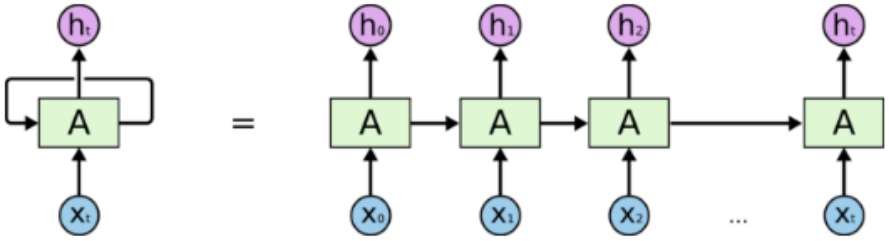
Convolutional Layers

A few number of localized neurons in the previous layer are connected to a single neuron of the convolutional layer.



Recurrent layers

Instead of the classical feed-forward neural network, we allow backward connections to represent temporal data.



An unrolled recurrent neural network.

The state of a RNN (its hidden layer) receives inputs from a previous layer and from its own past state.

Overall Deep Learning Architecture

A Deep Learning Architecture is composed of:

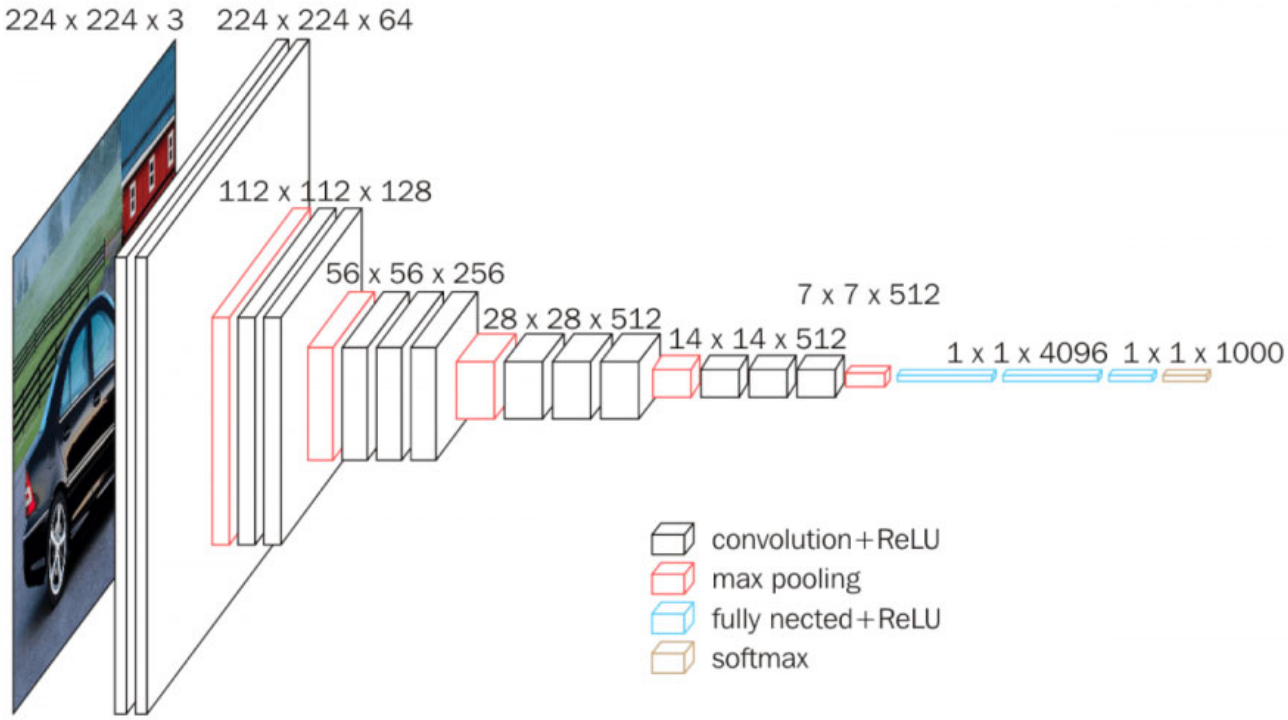
- a succession of layers
- a loss function that translate our objective / idea into a quantity that we want to optimize
- an optimization algorithm to minimize this loss function

Push the button, and let it run!

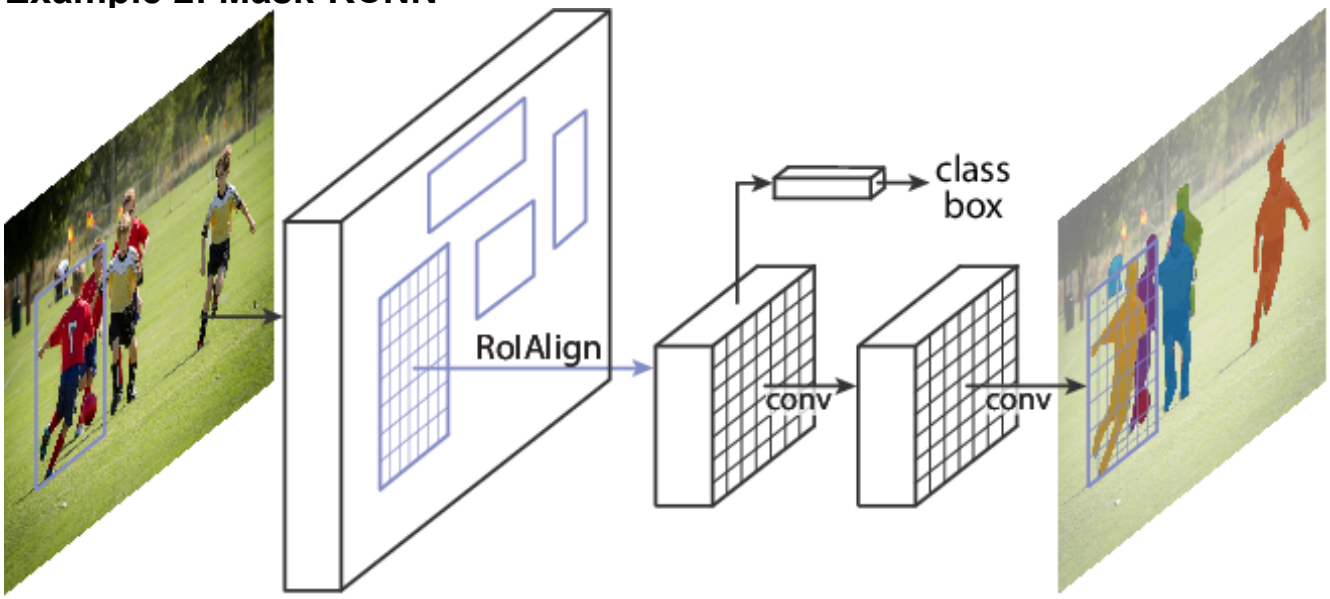
Other topics: regularization, normalization, vanishing / exploding gradient

Very empirical approach.

Example 1 : VGG

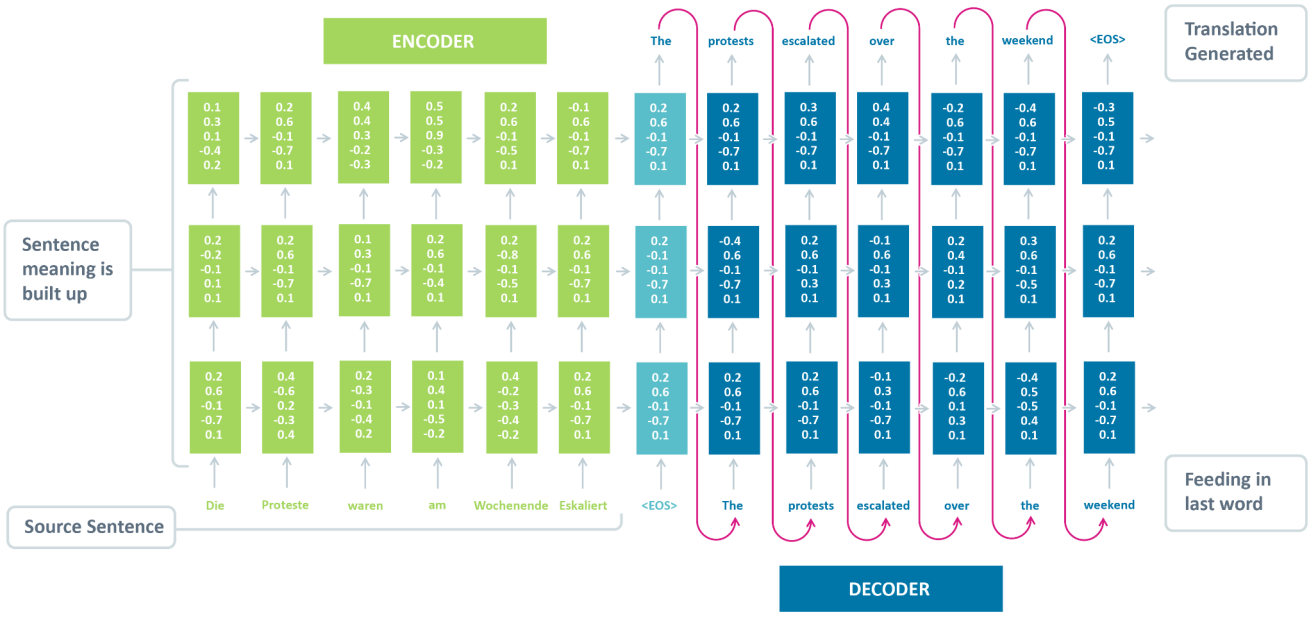


Example 2: Mask-RCNN



Example 3: Neural Machine translation

A Recurrent Neural Network for Machine Translation



Advantages and limitations

There are several limitations:

- requires huge datasets
- convolutions benefit from a GPU
- not interpretable: black box
- hyperparameter space is enormous

But a lot of benefits:

- no more feature engineering
- proved to be extremely powerful
- will work for a great range of hyperparameters
- plenty of already existing models and libraries

IV - Conclusion

1. Assess your problem, decide on a sensible metric.
2. Test different algorithms, there is no one-algorithm-fits-all.
3. Evaluate properly to gain solid insights