Hands-on Workshop on Machine Learning Applied to Medical Imaging

Introduction to Machine Learning

Jonathan Weber[†], Cédric Wemmert[*], Germain Forestier[†]

March 9, 2020

[†] IRIMAS Mulhouse, [*] ICube Strasbourg

Outline

Introduction

What is data science?

- Knowledge extraction from data
- Linked to many domains:
    - artificial intelligence
    - machine learning
    - Deep learning
    - Data mining
    - Big Data

an many more... *Business Intelligence*, *Data Analytics*, *Data vizualisation*, *KDD*, etc.

Artificial intelligence

- "Designing machines capable of simulating intelligence"
- Robotics, games, chatbot, etc. and machine learning

Machine learning

- "Training" a machine to perform a specific task
- "Learning" needs examples = data!

Deep Learning

- A specific family of machine learning algorithms (neural nets)
- Need a lot of data = many examples

Some definitions

Artificial intelligence

- "Designing machines capable of simulating intelligence"
- Robotics, games, chatbot, etc. and machine learning

Machine learning

- "Training" a machine to perform a specific task
- "Learning" needs examples = data!

Deep Learning

- A specific family of machine learning algorithms (neural nets)
- Need a lot of data = many examples

Some definitions

Artificial intelligence

- "Designing machines capable of simulating intelligence"
- Robotics, games, chatbot, etc. and machine learning

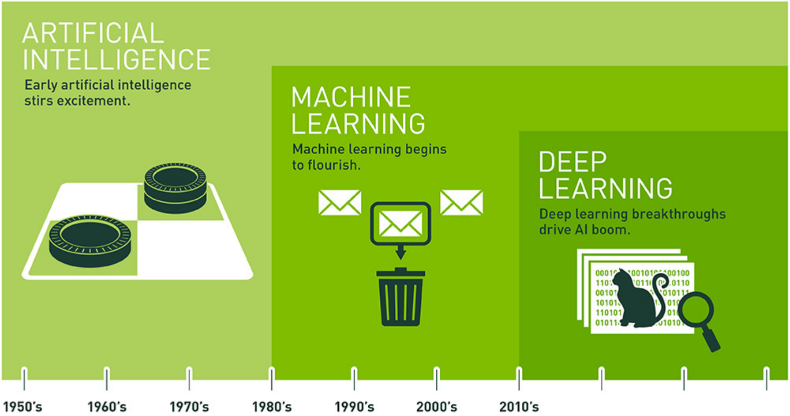
Machine learning

- "Training" a machine to perform a specific task
- "Learning" needs examples = data!

Deep Learning

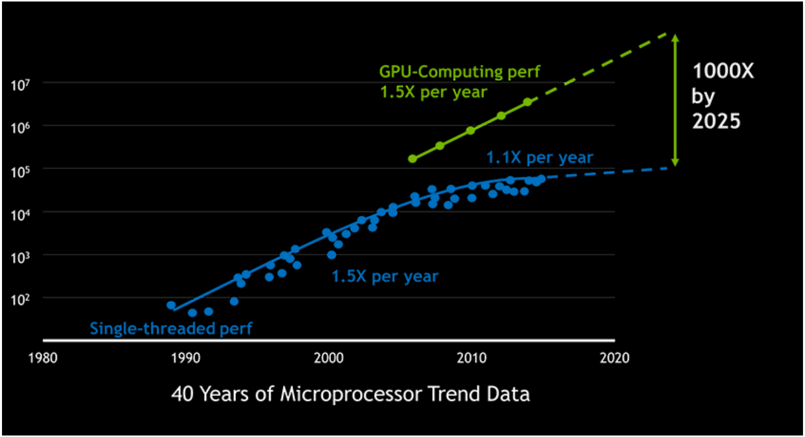
- A specific family of machine learning algorithms (neural nets)
- Need a lot of data = many examples

Artificial intelligence, machine learning, deep learning:



source: `nvidia.com`

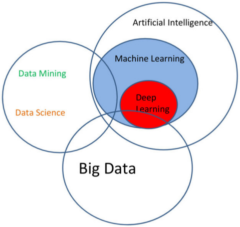
Artificial intelligence, machine learning, deep learning:



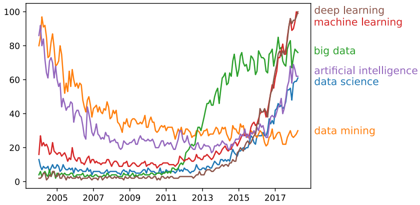
source: `nvidia.com`

Data Science Mapping

- All the tools and algorithms for data processing
- Data mining + AI + Machine learning + Big Data

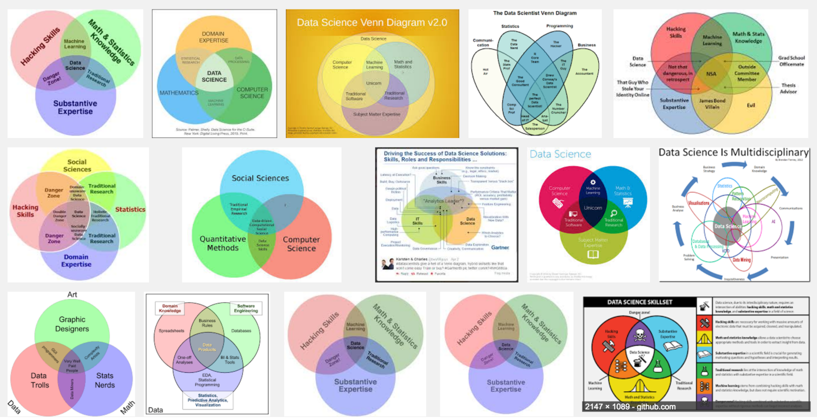


source: KDnuggets.



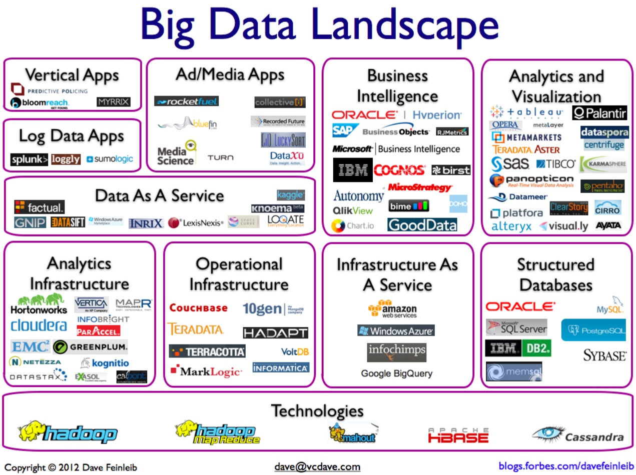
source: Google Trends

Data Science Mapping

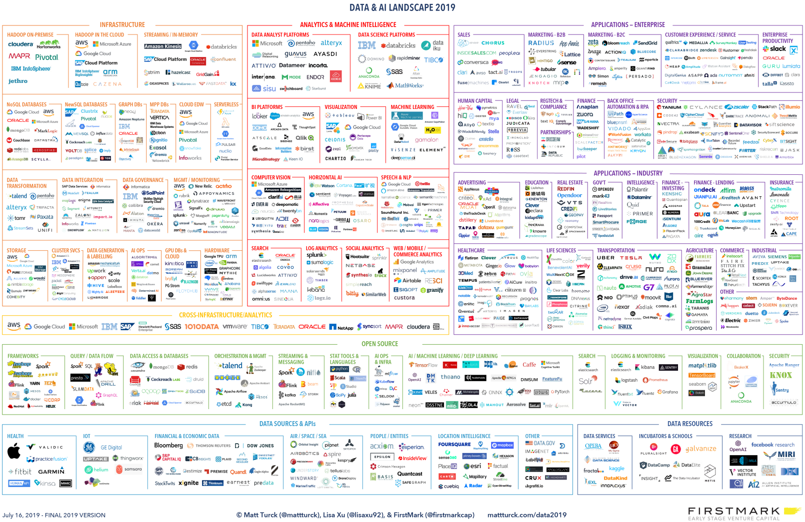


source: `https://matthewlincoln.net/2016/11/23/histories-of-data.html`

A rich ecosystem:



A rich ecosystem:



Data types

- Structured data: data associated to metadata
- Unstructured data: text, images, video, etc.

Example of structured data:

	attribute 1	attribute 2	attribute 3
Instance 1	1.1	small	yes
Instance 2	2.1	small	no
Instance 3	1.7	large	no

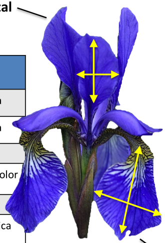
- Data types: real, integer, string, boolean, etc.
- Location: database, data warehouse, files, cloud, etc.

Example of dataset

- Input data: width and length of petals and sepals + species
- **Goal**: to predict the iris species

Samples
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica



Features
(attributes, measurements, dimensions)

Class labels
(targets)

source: https://rpubs.com/wjholst/322258

Example of dataset

- Input data: width and length of petals and sepals + species
- **Goal**: to predict the iris species

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
6.7	3.1	4.7	1.5	versicolor
5.6	3	4.1	1.3	versicolor
6.3	2.8	5.1	1.5	virginica
6.7	3.3	5.7	2.5	virginica
6.7	3	5.2	2.3	?

Example of dataset

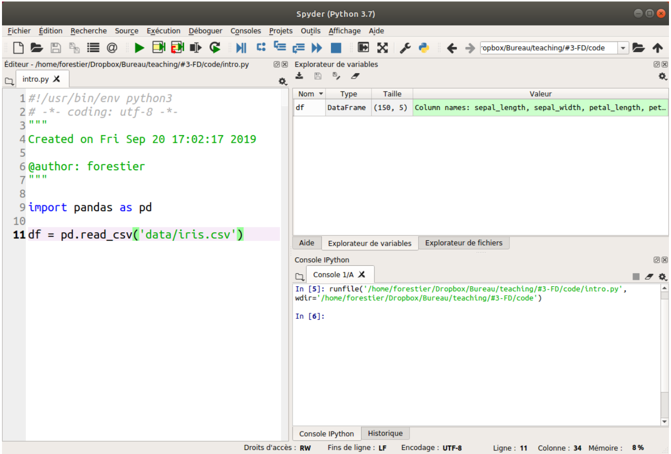
- Input data: width and length of petals and sepals + species
- **Goal**: to predict the iris species

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
6.7	3.1	4.7	1.5	versicolor
5.6	3	4.1	1.3	versicolor
6.3	2.8	5.1	1.5	virginica
6.7	3.3	5.7	2.5	virginica
6.7	3	5.2	2.3	?

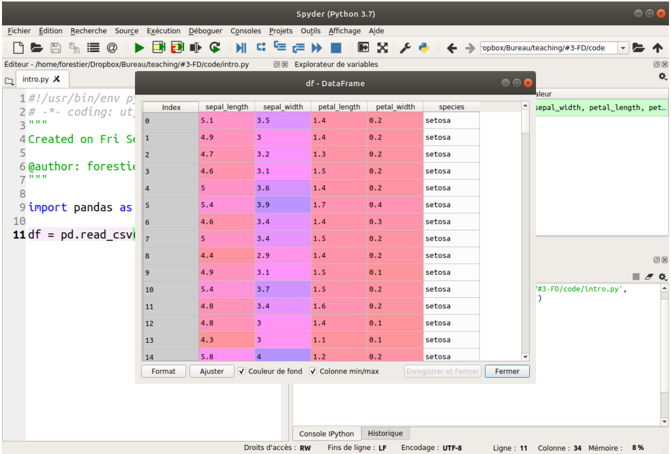
Example of dataset

```
sepal_length,sepal_width,petal_length,petal_width,species
5.1,3.5,1.4,0.2,setosa
4.9,3,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
5,3.6,1.4,0.2,setosa
5.4,3.9,1.7,0.4,setosa
4.6,3.4,1.4,0.3,setosa
5,3.4,1.5,0.2,setosa
4.4,2.9,1.4,0.2,setosa
4.9,3.1,1.5,0.1,setosa
5.4,3.7,1.5,0.2,setosa
4.8,3.4,1.6,0.2,setosa
4.8,3,1.4,0.1,setosa
4.3,3,1.1,0.1,setosa
5.8,4,1.2,0.2,setosa
5.7,4.4,1.5,0.4,setosa
5.4,3.9,1.3,0.4,setosa
5.1,3.5,1.4,0.3,setosa
5.7,3.8,1.7,0.3,setosa
5.1,3.8,1.5,0.3,setosa
5.4,3.4,1.7,0.2,setosa
5.1,3.7,1.5,0.4,setosa
4.6,3.6,1,0.2,setosa
5.1,3.3,1.7,0.5,setosa
4.8,3.4,1.9,0.2,setosa
```

Example of dataset

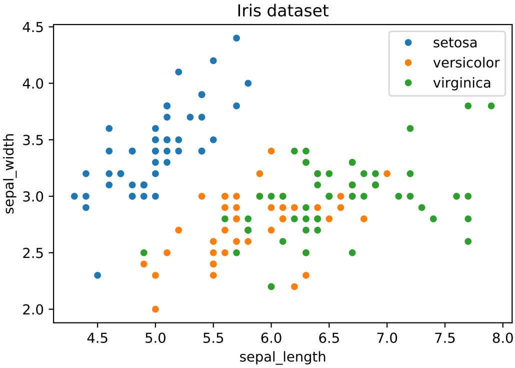


Example of dataset



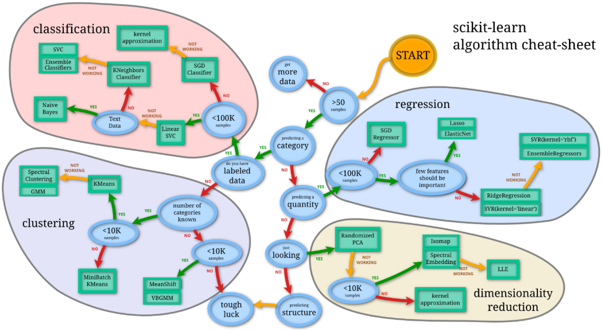
Exploratory Data Analysis (EDA)

- A good practice is to visualize the data before any treatment
- Allows to have intuitions on the distribution of classes



Different tasks in data analysis:

1. Classification: prediction of a discrete (countable) value
2. Regression: prediction of a continuous value
3. Clustering: automatic creation of groups (without notion of class)



source: `https://scikit-learn.org/`

Pandas

Pandas

- High-level package for data manipulation
- Based on `numpy`
- Allows you to manage different types of data
- The `DataFrame` object allows to easily manipulate data

```
import numpy as np
import pandas as pd

df = pd.DataFrame({'A' : 1.,
'B' : pd.Timestamp('20130102'),
'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
'D' : np.array([3] * 4,dtype='int32'),
'E' : pd.Categorical(["test","train","test","train"]),
'F' : 'foo' })

print(df)
print(df.dtypes)
```

Pandas

- High-level package for data manipulation
- Based on `numpy`
- Allows you to manage different types of data
- The `DataFrame` object allows to easily manipulate data

```
     A          B    C  D      E    F
0  1.0 2013-01-02  1.0  3   test  foo
1  1.0 2013-01-02  1.0  3  train  foo
2  1.0 2013-01-02  1.0  3   test  foo
3  1.0 2013-01-02  1.0  3  train  foo

print(df.dtypes)
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

Pandas

- The `DataFrame` object allows to select data
- The function `loc` selects elements by name
- The function `iloc` selects elements by index

```
df['A']
df[['A','B']]
df[0:3]

# selection by name
df.loc[:,['A','B']]

# selection by index
df.iloc[3]
df.iloc[1:3]
```

How to read data?

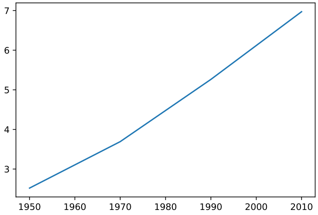
- It is often necessary to read data from files in order to process them and build predictive models.
- The function `genfromtxt()` of `numpy` allows to read files, but only with attributes of the same type
- The function `read_csv()` of `pandas` allows to read files with different types of attributes

```python
# data reading with numpy
import numpy as np
data = np.genfromtxt('data/iris.csv', delimiter=',',skip_header=True)

# data reading with pandas
import pandas as pd
df = pd.read_csv('data/iris.csv', header=0)
```

How to visualize data?

- Very important in data analysis
- Allows you to explore the data
- Allows you to report results
- `matplotlib` allows easy visualization of data

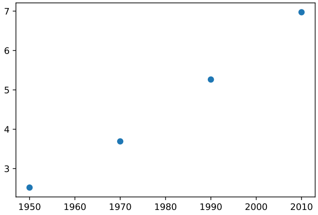
```
import matplotlib.pyplot as plt

year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]

plt.plot(year, pop)
plt.show()
```



How to visualize data?

- Very important in data analysis
- Allows you to explore the data
- Allows you to report results
- `matplotlib` allows easy visualization of data

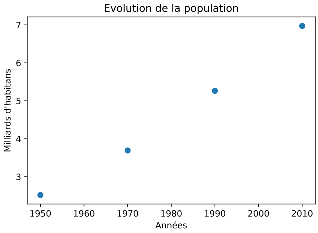
```python
import matplotlib.pyplot as plt

year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]

plt.scatter(year, pop)
plt.show()
```



How to visualize data?

- Very important in data analysis
- Allows you to explore the data
- Allows you to report results
- `matplotlib` allows easy visualization of data

```python
import matplotlib.pyplot as plt

year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]

plt.title('Evolution de la population')
plt.xlabel('Annees')
plt.ylabel('Milliards d\'habitants')
plt.scatter(year, pop)
plt.show()
```



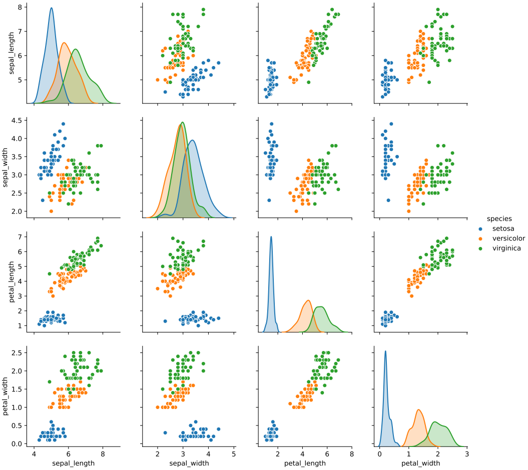
How to visualize data?

- The `seaborn` package is an alternative to `matplotlib`
- Adds functionality
- Allows advanced visualizations

```
import seaborn as sns
import pandas as pd

df = pd.read_csv('data/iris.csv', header=0)
ax = sns.pairplot(df, hue='species')
plt.title('Pairwise relationships between the features')
```

https://seaborn.pydata.org/

How to visualize data?



Validation (train/test)

- In order to be able to validate a model, it is important to be able to test its ability to make good predictions.
- We avoid testing a model on the same data used to build the model.
- Using the same data will tend to overestimate performance and does not test generalizability.
- The most classical method consists in dividing the dataset into two, one for training (*train*) and one for evaluation (*test*)

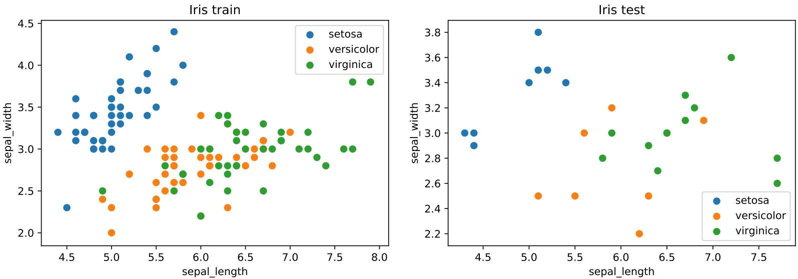
Validation (train/test)

- 80% of the data are kept for training and 20% for testing:

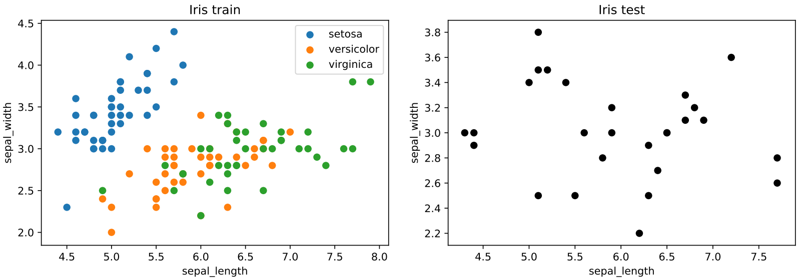
```
import pandas as pd
import numpy as np

df = pd.read_csv('data/iris.csv', header=0)
mask = np.random.rand(len(df)) < 0.8
df_train = df[mask]
df_test = df[~mask]
```

- Attention must be paid to the distribution of classes (number of elements in each class, which can be different).
- In order to keep the same distribution, we talk about stratification.

Validation (train/test)



Validation (train/test)



Validation (train/test)

sepal_length	sepal_width	species
5.1	3.5	setosa
4.9	3	setosa
6.7	3.1	versicolor
6.3	2.3	versicolor
6.3	2.8	virginica
6.1	2.6	virginica

Table 1: Train set

sepal_length	sepal_width	species
4.7	3.2	setosa
5.6	3	versicolor
6.7	3.3	virginica

Table 2: Test set

Validation (train/test)

sepal_length	sepal_width	species
5.1	3.5	setosa
4.9	3	setosa
6.7	3.1	versicolor
6.3	2.3	versicolor
6.3	2.8	virginica
6.1	2.6	virginica

Table 1: Train set

sepal_length	sepal_width	~~species~~
4.7	3.2	~~setosa~~
5.6	3	~~versicolor~~
6.7	3.3	~~virginica~~

Table 2: Test set

Validation (train/test)

sepal_length	sepal_width	species
5.1	3.5	setosa
4.9	3	setosa
6.7	3.1	versicolor
6.3	2.3	versicolor
6.3	2.8	virginica
6.1	2.6	virginica

Table 1: Train set

sepal_length	sepal_width	~~species~~
4.7	3.2	~~setosa~~
5.6	3	~~versicolor~~
6.7	3.3	~~virginica~~

Table 2: Test set

Goal: to predict the class of the test set data using the train set data.

Evaluation

Confusion matrix

- Matrix for comparing predictions with the true class of objects
- Allows to evaluate the performance of a classification method
- Allows to highlight confusions (errors) between classes

```
[[6  0  0]
 [0  9  0]
 [0  1  4]]
             precision    recall  f1-score   support

     setosa       1.00      1.00      1.00         6
 versicolor       0.90      1.00      0.95         9
  virginica       1.00      0.80      0.89         5

avg / total       0.96      0.95      0.95        20
```

Validation

sepal_length	sepal_width	species
5.1	3.5	setosa
4.7	3.2	setosa
6.7	3.1	versicolor
5.6	3	versicolor
6.3	2.8	virginica
6.7	3.3	virginica

Table 3: Test set

Validation

sepal_length	sepal_width	~~species~~
5.1	3.5	~~setosa~~
4.7	3.2	~~setosa~~
6.7	3.1	~~versicolor~~
5.6	3	~~versicolor~~
6.3	2.8	~~virginica~~
6.7	3.3	~~virginica~~

Table 3: Test set

Validation

sepal_length	sepal_width	~~species~~
5.1	3.5	~~setosa~~
4.7	3.2	~~setosa~~
6.7	3.1	~~versicolor~~
5.6	3	~~versicolor~~
6.3	2.8	~~virginica~~
6.7	3.3	~~virginica~~

Table 3: Test set

predictions
setosa
setosa
versicolor
virginica
virginica
virginica

Table 4: Predictions

Validation

sepal_length	sepal_width	~~species~~
5.1	3.5	~~setosa~~
4.7	3.2	~~setosa~~
6.7	3.1	~~versicolor~~
5.6	3	~~versicolor~~
6.3	2.8	~~virginica~~
6.7	3.3	~~virginica~~

Table 3: Test set

predictions
setosa
setosa
versicolor
virginica
virginica
virginica

Table 4: Predictions

	setosa	versicolor	virginica
setosa	0	0	0
versicolor	0	0	0
virginica	0	0	0

Table 5: Confusion matrix

Validation

sepal_length	sepal_width	species
5.1	3.5	setosa
4.7	3.2	setosa
6.7	3.1	versicolor
5.6	3	versicolor
6.3	2.8	virginica
6.7	3.3	virginica

Table 3: Test set

predictions
setosa
setosa
versicolor
virginica
virginica
virginica

Table 4: Predictions

	setosa	versicolor	virginica
setosa	1	0	0
versicolor	0	0	0
virginica	0	0	0

Table 5: Confusion matrix

Validation

sepal_length	sepal_width	species
5.1	3.5	setosa
4.7	3.2	setosa
6.7	3.1	versicolor
5.6	3	versicolor
6.3	2.8	virginica
6.7	3.3	virginica

Table 3: Test set

predictions
setosa
setosa
versicolor
virginica
virginica
virginica

Table 4: Predictions

	setosa	versicolor	virginica
setosa	2	0	0
versicolor	0	0	0
virginica	0	0	0

Table 5: Confusion matrix

Validation

sepal_length	sepal_width	species
5.1	3.5	setosa
4.7	3.2	setosa
6.7	3.1	versicolor
5.6	3	versicolor
6.3	2.8	virginica
6.7	3.3	virginica

Table 3: Test set

predictions
setosa
setosa
versicolor
virginica
virginica
virginica

Table 4: Predictions

	setosa	versicolor	virginica
setosa	2	0	0
versicolor	0	1	0
virginica	0	0	0

Table 5: Confusion matrix

Validation

sepal_length	sepal_width	species
5.1	3.5	setosa
4.7	3.2	setosa
6.7	3.1	versicolor
5.6	3	versicolor
6.3	2.8	virginica
6.7	3.3	virginica

Table 3: Test set

predictions
setosa
setosa
versicolor
virginica
virginica
virginica

Table 4: Predictions

	setosa	versicolor	virginica
setosa	2	0	0
versicolor	0	1	1
virginica	0	0	0

Table 5: Confusion matrix

Classification errors

Validation

sepal_length	sepal_width	species
5.1	3.5	setosa
4.7	3.2	setosa
6.7	3.1	versicolor
5.6	3	versicolor
6.3	2.8	virginica
6.7	3.3	virginica

Table 3: Test set

predictions
setosa
setosa
versicolor
virginica
virginica
virginica

Table 4: Predictions

	setosa	versicolor	virginica
setosa	2	0	0
versicolor	0	1	1
virginica	0	0	2

Table 5: Confusion matrix

Validation

sepal_length	sepal_width	species
5.1	3.5	setosa
4.7	3.2	setosa
6.7	3.1	versicolor
5.6	3	versicolor
6.3	2.8	virginica
6.7	3.3	virginica

Table 3: Test set

predictions
setosa
setosa
versicolor
virginica
virginica
virginica

Table 4: Predictions

	setosa	versicolor	virginica
setosa	2	0	0
versicolor	0	1	1
virginica	0	0	2

Table 5: Confusion matrix

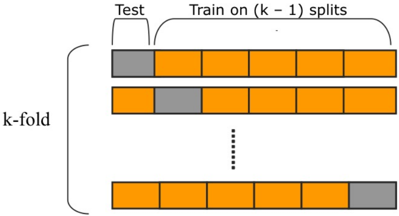
Good predictions (diagonal) = 2 + 1 + 2

Bad predictions (off-diagonal) = 1

Precision = $\frac{2+1+2}{2+1+2+1} = 0.83$

Cross-validation

- Alternative to train / test cutting
- The dataset is divided into K subsets
- K-1 sets used for the train, 1 set used for the test
- Each set is used once for the test



source: `https://raw.githubusercontent.com/qingkaikong/`

Scikit-learn

Scikit-learn

- Scikit-learn contains the majority of the data mining algorithms
- It also offers many tools for the evaluation of models
- Scikit-learn separates the data and the variable to be predicted (*target*)

```
from sklearn.model_selection import train_test_split

df = pd.read_csv('data/iris.csv', header=0)
X = df[['sepal_length','sepal_width','petal_length','petal_width']]
y = df[['species']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Scikit-learn

- Scikit-learn separates the data and the variable to be predicted (*target*)

sepal_length	sepal_width	species
5.1	3.5	setosa
4.9	3	setosa
6.7	3.1	versicolor
6.3	2.3	versicolor
6.3	2.8	virginica
6.1	2.6	virginica

Table 6: Train set

sepal_length	sepal_width	species
4.7	3.2	setosa
5.6	3	versicolor
6.7	3.3	virginica

Table 7: Test set

Scikit-learn

- Scikit-learn separates the data and the variable to be predicted (*target*)

X_train		y_train
sepal_length	sepal_width	species
5.1	3.5	setosa
4.9	3	setosa
6.7	3.1	versicolor
6.3	2.3	versicolor
6.3	2.8	virginica
6.1	2.6	virginica

Table 8: Train set

X_test		y_test
sepal_length	sepal_width	species
4.7	3.2	setosa
5.6	3	versicolor
6.7	3.3	virginica

Table 9: Test set

Scikit-learn

Scikit-learn

- Scikit-learn can also be used to calculate the confusion matrix and evaluate model performance

```python
from sklearn.metrics import confusion_matrix, classification_report,
    precision_score

real = ['setosa','setosa','versicolor','versicolor','virginica','virginica']
pred = ['setosa','setosa','versicolor','virginica','virginica','virginica']

print(confusion_matrix(real, pred))
print(classification_report(real , pred))
print(precision_score(real , pred, average='micro'))
```

```
[[2 0 0]
 [0 1 1]
 [0 0 2]]

0.8333333333333334
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         2
  versicolor       1.00      0.50      0.67         2
   virginica       0.67      1.00      0.80         2

    accuracy                           0.83         6
   macro avg       0.89      0.83      0.82         6
weighted avg       0.89      0.83      0.82         6
```

Exercise

- Using the data visualization of the Iris dataset, create a simple threshold-based classifier to classify the different types of iris
- Evaluate the performance of your classifier on all the data
- Make a graph to illustrate the class boundary

```python
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# data reading
df = pd.read_csv('data/iris.csv', header=0)

# create a threshold classifier
```

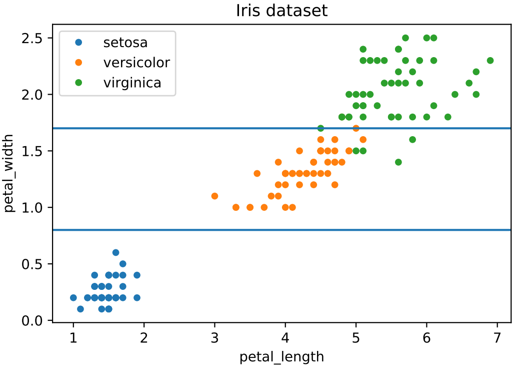
Correction

```python
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# lecture des donnees
df = pd.read_csv('data/iris.csv', header=0)

df['pred'] = 'setosa'
df.loc[df['petal_width'] > 0.8,'pred'] = 'versicolor'
df.loc[df['petal_width'] > 1.7,'pred'] = 'virginica'

print(confusion_matrix(df['pred'] , df['species']))
print(classification_report(df['pred'] , df['species']))
```

Correction

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data/iris.csv', header=0)
groups = df.groupby('species')

fig, ax = plt.subplots()
ax.margins(0.05)
for name, group in groups:
    ax.plot(group.petal_length, group.petal_width, marker='o', linestyle='', ms=4, label=name)
plt.xlabel("petal_length")
plt.ylabel("petal_width")
plt.title("Iris dataset")
plt.axhline(0.8)
plt.axhline(1.7)
ax.legend()
plt.show()
```

Correction

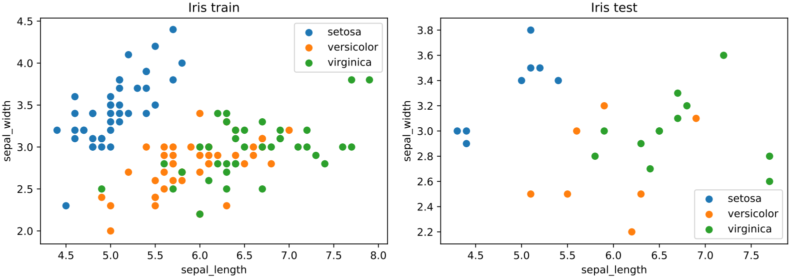

Iris dataset
setosa
versicolor
virginica
petal_width
petal_length
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7

K-nearest neighbors

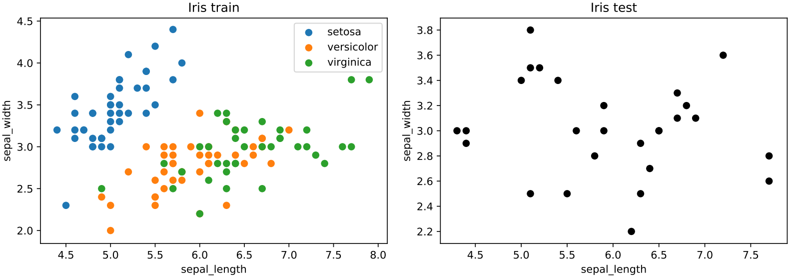
K-nearest neighbors method:

- The 1-nearest-neighbor consists in finding the closest instance A in the available data to instance B to classify. The class of A (which is known) is then assigned to B
- The K-NN (*K-nearest neighbors*) method consists in finding the K nearest neighbors and assigning the majority class to the instance to be classified
- It is qualified as *lazy learning* as there is no model construction step
- It is necessary to have a distance between the instances (not trivial depending on the type of data)

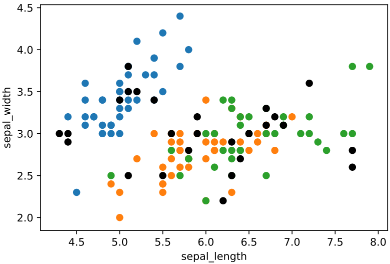
K-nearest neighbors



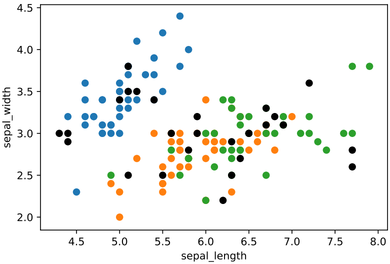
K-nearest neighbors

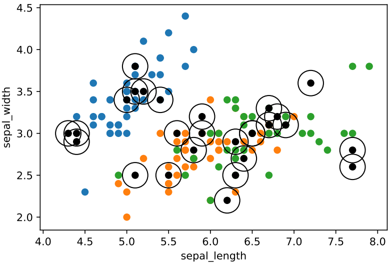


K-nearest neighbors

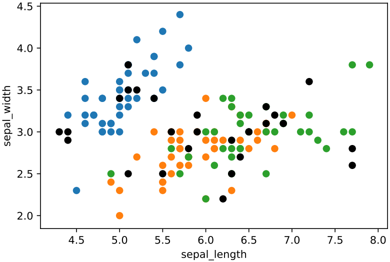


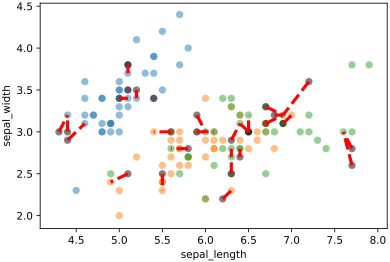
K-nearest neighbors





K-nearest neighbors





K-nearest neighbors method:

- The most commonly used distance between two vectors is the Euclidean distance:

$$d(a, b) = \sqrt{\sum_{i=0}^{n} (a_i - b_i)^2}$$

- There are many distances depending on the type of data
- It is generally necessary to normalize the numeric attributes

Normalization

- Need to normalize characteristics (e.g. age vs. salary)
- Several normalization techniques exist

Example: comparing customers

raw data

Customer	Age	Salary
Marie	24	1800
Bruno	31	2200
Laurent	27	1500

normalized data

Customer	Age	Salary
Marie	0,00	0,43
Bruno	1,00	1,00
Laurent	0,43	0,00

Normalization *MinMax*:

$$z = \frac{x - min(x)}{max(x) - min(x)}$$

Method implementation

```
import math
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

# data reading
df = pd.read_csv('data/iris.csv', header=0)

# Euclidean distance function
def euclidean_distance(a, b):
    sum = 0
    for i in range(a.size-1):
        sum += (a[i]-b[i])**2
    return math.sqrt(sum)

# creation of train / test sets
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:-1],df.iloc
     [:,-1], test_size=0.2)

prediction = np.zeros(X_test.shape[0],dtype='object')
```

Method implementation

```
# nearest neighbor search
for i in range(X_test.shape[0]):
    distMin = np.inf
    indexMin = -1;
    current = X_test.iloc[i,:]
    for j in range(X_train.shape[0]):
        t = X_train.iloc[j,:]
        dist = euclidean_distance(current,t)
        if dist < distMin:
            distMin = dist
            indexMin = j
    prediction[i] = y_train.iloc[indexMin]

# classifier's evaluation
cnf_matrix = confusion_matrix(prediction, y_test)
print(cnf_matrix)
print(classification_report(prediction, y_test))
```

NB: this is an underperforming implementation

Scikit-learn

- Implementation with `scikit-learn`

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# data reading
df = pd.read_csv('data/iris.csv', header=0)

# creation of train / test sets
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:-1],df.iloc
    [:,-1], test_size=0.2)

# classifier creation
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

predictions = knn.predict(X_test)

# classifier's evaluation
print(confusion_matrix(predictions, y_test))
print(classification_report(predictions, y_test))
```

Scikit-learn

- Visualization of the class boundaries

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier

n_classes = 3
plot_colors = "ryb"
plot_step = 0.02
iris = load_iris()
for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):
    X = iris.data[:, pair]
    y = iris.target

    clf = KNeighborsClassifier(n_neighbors=3).fit(X, y)

    plt.subplot(2, 3, pairidx + 1)
    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, plot_step),
                         np.arange(y_min, y_max, plot_step))
    plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)
```

K-nearest neighbors

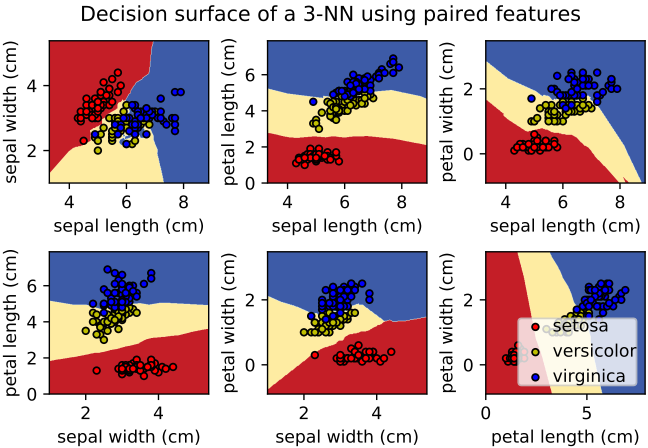
Scikit-learn

- Visualization of the class boundaries

```python
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[1]])

    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                    cmap=plt.cm.RdYlBu, edgecolor='black', s=15)
plt.suptitle("Decision surface of a 3-NN using paired features")
plt.legend(loc='lower right', borderpad=0, handletextpad=0)
plt.axis("tight")
plt.show()
```

Scikit-learn

- Visualization of the class boundaries



Conclusion

Advantages

- Easy to understand
- Explicability
- Allows non-linear class boundaries

Drawbacks

- Requires the definition of a distance between instances
- Classification time increases with data
- Need to find the right K

Decision trees and random forest

Decision tree

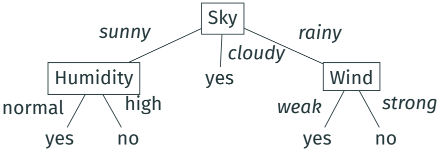
- Tree representation of a classification function
- One of the best known and applied methods in classification
- A whole family of algorithms (e.g. ID3, ID4, C4.5, C5.0)
- Allows processing both numerical and categorical data

Example of data:

Sky	Temperature	Humidity	Wind	Play
sunny	warm	high	weak	no
sunny	warm	high	strong	no
cloudy	warm	high	weak	yes
rainy	gentle	high	weak	yes
rainy	cold	normal	weak	yes
rainy	cold	normal	strong	no
cloudy	cold	normal	strong	yes
sunny	gentle	high	weak	no
sunny	cold	normal	weak	yes
rainy	gentle	normal	weak	yes
sunny	gentle	normal	strong	yes
cloudy	gentle	high	strong	yes
cloudy	warm	normal	weak	yes
rainy	gentle	high	strong	no

Structure

Tree structure



Decision tree

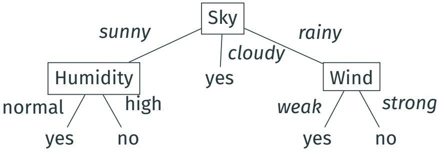
- Constructed from the data
- Nodes: attributes
- Edges: values
- Leafs: decisions (classes)

Classification

- A new instance is tested by its **path** from the root to the leaf

Structure

Tree structure



Sky	Temp.	Humidity	Wind	Class
sunny	warm	high	strong	?

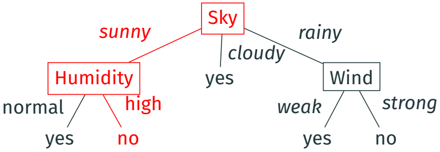
Decision tree

- Constructed from the data
- Nodes: attributes
- Edges: values
- Leafs: decisions (classes)

Classification

- A new instance is tested by its **path** from the root to the leaf

Tree structure



Sky	Temp.	Humidity	Wind	Class
sunny	warm	high	strong	?

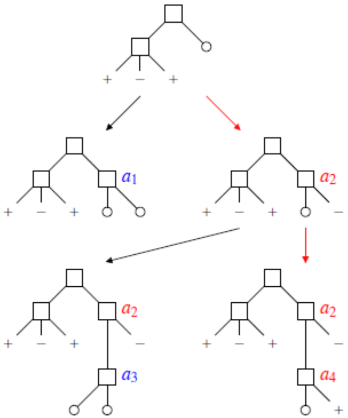
Decision tree

- Constructed from the data
- Nodes: attributes
- Edges: values
- Leafs: decisions (classes)

Classification

- A new instance is tested by its **path** from the root to the leaf

Training algorithm



Strategy

- Extend the structure incrementally until a tree is obtained

Heuristic

- Finding an evaluation function that favours discriminating attributes

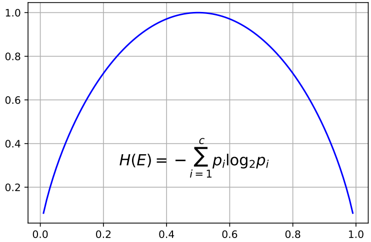
Decision tree algorithm (ID3)

1. it begins with the original set of samples E as the root node
2. on each iteration of the algorithm, it iterates through the unused attributes of the set A and calculates Entropy (H) and Information gain (IG) of this attribute
3. it selects the attribute which has the smallest Entropy or largest Information gain.
4. the set E is then split by the selected attribute to produce a subset of the data
5. the algorithm continues to recur on each subset, considering only attributes never selected before

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$



Entropy

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
no
no
no
no

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

$H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
no
no
no
no

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
no
no
no
no

$$H(E) = -(4/8) * log_2(4/8) - (4/8) * log_2(4/8) = 1$$

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
no
no
no
no

$H(E) = -(4/8) * log_2(4/8) - (4/8) * log_2(4/8) = 1$

Entropy (uncertainty) is maximum

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
yes
yes
yes
yes

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
yes
yes
yes
yes

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
yes
yes
yes
yes

$H(E) = -(8/8) * log_2(8/8) = 0$

Entropy

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
yes
yes
yes
yes

$H(E) = -(8/8) * log_2(8/8) = 0$

Entropy (uncertainty) is zero

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
yes
yes
yes
no

Entropy

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
yes
yes
yes
no

Entropy

- Measurement of an amount of information or uncertainty
- Let p_i the proportion of examples from class i in E:

  $H(E) = -\sum_{i=1}^{c} p_i \log_2 p_i$

Example:

Play
yes
yes
yes
yes
yes
yes
yes
no

$H(E) = -(7/8) * log_2(7/8) - (1/8) * log_2(1/8) \approx 0.54$

Entropy gain

- Decrease of the entropy generated by the partition of a set of examples according to an attribute
- Let E be a set of examples, a an attribute and $V(a)$ the values of a:

  $G(a, E) = H(E) - \sum_{v \in V(a)}^{c} \frac{|E_{a,v}|}{|E|} H(E_{a,v})$

Example:

Sky	Play
sunny	yes
sunny	yes
cloudy	yes
cloudy	yes
sunny	no
sunny	no
cloudy	no
cloudy	no

$\rightarrow$

Sky	Play
sunny	yes
sunny	yes
sunny	no
sunny	no
cloudy	yes
cloudy	yes
cloudy	no
cloudy	no

Entropy gain

Entropy gain

- Decrease of the entropy generated by the partition of a set of examples according to an attribute
- Let E be a set of examples, a an attribute and $V(a)$ the values of a:

$$G(a, E) = H(E) - \sum_{v \in V(a)}^{c} \frac{|E_{a,v}|}{|E|} H(E_{a,v})$$

Example:

Sky	Play
sunny	yes
sunny	yes
cloudy	yes
cloudy	yes
sunny	no
sunny	no
cloudy	no
cloudy	no

$\rightarrow$

Sky	Play
sunny	yes
sunny	yes
sunny	no
sunny	no
cloudy	yes
cloudy	yes
cloudy	no
cloudy	no

Entropy gain

Entropy gain

- Decrease of the entropy generated by the partition of a set of examples according to an attribute
- Let E be a set of examples, a an attribute and $V(a)$ the values of a:

  $G(a, E) = H(E) - \sum_{v \in V(a)}^{c} \frac{|E_{a,v}|}{|E|} H(E_{a,v})$

Example:

Sky	Play
sunny	yes
sunny	yes
cloudy	yes
cloudy	yes
sunny	no
sunny	no
cloudy	no
cloudy	no

$\rightarrow$

Sky	Play
sunny	yes
sunny	yes
sunny	no
sunny	no
cloudy	yes
cloudy	yes
cloudy	no
cloudy	no

$G(Sky, E) = 1 - ((0.5 * 1) + (0.5 * 1)) = 0$

Entropy gain

- Decrease of the entropy generated by the partition of a set of examples according to an attribute
- Let E be a set of examples, a an attribute and $V(a)$ the values of a:

$$G(a, E) = H(E) - \sum_{v \in V(a)}^{c} \frac{|E_{a,v}|}{|E|} H(E_{a,v})$$

Example:

Sky	Play
sunny	yes
sunny	yes
cloudy	yes
cloudy	yes
sunny	no
sunny	no
cloudy	no
cloudy	no

$\rightarrow$

Sky	Play
sunny	yes
sunny	yes
sunny	no
sunny	no
cloudy	yes
cloudy	yes
cloudy	no
cloudy	no

$$G(Sky, E) = 1 - ((0.5 * 1) + (0.5 * 1)) = 0$$

Knowing the "Sky" attribute does not reduce entropy (uncertainty).

Entropy gain

- Decrease of the entropy generated by the partition of a set of examples according to an attribute
- Let E be a set of examples, a an attribute and $V(a)$ the values of a:

  $G(a, E) = H(E) - \sum_{v \in V(a)}^{c} \frac{|E_{a,v}|}{|E|} H(E_{a,v})$

Example:

Sky	Play
sunny	yes
sunny	yes
cloudy	yes
cloudy	yes
cloudy	no
cloudy	no
rainy	no
rainy	no

Entropy gain

- Decrease of the entropy generated by the partition of a set of examples according to an attribute
- Let E be a set of examples, a an attribute and $V(a)$ the values of a:

$$G(a, E) = H(E) - \sum_{v \in V(a)}^{c} \frac{|E_{a,v}|}{|E|} H(E_{a,v})$$

Example:

Sky	Play
sunny	yes
sunny	yes
cloudy	yes
cloudy	yes
cloudy	no
cloudy	no
rainy	no
rainy	no

Entropy gain

- Decrease of the entropy generated by the partition of a set of examples according to an attribute
- Let E be a set of examples, a an attribute and $V(a)$ the values of a:

$$G(a, E) = H(E) - \sum_{v \in V(a)}^{c} \frac{|E_{a,v}|}{|E|} H(E_{a,v})$$

Example:

Sky	Play
sunny	yes
sunny	yes
cloudy	yes
cloudy	yes
cloudy	no
cloudy	no
rainy	no
rainy	no

$$G(Sky, E) = 1 - ((0.25 * 0) + (0.5 * 1) + (0.25 * 0)) = 0.5$$

Entropy gain

- Decrease of the entropy generated by the partition of a set of examples according to an attribute
- Let E be a set of examples, a an attribute and $V(a)$ the values of a:

  $G(a, E) = H(E) - \sum_{v \in V(a)}^{c} \frac{|E_{a,v}|}{|E|} H(E_{a,v})$

Example:

Sky	Play
sunny	yes
sunny	yes
cloudy	yes
cloudy	yes
cloudy	no
cloudy	no
rainy	no
rainy	no

$G(Sky, E) = 1 - ((0.25 * 0) + (0.5 * 1) + (0.25 * 0)) = 0.5$

Knowing the "Sky" attribute reduces entropy (uncertainty), because when there is sun, people always come to play, and don't come when it rains.

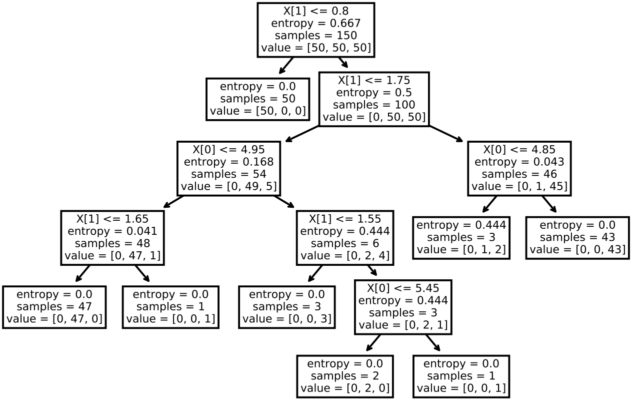
Scikit-learn

- Decision tree implementation in `scikit-learn`

```
import pandas as pd
from sklearn.metrics import  precision_score
from sklearn import tree
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

df = pd.read_csv('data/golf.csv', header=0)

dummies = [pd.get_dummies(df[c]) for c in df.drop('Play golf', axis=1).columns]
binary_data = pd.concat(dummies, axis=1)

X = binary_data.values

le = preprocessing.LabelEncoder()
y = le.fit_transform(df['Play golf'].values)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    stratify=y)

clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
accuracy = precision_score(y_test, y_pred)
print('DecisionTreeClassifier accuracy score: {}'.format(accuracy))
```

Scikit-learn

- Decision tree for the iris dataset



Continuous variables

Sky	Temp.	Humidity	Wind	Decision
Sunny	85	85	Weak	No
Sunny	80	90	Strong	No
Overcast	83	78	Weak	Yes
Rain	70	96	Weak	Yes
Rain	68	80	Weak	Yes
Rain	65	70	Strong	No
Overcast	64	65	Strong	Yes
Sunny	72	95	Weak	No
Sunny	69	70	Weak	Yes
Rain	75	80	Weak	Yes
Sunny	75	70	Strong	Yes
Overcast	72	90	Strong	Yes
Overcast	81	75	Weak	Yes
Rain	71	80	Strong	No

Continuous variable

- Impossible to calculate the gain for each value of a continuous variable:

$$...H(E_{Temp.,85})\ \ H(E_{Temp.,80})\ \ H(E_{Temp.,83})...$$

- Solution: discretization of the variable

  $< 70 = cold$

  $[70 - 75] = gentle$

  $> 75 = warm$

- Find the thresholds before or during the construction of the tree (C4.5)

Scikit-learn

- Visualization of the class boundaries

```python
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

n_classes = 3
plot_colors = "ryb"
plot_step = 0.02
iris = load_iris()
for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):
    X = iris.data[:, pair]
    y = iris.target

    clf = DecisionTreeClassifier().fit(X, y)

    plt.subplot(2, 3, pairidx + 1)
    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, plot_step),
                         np.arange(y_min, y_max, plot_step))
    plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)
```

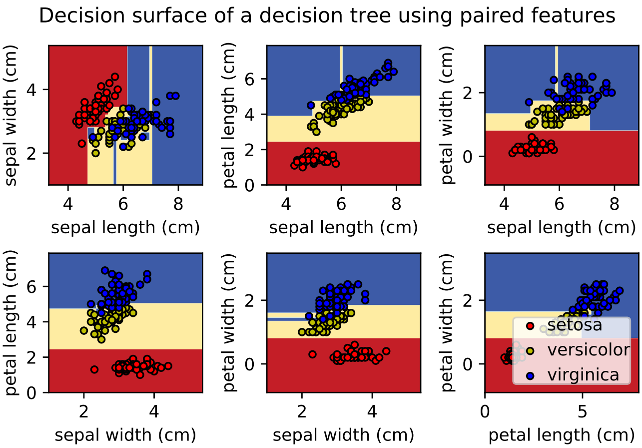
Scikit-learn

- Visualization of the class boundaries

```
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[1]])

    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                    cmap=plt.cm.RdYlBu, edgecolor='black', s=15)
plt.suptitle("Decision surface of a decision tree using paired features")
plt.legend(loc='lower right', borderpad=0, handletextpad=0)
plt.axis("tight")
plt.show()
```

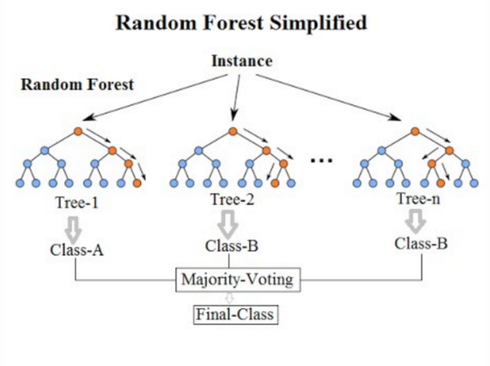
Scikit-learn

- Visualization of the class boundaries



Random forest

- Builds multiple trees with data subsets
- Average tree predictions for the final decision
- Reduces overfitting
- Can also be used to assess the importance of the attributes



source: `https://commons.wikimedia.org/`

Conclusion

Advantages

- Easy to understand
- Relatively short training time
- Very fast classification
- Manages both numerical and categorical data

Drawbacks

- The model can get complicated with a lot of attributes.
- Difficulty for updating the model
- Not always efficient

Support Vector Machine

SVM

- Support Vector Machine (SVM)
- Developed in the 1990s by Vladimir Vapnik
- Notion of "support vectors"
- Use of the concept of maximum margin (distance between the boundary and the nearest samples)

General idea

- Prediction function:
$$y = h(x)$$
- Binary classification:
$$y \in \{-1, 1\}$$
- x is from class 1 if $h(x) \geq 0$ and from class -1 otherwise

General idea

- The easiest case is the case of a linear function, obtained by linear combination of the input vector $x = (\mathbf{x}_1, ..., \mathbf{x}_N)^T$, with a vector of weight $w = (\mathbf{w}_1, ..., \mathbf{w}_N)^T$:
$$h(x) = w^T x + w_0$$
- The goal is to learn the function $h(x)$ (a hyperplane) using the following learning examples:
$$\{(x_1, l_1), (x_2, l_2), \ldots, (x_k, l_k), \ldots, (x_p, l_p)\} \subset \mathbb{R}^N x\{-1, 1\}$$
where l_k are the labels, p is the size of the learning set, N is the size of the input vectors
- If the problem is linearly separable, then we have
$$l_k h(x_k) \geq 0 \quad 1 \leq k \leq p$$
$$l_k(w^T x_k + w_0) \geq 0 \quad 1 \leq k \leq p$$

Maximum margin

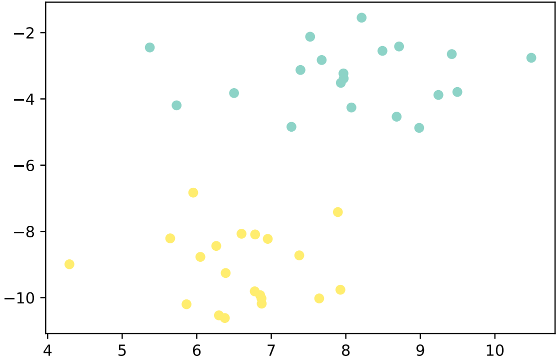
- In a linearly separable case we look for the separator hyperplane
- There are an infinite number of plans that have the same learning performances but different generalization performances
- The margin is the distance between the hyperplane and the nearest samples (called *support vectors*):

$$\arg\max_{w,w_0}\min_{k}\{||x - x_k|| : x \in \mathbb{R}^N, w^T x + w_0 = 0\}$$

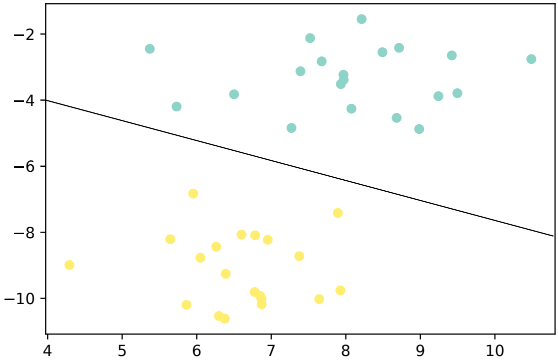
- So we have to find w and w_0 in order to determine the separator hyperplane equation:

$$h(x) = w^T x + w_0 = 0$$

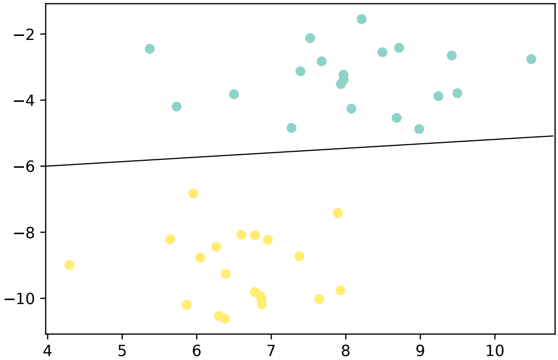
Example with a linearly separable classification problem:



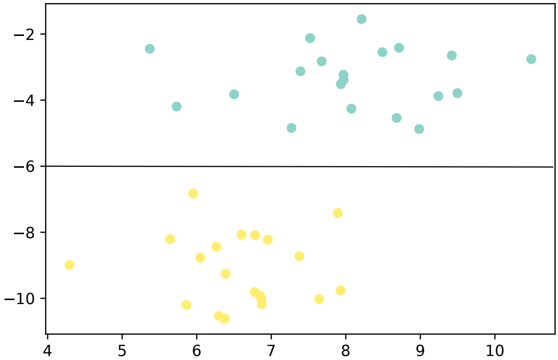
Example with a linearly separable classification problem:



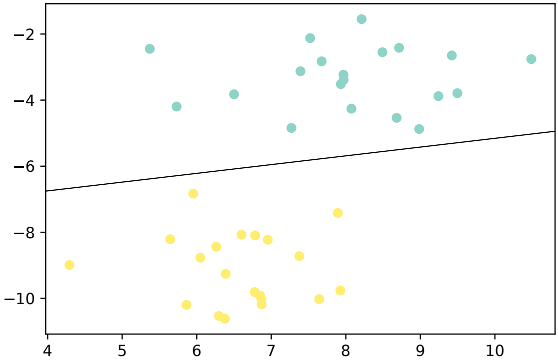
Example with a linearly separable classification problem:



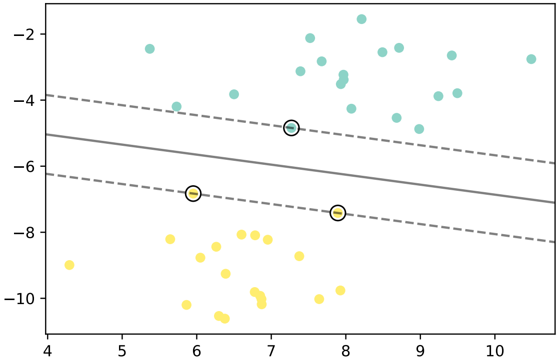
Example with a linearly separable classification problem:



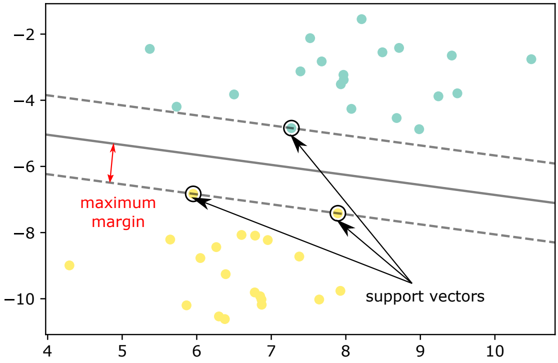
Example with a linearly separable classification problem:



Example with a linearly separable classification problem:



Example with a linearly separable classification problem:



Search for the optimal hyperplane

- The margin is the smallest distance between the training samples and the separating hyperplane that satisfies the separability condition:

$$l_k(w^T x_k + w_0) \geq 0$$

- The distance of a sample x_k from the hyperplane is given by its orthogonal projection on the weight vector:

$$\frac{l_k(w^T x_k + w_0)}{||w||}$$

- The separator hyperplane (w, w_0) of maximum margin is given by:

$$\arg\max_{w, w_0} \left\{ \frac{1}{||w||} \min_k \left[l_k(w^T x_k + w_0) \right] \right\}$$

- The Lagrange multiplier method is used to optimize the values of w and w_0

Non-linearly separable case

- The data is projected in a larger space using a kernel
- A non-linear transformation ϕ is applied to the input vectors x
- The $\phi(X)$ arrival space is called the *feature space*
- In this space, we then look for the hyperplane:

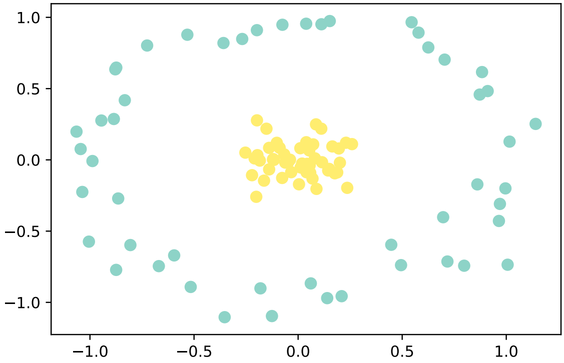
  $$h(x) = w^T \phi(x) + w_0$$

  verifying:

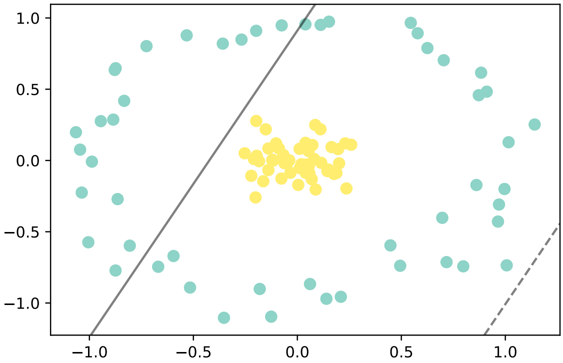
  $$l_k h(x_k) > 0$$

  for all points x_k of the learning set, i.e. the hyperplane separator in the feature space.
- With a "well chosen" kernel the calculation is done in the original space which is less expensive (*kernel trick*)

Example with a non-linearly separable classification problem:

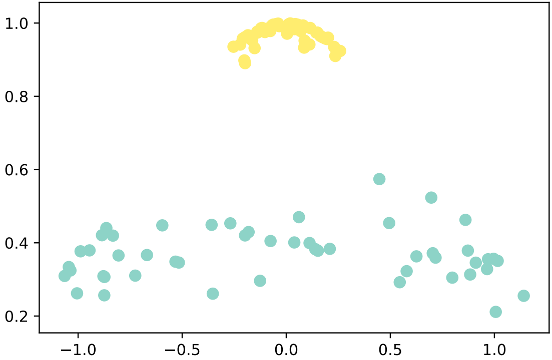


Example with a non-linearly separable classification problem:



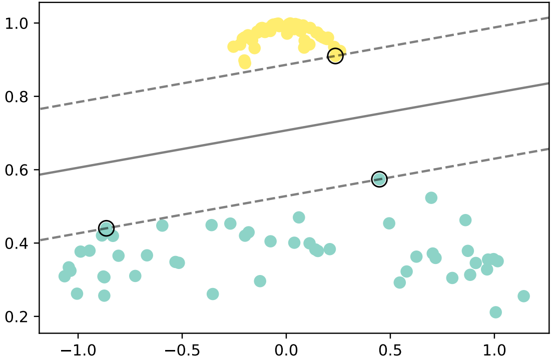
A linear SVM does not work

Example with a non-linearly separable classification problem:



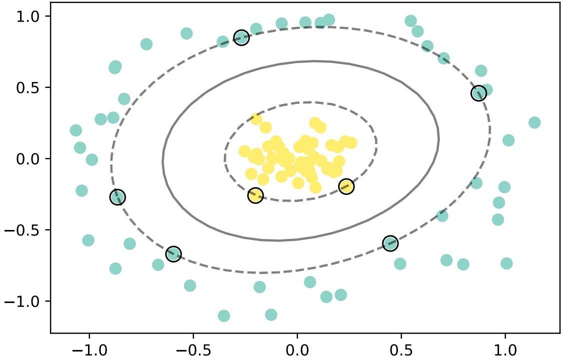
Data transformation, $Y = e^{-(x^2+y^2)}$

Example with a non-linearly separable classification problem:



Applying an SVM in the feature space

Example with a non-linearly separable classification problem:



Boundaries in the original space

Choice of kernel function

- The ϕ transformation must meet certain conditions. It must correspond to a scalar product in a large space.
- The simplest example of a kernel function is the linear kernel:

$$K(x_i, x_j) = x_i^T \cdot x_j$$

- Common kernels used with SVMs are

  - the polynomial kernel:

$$K(x_i, x_j) = (x_i^T \cdot x_j + 1)^d$$

  - the Gaussian kernel:

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right)$$

Scikit-learn

- SVM implementation in `scikit-learn`

```
import pandas as pd
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# data reading
df = pd.read_csv('data/iris.csv', header=0)

# creation of train / test sets
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:-1],df.iloc
    [:,-1], test_size=0.2)

# classifier creation
svm = svm.SVC(kernel='rbf', C=1E6, gamma='auto')
svm.fit(X_train, y_train)

predictions = svm.predict(X_test)

# classifier's evaluation
cnf_matrix = confusion_matrix(predictions, y_test)
print(cnf_matrix)
print(classification_report(predictions, y_test))
```

Scikit-learn

- Visualization of the class boundaries

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn import svm

n_classes = 3
plot_colors = "ryb"
plot_step = 0.02
iris = load_iris()
for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):
    X = iris.data[:, pair]
    y = iris.target

    clf = svm.SVC(kernel='rbf').fit(X, y)

    plt.subplot(2, 3, pairidx + 1)
    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, plot_step),
                         np.arange(y_min, y_max, plot_step))
    plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)
```

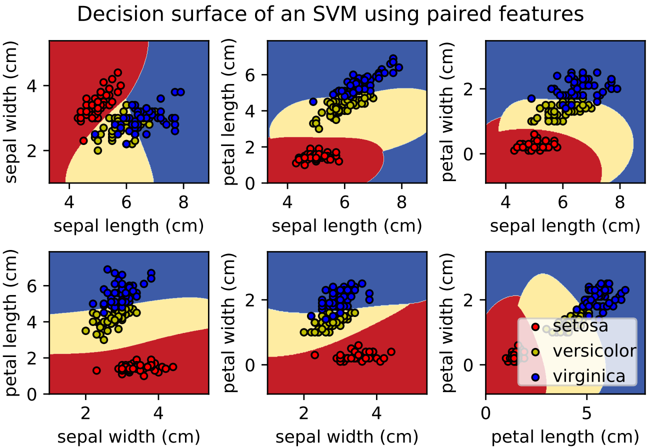
Scikit-learn

- Visualization of the class boundaries

```
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[1]])

    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                    cmap=plt.cm.RdYlBu, edgecolor='black', s=15)
plt.suptitle("Decision surface of a GaussianNB using paired features")
plt.legend(loc='lower right', borderpad=0, handletextpad=0)
plt.axis("tight")
plt.show()
```

Scikit-learn

- Visualization of the class boundaries



Conclusion

Advantages

- Ability to handle large dimensions
- Resolution of non-linear problems with the use of kernels
- Robust against outliers

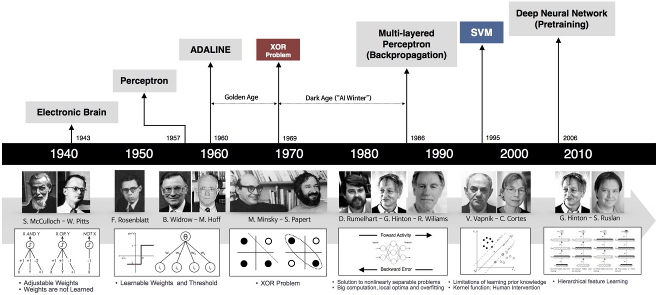
Drawbacks

- Difficulty in identifying the right parameter values
- Problem when classes are too noisy (multiplication of support points)
- Difficulty of interpretation (relevance of variables)

Neural Networks

History

- 1960s: Rosenblatt programmed a perceptron (binary classifier)
- 1969: Minsky and Papert showed the problem of non-linear decisions
- 1970s: *AI Winter*
- 1974: Werbos during his thesis proposed the retropropagation algo
- 1986: Rumelhart, Hinton and Williams rediscovered the algorithm...
- 1990s: Convolutional neural networks by Yann LeCun



source: `https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html`

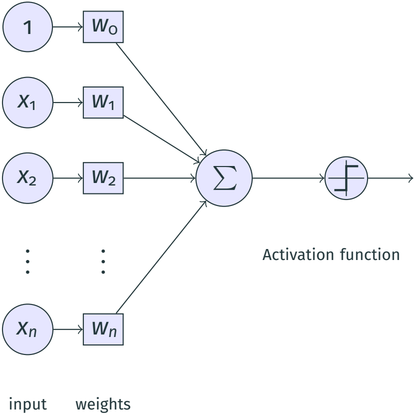
The perceptron

Perceptron

- A linear perceptron takes as input n values $x_1, \ldots, x_n$ (with an input $x_0 = 1$) and computes an output o
- The parameters to be learned are the weights $w_0, \ldots, w_n$
- The output o consists of calculating $\sum_i w_i x_i$ and applying an activation function to it
- The activation function in the case of the perceptron is the Heaviside function defined by:

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

Perceptron:


1
w_0
x_1
w_1
x_2
w_2
x_n
w_n
Σ
Activation function
input
weights

The "classical" training algorithm

1. The perceptron weights are initialized to a random value
2. Each time a new example is presented, the weights are adjusted according to whether or not the perceptron has correctly classified it
3. The algorithm stops when all examples have been presented without modifying any weights

Error correction algorithm

Input : A set of examples $x_1, \ldots, x_n$ and the expected outputs c

Random initialisation of the weights w_i for i between 0 and n

Repeat

Choose an example

Calculate the output o of the perceptron for the example

- - Updating weights - -

For i from 0 to n

$w_i = w_i + (c - o)x_i$

endFor

endRepeat

Output : a perceptron defined by $(w_0, w_1, ..., w_n)$

Error correction algorithm

Input : A set of examples $x_1, \ldots, x_n$ and the expected outputs c

Random initialisation of the weights w_i for i between 0 and n

Repeat

Choose an example

Calculate the output o of the perceptron for the example

- - Updating weights - -

For i from 0 to n

$w_i = w_i + (c - o)x_i$

endFor

endRepeat

Output : a perceptron defined by $(w_0, w_1, ..., w_n)$

Gradient descent method

General case

- Learn any function (not necessarily binary)
- For example, a linear function: $y = w * x + b$
- Learn w and b from examples
- Need to evaluate the error given a value for w and b:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w * x_i + b))^2 \tag{1}$$

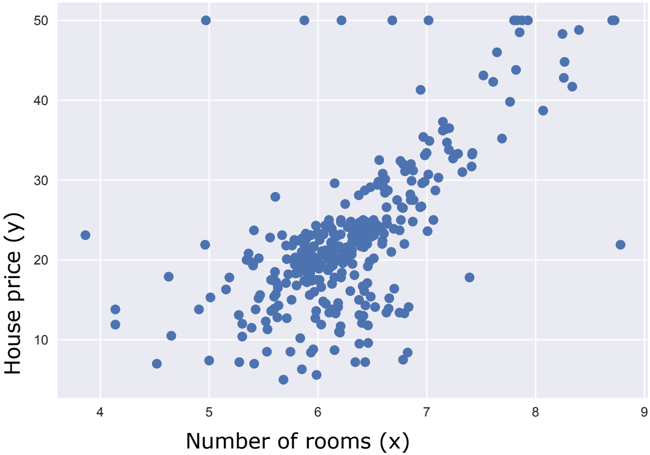
- The goal is to find the values of w and b that minimize L

Neural networks steps

1. Define a model (e.g. linear function)
2. Define an objective (e.g. cost function)
3. Minimize the objective function

Example: Boston House Prices

Predict *y* based on *x*.



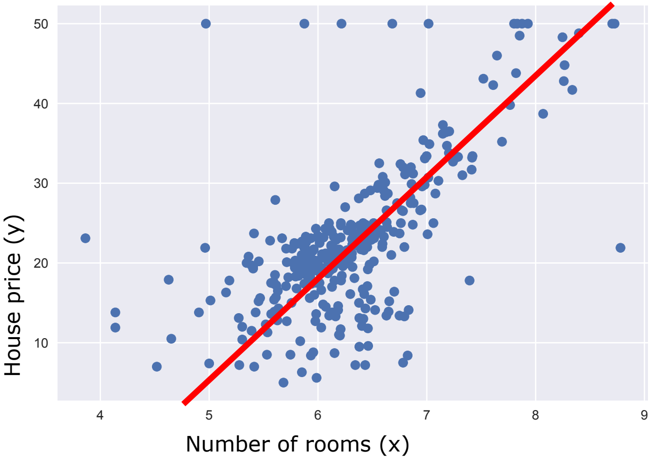
Define a model

A hypothesis:

$$\hat{y} = h(x) \tag{2}$$

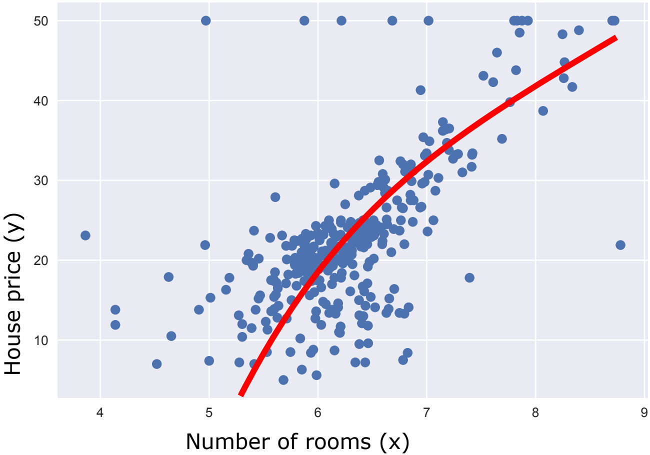
Hypothesis: line

$$\hat{y} = h(x) = w * x + b$$



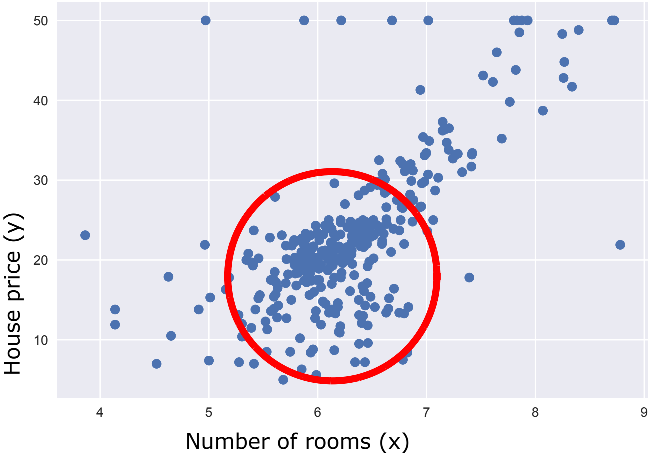
Hypothesis: polynom

$$\hat{y} = h(x) = w_1 * x^2 + w_2 * x + b$$



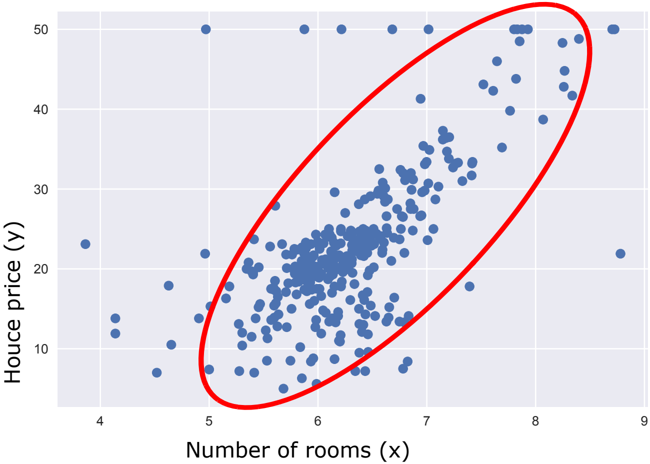
Hypothesis: circle

$$\hat{y} = h(x) = + - \sqrt{x^2 - b^2}$$



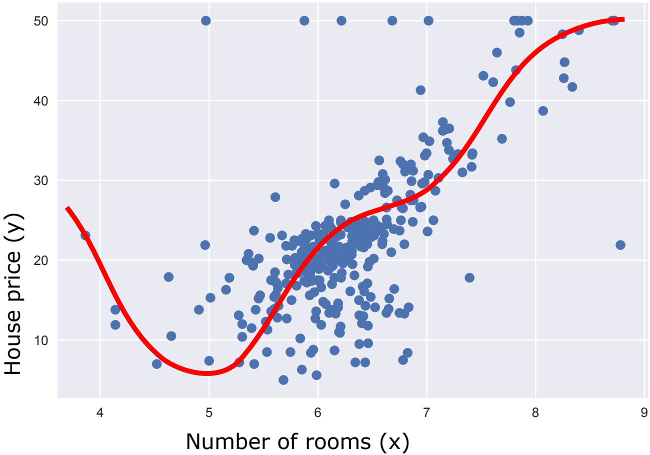
Hypothis: ellipse

$$\hat{y} = h(x) = + - \sqrt{1 - w_1 * x^2}$$



Hypothesis: complex function

$$\hat{y} = h(x) = f(x, w_1, w_2, \ldots, w_n)$$



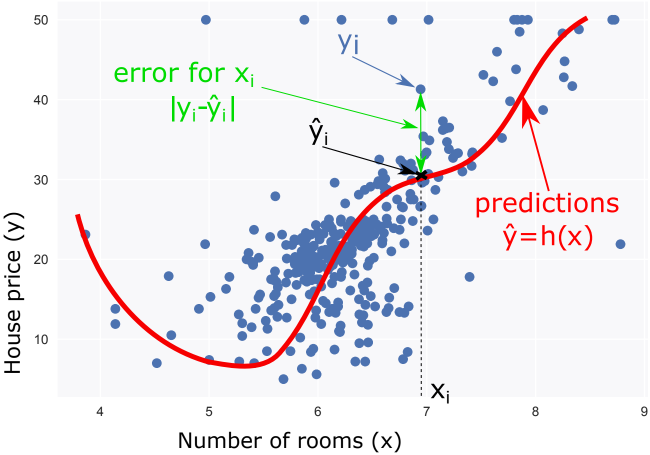
Cost function:

$$L(h) \tag{3}$$

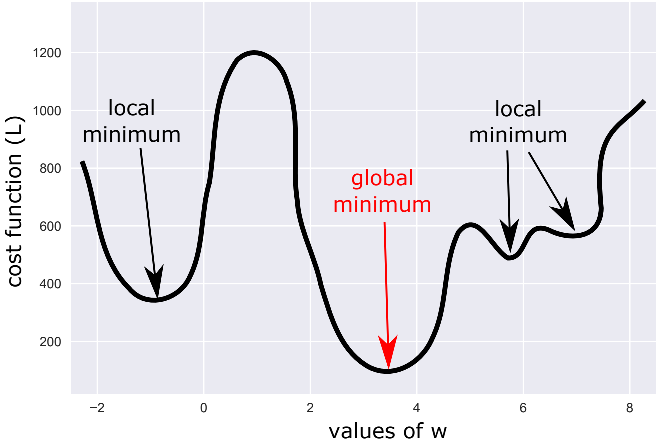
Goal:

$$\min_h L(h) \tag{4}$$

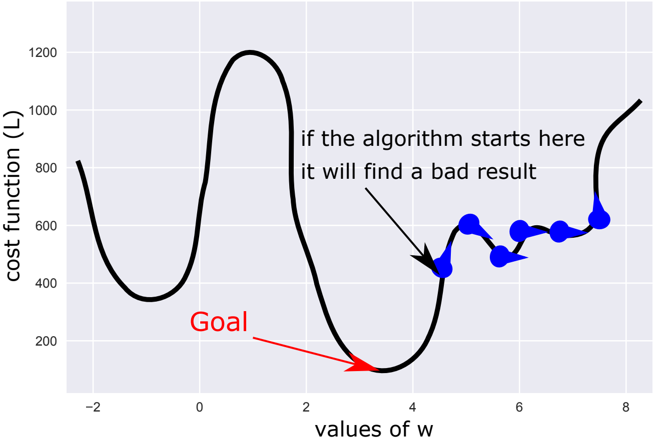
Definition of the objective is independent to the model



Minimizing a cost function



Brute force method

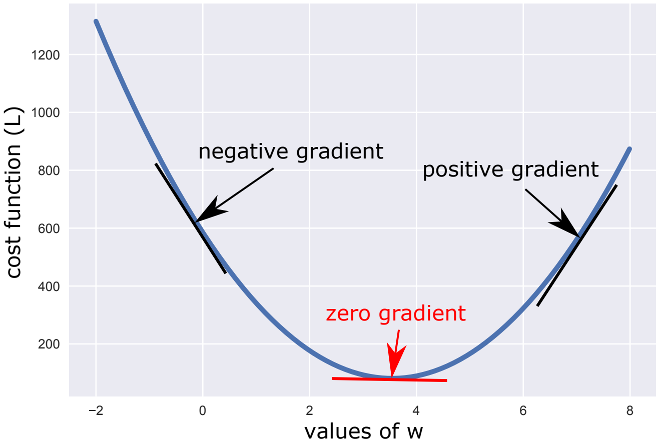


Brute force method

Do not apply brute force because:

- Very slow and costly method
- It's hard to know which w to start with
- You can hit a bad minimum

Gradient method



Optimal solution

Goal: Find w for

$$\frac{\partial L}{\partial w} = 0 \tag{5}$$

Several methods exist:

- Invert the normal equations
- Orthogonal decomposition

Limitations:

- Very slow methods, especially if the model is complex
- Problem if more than one solution exists for Eq. (5)

$\Longrightarrow$ **Gradient descent method**

Optimal solution

Goal: Find w for

$$\frac{\partial L}{\partial w} = 0 \tag{5}$$

Several methods exist:

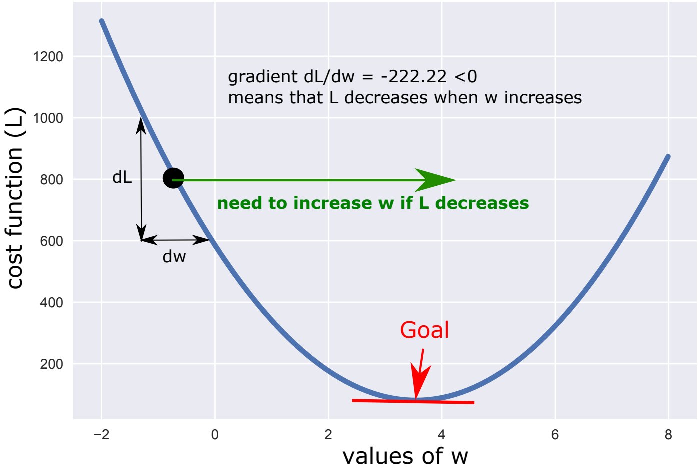
- Invert the normal equations
- Orthogonal decomposition

Limitations:

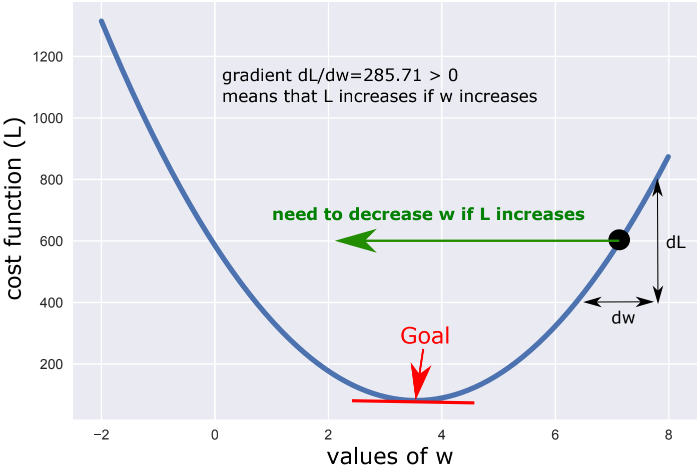
- Very slow methods, especially if the model is complex
- Problem if more than one solution exists for Eq. (5)

$\Longrightarrow$ **Gradient descent method**

Cost decreases



Cost increases

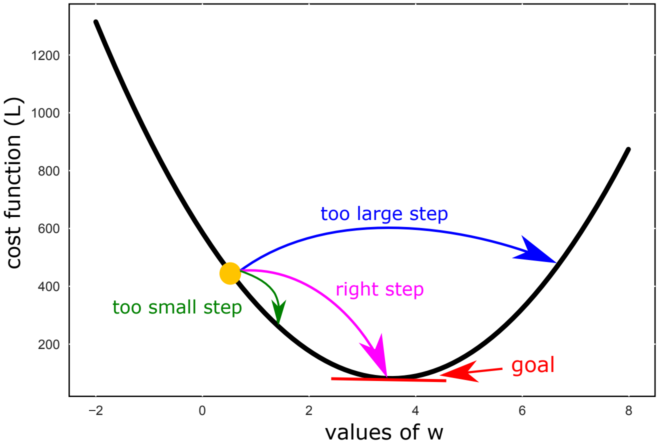


Weigths updating

$$w = w - \alpha \frac{\partial L}{\partial w} \tag{6}$$

- w: all the parameters to be learned (weights)
- L: the cost function to minimize
- α: the learning rate controlling the update of w
- "$-$" : (the minus sign) means that w is updated in the opposite direction of the variation of $L \implies$ the error is minimized

Why do we need α the learning rate?



Good choice of α

How to chose α?

- A very large α can cause the missing of the target
- A very small α can lead to too slow convergence requiring too many iterations

$\Longrightarrow$ You have to choose α to balance the available computing time with the precision of the minimum you want to achieve

Good choice of α

How to chose α?

- A very large α can cause the missing of the target
- A very small α can lead to too slow convergence requiring too many iterations

$\Longrightarrow$ You have to choose α to balance the available computing time with the precision of the minimum you want to achieve

Linear regression

General formulation:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} error(y_i, \hat{y}_i) \tag{7}$$

The root mean square error:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i(w))^2 \tag{8}$$

The root mean square error for linear regression:

$$L(w, b) = \frac{1}{n}\sum_{i=1}^{n} (y_i - (w * x_i + b))^2 \tag{9}$$

For linear regression: two parameters to learn w **and** b

General formulation:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} error(y_i, \hat{y}_i) \tag{7}$$

The root mean square error:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i(w))^2 \tag{8}$$

The root mean square error for linear regression:

$$L(w, b) = \frac{1}{n} \sum_{i=1}^{n} (y_i - (w * x_i + b))^2 \tag{9}$$

For linear regression: two parameters to learn w **and** b

General formulation:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} error(y_i, \hat{y}_i) \tag{7}$$

The root mean square error:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i(w))^2 \tag{8}$$

The root mean square error for linear regression:

$$L(w, b) = \frac{1}{n}\sum_{i=1}^{n} (y_i - (w * x_i + b))^2 \tag{9}$$

For linear regression: two parameters to learn w and b

General formulation:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} error(y_i, \hat{y}_i) \tag{7}$$

The root mean square error:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i(w))^2 \tag{8}$$

The root mean square error for linear regression:

$$L(w,b) = \frac{1}{n}\sum_{i=1}^{n} (y_i - (w * x_i + b))^2 \tag{9}$$

For linear regression: two parameters to learn w **and** b

Multi-variable linear regression

If the instance has several attributes:

- Example: predict the price of a house based on the number of rooms, the population of the neighborhood, the climate of the region, etc.
- Instead of a single w we will have $W = (w_1, w_2, \ldots, w_r)$ for r different attributes
- The same process is repeated for each $w_i \in W$
- The model becomes: $\hat{y}_i = b + \sum_{j=1}^{r} w_j * x_{i,j}$
- With $x_{i,j}$ the j^{th} attribute of the i^{th} instance of the training set X

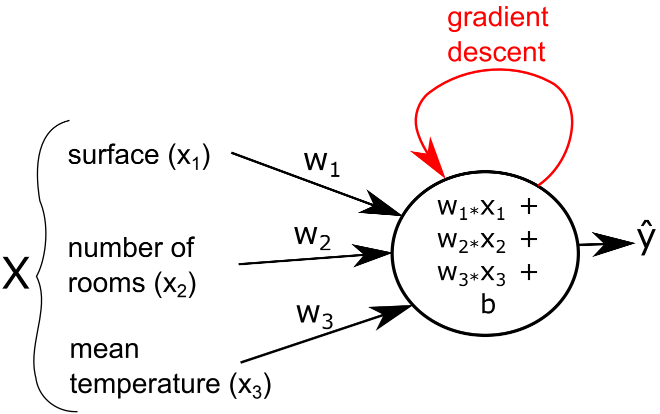
If the instance has several attributes:

- Example: predict the price of a house based on the number of rooms, the population of the neighborhood, the climate of the region, etc.
- Instead of a single w we will have $W = (w_1, w_2, \ldots, w_r)$ for r different attributes
- The same process is repeated for each $w_i \in W$
- The model becomes: $\hat{y}_i = b + \sum_{j=1}^{r} w_j * x_{i,j}$
- With $x_{i,j}$ the j^{th} attribute of the i^{th} instance of the training set X

If the instance has several attributes:

- Example: predict the price of a house based on the number of rooms, the population of the neighborhood, the climate of the region, etc.
- Instead of a single w we will have $W = (w_1, w_2, \ldots, w_r)$ for r different attributes
- The same process is repeated for each $w_i \in W$
- The model becomes: $\hat{y}_i = b + \sum_{j=1}^{r} w_j * x_{i,j}$
- With $x_{i,j}$ the j^{th} attribute of the i^{th} instance of the training set X

Example of linear regression with a neuron (perceptron)



Normalization of input data:

- Consists of setting each attribute between 0 and 1
- This is important to ensure that all attributes contribute equally
- For example:
  - The surface area of a house is between 0 and 1000 m^2
  - The number of rooms is between 0 and 10

Without normalizing $\implies$ a small variation in the number of rooms does not have a big effect compared to the large values of the surface area $\implies$ the model will have difficulties to learn

$$X_{:,i} = \frac{X_{:,i} - min(X_{:,i})}{max(X_{:,i}) - min(X_{:,i})} \; \forall i \in [1, \ldots, r] \tag{10}$$

r being the number of attributes in the dataset

Normalization of input data:

- Consists of setting each attribute between 0 and 1
- This is important to ensure that all attributes contribute equally
- For example:
  - The surface area of a house is between 0 and 1000 m^2
  - The number of rooms is between 0 and 10

Without normalizing $\implies$ a small variation in the number of rooms does not have a big effect compared to the large values of the surface area $\implies$ the model will have difficulties to learn

$$X_{:,i} = \frac{X_{:,i} - min(X_{:,i})}{max(X_{:,i}) - min(X_{:,i})} \; \forall i \in [1, \ldots, r] \tag{10}$$

r being the number of attributes in the dataset

Normalization of input data:

- Consists of setting each attribute between 0 and 1
- This is important to ensure that all attributes contribute equally
- For example:
  - The surface area of a house is between 0 and 1000 m^2
  - The number of rooms is between 0 and 10

Without normalizing $\implies$ a small variation in the number of rooms does not have a big effect compared to the large values of the surface area $\implies$ the model will have difficulties to learn

$$X_{:,i} = \frac{X_{:,i} - min(X_{:,i})}{max(X_{:,i}) - min(X_{:,i})} \; \forall i \in [1, \dots, r] \tag{10}$$

r being the number of attributes in the dataset

Normalization of input data:

- Consists of setting each attribute between 0 and 1
- This is important to ensure that all attributes contribute equally
- For example:
  - The surface area of a house is between 0 and 1000 m^2
  - The number of rooms is between 0 and 10

Without normalizing $\implies$ a small variation in the number of rooms does not have a big effect compared to the large values of the surface area $\implies$ the model will have difficulties to learn

$$X_{:,i} = \frac{X_{:,i} - min(X_{:,i})}{max(X_{:,i}) - min(X_{:,i})} \; \forall i \in [1, \ldots, r] \tag{10}$$

r being the number of attributes in the dataset

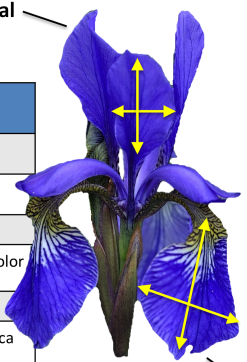
Classification with neural nets

Example of binary classification

Samples
(instances, observations)

Petal

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			…		
50	6.4	3.5	4.5	1.2	Versicolor
			…		
150	5.9	3.0	5.0	1.8	Virginica



Sepal

Class labels
(targets)

Features
(attributes, measurements, dimensions)

source: https://rpubs.com/wjholst/322258

Binary problem: setosa or not?

Category	SL (Sepal Length)	SW (Sepal Width)	PL (Petal Length)	PW (Petal Width)
Setosa (0)	5.1	3.5	1.4	0.2
Versicolor (1)	6.4	3.5	4.5	1.2
Virginica (2)	5.9	3.0	5.0	1.8
⋮	⋮	⋮	⋮	⋮

Table 10: Iris data overview

Category	SL (Sepal Length)	SW (Sepal Width)	PL (Petal Length)	PW (Petal Width)
Setosa (0)	5.1	3.5	1.4	0.2
Non-Setosa (1)	6.4	3.5	4.5	1.2
Non-Setosa (1)	5.9	3.0	5.0	1.8
⋮	⋮	⋮	⋮	⋮

Table 11: Iris data for binary classification

Binary problem: setosa or not?

Category	SL (Sepal Length)	SW (Sepal Width)	PL (Petal Length)	PW (Petal Width)
Setosa (0)	5.1	3.5	1.4	0.2
Versicolor (1)	6.4	3.5	4.5	1.2
Virginica (2)	5.9	3.0	5.0	1.8
⋮	⋮	⋮	⋮	⋮

Table 10: Iris data overview

Category	SL (Sepal Length)	SW (Sepal Width)	PL (Petal Length)	PW (Petal Width)
Setosa (0)	5.1	3.5	1.4	0.2
Non-Setosa (1)	6.4	3.5	4.5	1.2
Non-Setosa (1)	5.9	3.0	5.0	1.8
⋮	⋮	⋮	⋮	⋮

Table 11: Iris data for binary classification

Data pre-processing:

- Normalize the data so that everything is between 0 and 1
- Transforming the categories: (string) $\implies$ (integer)
  - Iris-Setosa $\implies$ 0
  - Iris-Versicolor (non-Setosa) $\implies$ 1
  - Iris-Virginica (non-Setosa) $\implies$ 1
- Divide the dataset into train and test subsets

Data pre-processing:

- Normalize the data so that everything is between 0 and 1
- Transforming the categories: (string) $\implies$ (integer)
  - Iris-Setosa $\implies$ 0
  - Iris-Versicolor (non-Setosa) $\implies$ 1
  - Iris-Virginica (non-Setosa) $\implies$ 1
- Divide the dataset into train and test subsets

Data pre-processing:

- Normalize the data so that everything is between 0 and 1
- Transforming the categories: (string) $\Longrightarrow$ (integer)
  - Iris-Setosa $\Longrightarrow$ 0
  - Iris-Versicolor (non-Setosa) $\Longrightarrow$ 1
  - Iris-Virginica (non-Setosa) $\Longrightarrow$ 1
- Divide the dataset into train and test subsets

Different steps:

1. Define a model (hypothesis)
2. Define an objective (cost function)
3. Minimize the cost function using gradient descent method

First step: define the model according to the data

Y (Category)	X_1 (Sepal Length)	X_2 (Sepal Width)	X_3 (Petal Length)	X_4 (Petal Width)
0 (Setosa)	5.1	3.5	1.4	0.2
1 (non-Setosa)	6.4	3.5	4.5	1.2
1 (non-Setosa)	5.9	3.0	5.0	1.8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Goal: predict category Y given $X = (X_1, X_2, X_3, X_4)$

- We know that we want to predict either 0 (Setosa) or 1 (Non-Setosa)
- Probabilistic: Predict $\hat{Y}$ the probability that class is 0 or 1
- Probability $\implies$ real value between 0 and 1
- Choose a hypothesis that satisfies the condition $0 \leq \hat{y} = h(x) \leq 1$
- **Problem:** the linear regression does not satisfy this condition

First step: define the model according to the data

Y (Category)	X_1 (Sepal Length)	X_2 (Sepal Width)	X_3 (Petal Length)	X_4 (Petal Width)
0 (Setosa)	5.1	3.5	1.4	0.2
1 (non-Setosa)	6.4	3.5	4.5	1.2
1 (non-Setosa)	5.9	3.0	5.0	1.8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Goal: predict category Y given $X = (X_1, X_2, X_3, X_4)$

- We know that we want to predict either 0 (Setosa) or 1 (Non-Setosa)
- Probabilistic: Predict $\hat{Y}$ the probability that class is 0 or 1
- Probability $\implies$ real value between 0 and 1
- Choose a hypothesis that satisfies the condition $0 \leq \hat{y} = h(x) \leq 1$
- **Problem:** the linear regression does not satisfy this condition

First step: define the model according to the data

Y (Category)	X_1 (Sepal Length)	X_2 (Sepal Width)	X_3 (Petal Length)	X_4 (Petal Width)
0 (Setosa)	5.1	3.5	1.4	0.2
1 (non-Setosa)	6.4	3.5	4.5	1.2
1 (non-Setosa)	5.9	3.0	5.0	1.8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Goal: predict category Y given $X = (X_1, X_2, X_3, X_4)$

- We know that we want to predict either 0 (Setosa) or 1 (Non-Setosa)
- Probabilistic: Predict $\hat{Y}$ the probability that class is 0 or 1
- Probability $\implies$ real value between 0 and 1
- Choose a hypothesis that satisfies the condition $0 \leq \hat{y} = h(x) \leq 1$
- **Problem:** the linear regression does not satisfy this condition

First step: define the model according to the data

Y (Category)	X_1 (Sepal Length)	X_2 (Sepal Width)	X_3 (Petal Length)	X_4 (Petal Width)
0 (Setosa)	5.1	3.5	1.4	0.2
1 (non-Setosa)	6.4	3.5	4.5	1.2
1 (non-Setosa)	5.9	3.0	5.0	1.8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Goal: predict category Y given $X = (X_1, X_2, X_3, X_4)$

- We know that we want to predict either 0 (Setosa) or 1 (Non-Setosa)
- Probabilistic: Predict $\hat{Y}$ the probability that class is 0 or 1
- Probability $\implies$ real value between 0 and 1
- Choose a hypothesis that satisfies the condition $0 \leq \hat{y} = h(x) \leq 1$
- **Problem:** the linear regression does not satisfy this condition

First step: define the model according to the data

Y (Category)	X_1 (Sepal Length)	X_2 (Sepal Width)	X_3 (Petal Length)	X_4 (Petal Width)
0 (Setosa)	5.1	3.5	1.4	0.2
1 (non-Setosa)	6.4	3.5	4.5	1.2
1 (non-Setosa)	5.9	3.0	5.0	1.8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Goal: predict category Y given $X = (X_1, X_2, X_3, X_4)$

- We know that we want to predict either 0 (Setosa) or 1 (Non-Setosa)
- Probabilistic: Predict $\hat{Y}$ the probability that class is 0 or 1
- Probability $\implies$ real value between 0 and 1
- Choose a hypothesis that satisfies the condition $0 \leq \hat{y} = h(x) \leq 1$
- **Problem:** the linear regression does not satisfy this condition

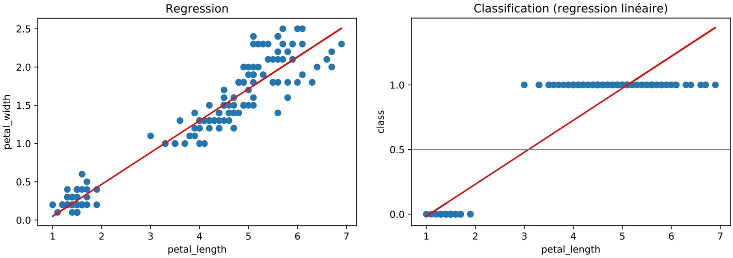
First step: define the model according to the data

Y (Category)	X_1 (Sepal Length)	X_2 (Sepal Width)	X_3 (Petal Length)	X_4 (Petal Width)
0 (Setosa)	5.1	3.5	1.4	0.2
1 (non-Setosa)	6.4	3.5	4.5	1.2
1 (non-Setosa)	5.9	3.0	5.0	1.8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Goal: predict category Y given $X = (X_1, X_2, X_3, X_4)$

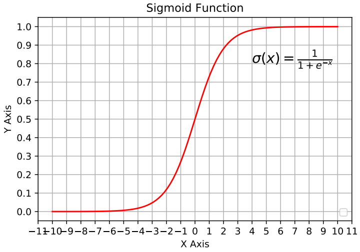
- We know that we want to predict either 0 (Setosa) or 1 (Non-Setosa)
- Probabilistic: Predict $\hat{Y}$ the probability that class is 0 or 1
- Probability $\implies$ real value between 0 and 1
- Choose a hypothesis that satisfies the condition $0 \leq \hat{y} = h(x) \leq 1$
- **Problem:** the linear regression does not satisfy this condition

Difference between regression and classification with linear regression:

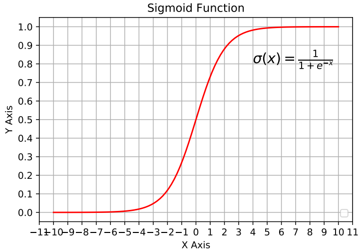


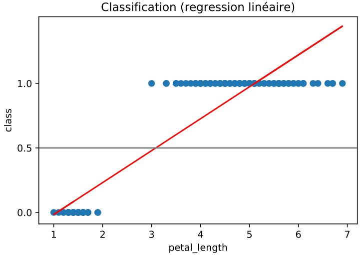
First step: define the model according to the data

Logistic function (sigmoid) : $\sigma(z) = \frac{1}{1+e^{-z}}$



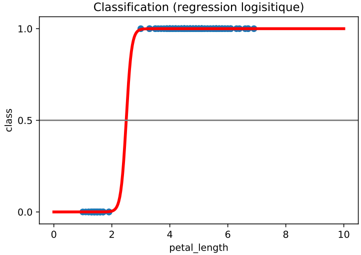
Logistic function (sigmoid) : $\sigma(z) = \frac{1}{1+e^{-z}}$

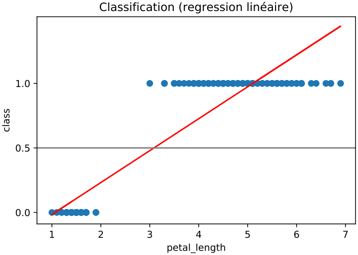


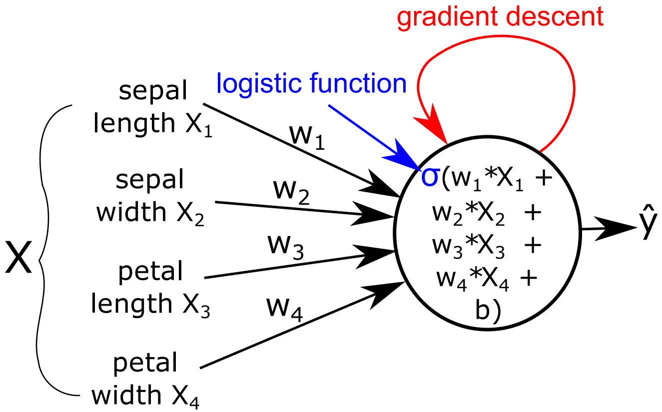


First step: define the model according to the data

Logistic function (sigmoid) : $\sigma(z) = \frac{1}{1+e^{-z}}$





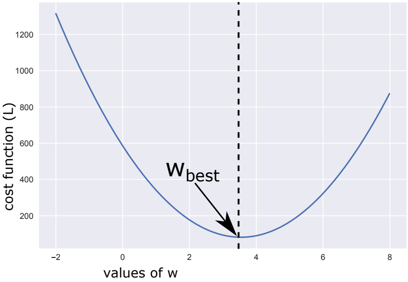

gradient descent
logistic function
sepal length X_1
sepal width X_2
petal length X_3
petal width X_4
X
w_1
w_2
w_3
w_4
$\sigma(w_1*X_1 + w_2*X_2 + w_3*X_3 + w_4*X_4 + b)$
$\hat{y}$

Defining our objective: mean square error?

For linear regression:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i(w))^2 \quad | \quad \hat{y}_i = W * X_i + b \tag{11}$$

The shape of the curve is convex in this case:

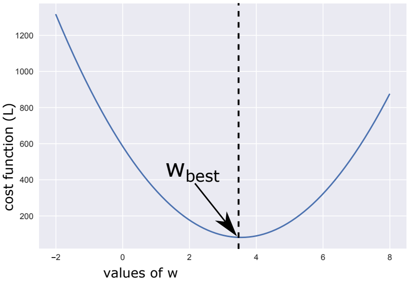


$\Longrightarrow$ **it is easy to find the minimum with the gradient descent method**

For linear regression:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i(w))^2 \quad | \quad \hat{y}_i = W * X_i + b \tag{11}$$

The shape of the curve is convex in this case:



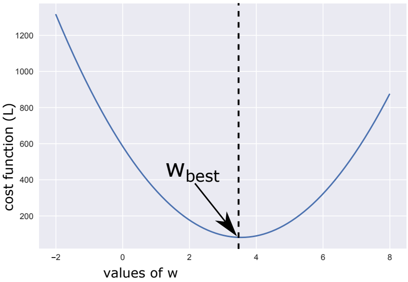
$\Longrightarrow$ **it is easy to find the minimum with the gradient descent method**

Defining our objective: mean square error?

For linear regression:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i(w))^2 \quad | \quad \hat{y}_i = W * X_i + b \tag{11}$$

The shape of the curve is convex in this case:



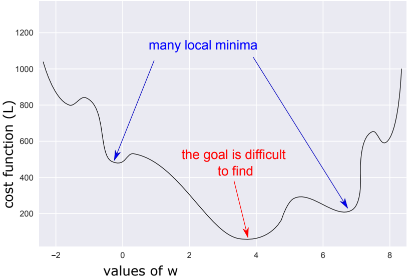
$\Longrightarrow$ **it is easy to find the minimum with the gradient descent method**

Defining our objective: mean square error?

The root mean square error for the logistic regression yields:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i(w))^2 \quad | \quad \hat{y}_i = \sigma(W * X_i + b) \tag{12}$$

σ being non-linear we have a non-convex cost function:



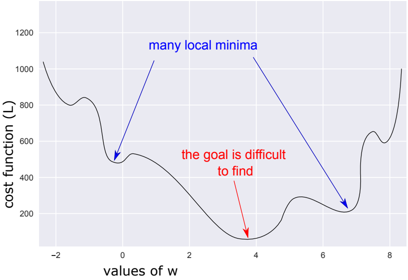
$\Longrightarrow$ **it is necessary to define a new cost function that is convex**

Defining our objective: mean square error?

The root mean square error for the logistic regression yields:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i(w))^2 \quad | \quad \hat{y}_i = \sigma(W * X_i + b) \tag{12}$$

σ being non-linear we have a non-convex cost function:



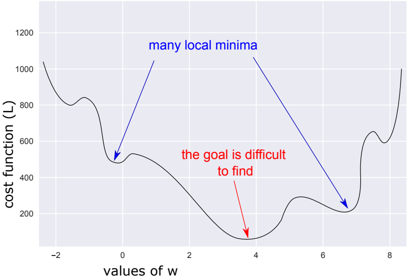
$\Longrightarrow$ **it is necessary to define a new cost function that is convex**

Defining our objective: mean square error?

The root mean square error for the logistic regression yields:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i(w))^2 \quad | \quad \hat{y}_i = \sigma(W * X_i + b) \tag{12}$$

σ being non-linear we have a non-convex cost function:



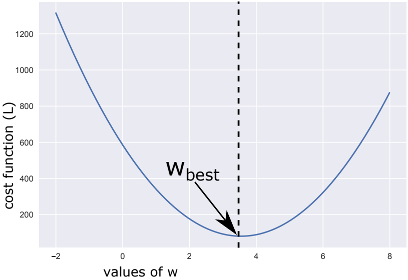
$\Longrightarrow$ **it is necessary to define a new cost function that is convex**

Defining our objective: cross-entropy

Cross-entropy is a good cost function:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} L_i(w) \quad | \quad L_i(w) = -\sum_{c} y_{i_c} * \log(\hat{y}_{i_c}) \tag{13}$$

For a binary classification we have $c = 2 \implies \hat{y}_{i_{c=0}} = 1 - \hat{y}_{i_{c=1}}$



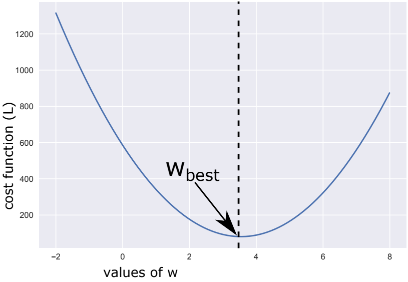
$\implies$ **convex function easy to optimize without local minima**

Defining our objective: cross-entropy

Cross-entropy is a good cost function:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} L_i(w) \quad | \quad L_i(w) = -\sum_{c} y_{ic} * \log(\hat{y}_{ic}) \tag{13}$$

For a binary classification we have $c = 2 \implies \hat{y}_{i c=0} = 1 - \hat{y}_{i c=1}$



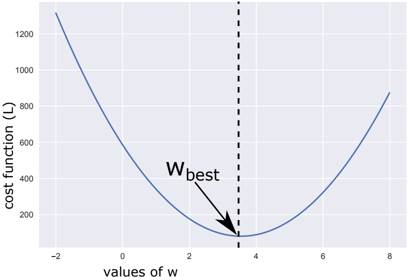
$\implies$ **convex function easy to optimize without local minima**

Defining our objective: cross-entropy

Cross-entropy is a good cost function:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} L_i(w) \quad | \quad L_i(w) = -\sum_{c} y_{i_c} * \log(\hat{y}_{i_c}) \tag{13}$$

For a binary classification we have $c = 2 \implies \hat{y}_{i_{c=0}} = 1 - \hat{y}_{i_{c=1}}$



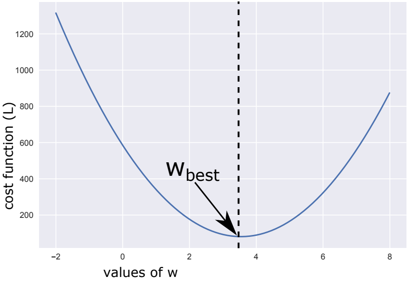
$\implies$ **convex function easy to optimize without local minima**

Defining our objective: cross-entropy

Cross-entropy is a good cost function:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} L_i(w) \quad | \quad L_i(w) = -\sum_{c} y_{i_c} * \log(\hat{y}_{i_c}) \tag{13}$$

For a binary classification we have $c = 2 \implies \hat{y}_{i_{c=0}} = 1 - \hat{y}_{i_{c=1}}$



$\implies$ **convex function easy to optimize without local minima**

The model: logistic regression

$$\hat{y} = \sigma(W * X + b) \tag{14}$$

Cost function: cross-entropy

$$L(w) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) \tag{15}$$

Optimisation algorithm: gradient descent

$$W = W - \alpha\frac{\partial L(W)}{\partial W} \tag{16}$$

$\implies$ gradient descent algorithm stay unchanged

only the calculation of $\frac{\partial L(W)}{\partial W}$ change:

$$w = w - \frac{\alpha}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i) * x_i \tag{17}$$

Binary classification

The model: logistic regression

$$\hat{y} = \sigma(W * X + b) \tag{14}$$

Cost function: cross-entropy

$$L(w) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \tag{15}$$

Optimisation algorithm: gradient descent

$$W = W - \alpha \frac{\partial L(W)}{\partial W} \tag{16}$$

$\Longrightarrow$ gradient descent algorithm stay unchanged

only the calculation of $\frac{\partial L(W)}{\partial W}$ change:

$$w = w - \frac{\alpha}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) * x_i \tag{17}$$

The model: logistic regression

$$\hat{y} = \sigma(W * X + b) \tag{14}$$

Cost function: cross-entropy

$$L(w) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \tag{15}$$

Optimisation algorithm: gradient descent

$$W = W - \alpha \frac{\partial L(W)}{\partial W} \tag{16}$$

$\Longrightarrow$ gradient descent algorithm stay unchanged

only the calculation of $\frac{\partial L(W)}{\partial W}$ change:

$$w = w - \frac{\alpha}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) * x_i \tag{17}$$

Binary classification

The model: logistic regression

$$\hat{y} = \sigma(W * X + b) \tag{14}$$

Cost function: cross-entropy

$$L(w) = -y \log(\hat{y}) - (1 - y)\log(1 - \hat{y}) \tag{15}$$

Optimisation algorithm: gradient descent

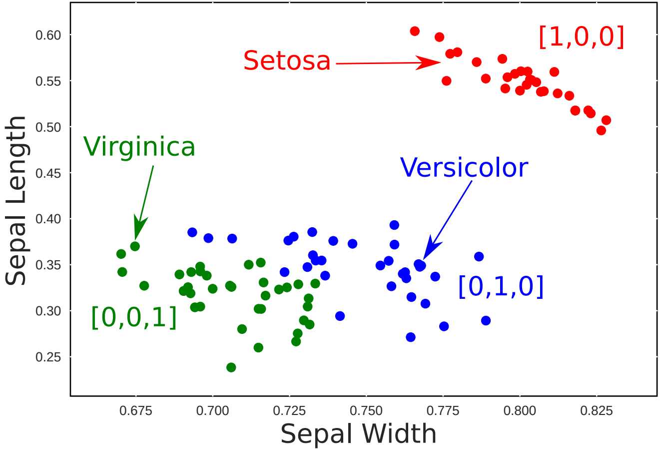
$$W = W - \alpha \frac{\partial L(W)}{\partial W} \tag{16}$$

$\Longrightarrow$ gradient descent algorithm stay unchanged

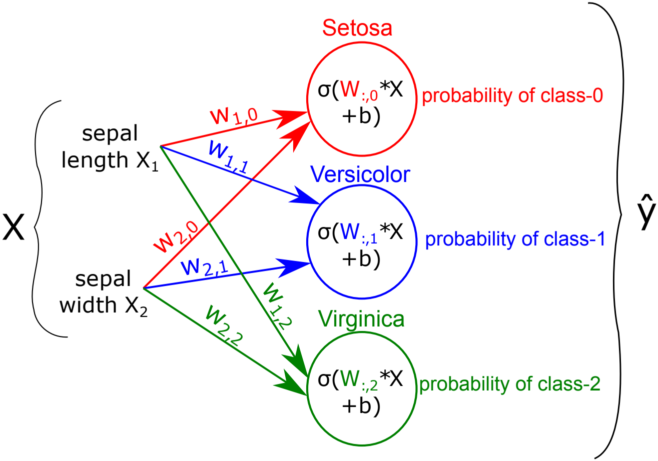
only the calculation of $\frac{\partial L(W)}{\partial W}$ change:

$$w = w - \frac{\alpha}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) * x_i \tag{17}$$

Multi-class problems

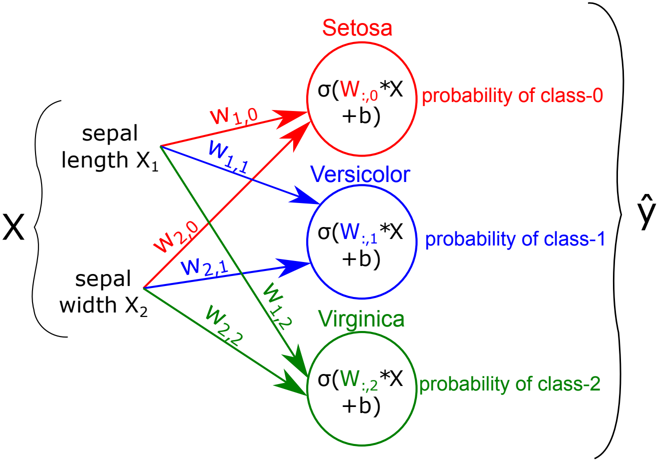


Softmax Classifier: C neurons for C classes

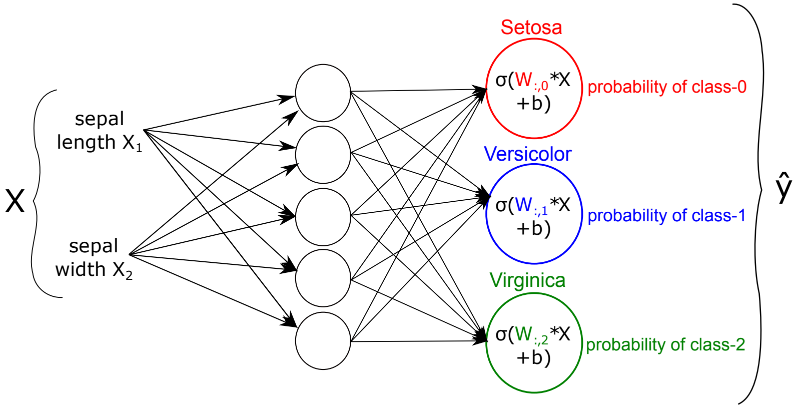


Multi-layer networks

A single-layer network (logistic regression)



A multi-layer network (only one hidden layer)



Why do we need to add hidden layers?

Necessary for learning non-linear functions

$$\hat{y} = h(X) \quad | \quad h \text{ being a nonlinear function} \tag{18}$$

Logistic regression gives a linear decision because there is no non-linear interaction between the terms X_1 and X_2, for example

$$\hat{y} = \sigma(W_1 * X_1 + W_2 * X_2 + b) \tag{19}$$

To make it non-linear you have to add interaction terms

$$\hat{y} = \sigma(W_1 * X_1 + W_2 * X_2 + W_3 * X_1 * X_2 + W_4 * X_1^2 + W_5 * X_2^2 + \cdots + b) \tag{20}$$

This is difficult to do for several reasons:

- Large number of attributes X: an image of 128×128 pixels for example
- We don't know what kind of interaction: $X_1^2 * \sqrt{X_2}$ or $X_1^2 * X_2^3$?

Why do we need to add hidden layers?

Necessary for learning non-linear functions

$$\hat{y} = h(X) \quad | \quad h \text{ being a nonlinear function} \tag{18}$$

Logistic regression gives a linear decision because there is no non-linear interaction between the terms X_1 and X_2, for example

$$\hat{y} = \sigma(W_1 * X_1 + W_2 * X_2 + b) \tag{19}$$

To make it non-linear you have to add interaction terms

$$\hat{y} = \sigma(W_1 * X_1 + W_2 * X_2 + W_3 * X_1 * X_2 + W_4 * X_1^2 + W_5 * X_2^2 + \cdots + b) \tag{20}$$

This is difficult to do for several reasons:

- Large number of attributes X: an image of 128×128 pixels for example
- We don't know what kind of interaction: $X_1^2 * \sqrt{X_2}$ or $X_1^2 * X_2^3$?

Necessary for learning non-linear functions

$$\hat{y} = h(X) \quad | \quad h \text{ being a nonlinear function} \tag{18}$$

Logistic regression gives a linear decision because there is no non-linear interaction between the terms X_1 and X_2, for example

$$\hat{y} = \sigma(W_1 * X_1 + W_2 * X_2 + b) \tag{19}$$

To make it non-linear you have to add interaction terms

$$\hat{y} = \sigma(W_1 * X_1 + W_2 * X_2 + W_3 * X_1 * X_2 + W_4 * X_1^2 + W_5 * X_2^2 + \cdots + b) \tag{20}$$

This is difficult to do for several reasons:

- Large number of attributes X: an image of 128×128 pixels for example
- We don't know what kind of interaction: $X_1^2 * \sqrt{X_2}$ or $X_1^2 * X_2^3$?

Why do we need to add hidden layers?

Necessary for learning non-linear functions

$$\hat{y} = h(X) \quad | \quad h \text{ being a nonlinear function} \tag{18}$$

Logistic regression gives a linear decision because there is no non-linear interaction between the terms X_1 and X_2, for example

$$\hat{y} = \sigma(W_1 * X_1 + W_2 * X_2 + b) \tag{19}$$

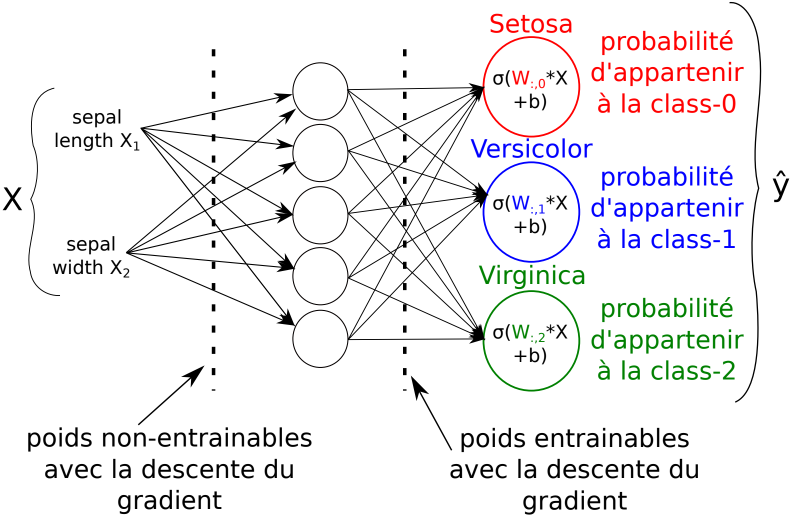
To make it non-linear you have to add interaction terms

$$\hat{y} = \sigma(W_1 * X_1 + W_2 * X_2 + W_3 * X_1 * X_2 + W_4 * X_1^2 + W_5 * X_2^2 + \cdots + b) \tag{20}$$

This is difficult to do for several reasons:

- Large number of attributes X: an image of 128×128 pixels for example
- We don't know what kind of interaction: $X_1^2 * \sqrt{X_2}$ or $X_1^2 * X_2^3$?

Why is it difficult to train multi-layer networks?



Why is it difficult to train multi-layer networks?

We only know what should be the output of a neuron belonging to the last layer (not hidden)

$\Longrightarrow$ **We can calculate the error for these neurons**

$\Longrightarrow$ **We can apply the gradient descent**

We don't know what the output of a neuron belonging to a hidden layer must be

$\Longrightarrow$ **We can't calculate the error for these neurons**

$\Longrightarrow$ **You cannot directly apply the gradient descent**

We must find a way to estimate the error for such a neuron belonging to a hidden layer: **backpropagation**

Why is it difficult to train multi-layer networks?

We only know what should be the output of a neuron belonging to the last layer (not hidden)

$\Longrightarrow$ **We can calculate the error for these neurons**

$\Longrightarrow$ **We can apply the gradient descent**

We don't know what the output of a neuron belonging to a hidden layer must be

$\Longrightarrow$ **We can't calculate the error for these neurons**

$\Longrightarrow$ **You cannot directly apply the gradient descent**

We must find a way to estimate the error for such a neuron belonging to a hidden layer: **backpropagation**

Why is it difficult to train multi-layer networks?

We only know what should be the output of a neuron belonging to the last layer (not hidden)

$\Longrightarrow$ **We can calculate the error for these neurons**

$\Longrightarrow$ **We can apply the gradient descent**

We don't know what the output of a neuron belonging to a hidden layer must be

$\Longrightarrow$ **We can't calculate the error for these neurons**

$\Longrightarrow$ **You cannot directly apply the gradient descent**

We must find a way to estimate the error for such a neuron belonging to a hidden layer: **backpropagation**

Why is it difficult to train multi-layer networks?

We only know what should be the output of a neuron belonging to the last layer (not hidden)

$\Longrightarrow$ **We can calculate the error for these neurons**

$\Longrightarrow$ **We can apply the gradient descent**

We don’t know what the output of a neuron belonging to a hidden layer must be

$\Longrightarrow$ **We can’t calculate the error for these neurons**

$\Longrightarrow$ **You cannot directly apply the gradient descent**

We must find a way to estimate the error for such a neuron belonging to a hidden layer: **backpropagation**

Why is it difficult to train multi-layer networks?

We only know what should be the output of a neuron belonging to the last layer (not hidden)

$\Longrightarrow$ **We can calculate the error for these neurons**

$\Longrightarrow$ **We can apply the gradient descent**

We don't know what the output of a neuron belonging to a hidden layer must be

$\Longrightarrow$ **We can't calculate the error for these neurons**

$\Longrightarrow$ **You cannot directly apply the gradient descent**

We must find a way to estimate the error for such a neuron belonging to a hidden layer: **backpropagation**

Why is it difficult to train multi-layer networks?

We only know what should be the output of a neuron belonging to the last layer (not hidden)

$\Longrightarrow$ **We can calculate the error for these neurons**

$\Longrightarrow$ **We can apply the gradient descent**

We don't know what the output of a neuron belonging to a hidden layer must be

$\Longrightarrow$ **We can't calculate the error for these neurons**

$\Longrightarrow$ **You cannot directly apply the gradient descent**

We must find a way to estimate the error for such a neuron belonging to a hidden layer: **backpropagation**

Why is it difficult to train multi-layer networks?

We only know what should be the output of a neuron belonging to the last layer (not hidden)

$\Longrightarrow$ **We can calculate the error for these neurons**

$\Longrightarrow$ **We can apply the gradient descent**

We don't know what the output of a neuron belonging to a hidden layer must be

$\Longrightarrow$ **We can't calculate the error for these neurons**

$\Longrightarrow$ **You cannot directly apply the gradient descent**

We must find a way to estimate the error for such a neuron belonging to a hidden layer: **backpropagation**

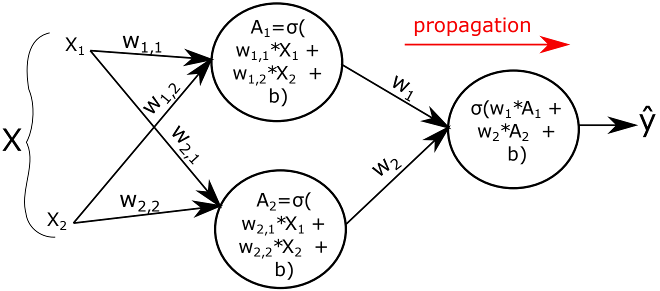
Gradient backpropagation

The cost function is independent of the model

For a binary classification problem: Binary Cross-Entropy

$$L(w) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \tag{21}$$

Our goal is to minimize $L(w)$ **by varying** w

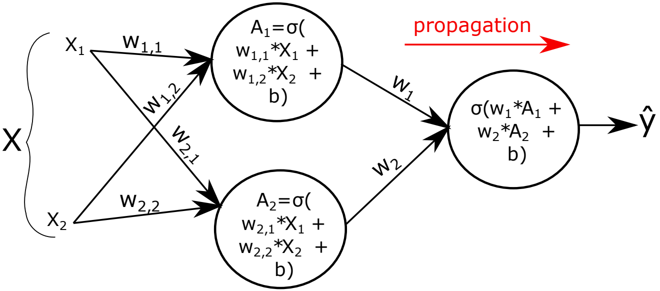


The output of the network is calculated such that:

$$\hat{y} = \sigma(w_1 * A_1 + w_2 * A_2) \tag{22}$$

A_i **activations of neurons** i **belonging to the hidden layer:**

$$A_i = \sigma(w_{i,1} * X_1 + w_{i,2} * X_2) \tag{23}$$

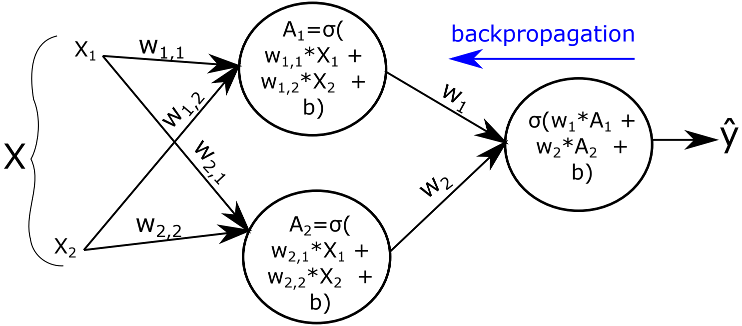


The output of the network is calculated such that:

$$\hat{y} = \sigma(w_1 * A_1 + w_2 * A_2) \tag{22}$$

A_i **activations of neurons** i **belonging to the hidden layer:**

$$A_i = \sigma(w_{i,1} * X_1 + w_{i,2} * X_2) \tag{23}$$

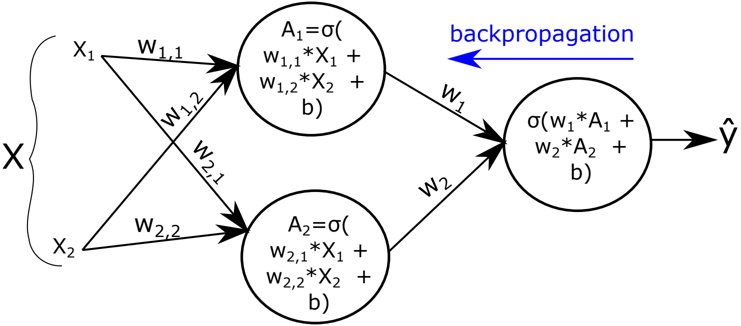


For w_1 and w_2 gradient descent can easily be applied

$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i} \tag{24}$$

We calculate the derivative with respect to w_i

$$\frac{\partial L}{\partial w_i} = (\hat{y} - y) * A_i \tag{25}$$

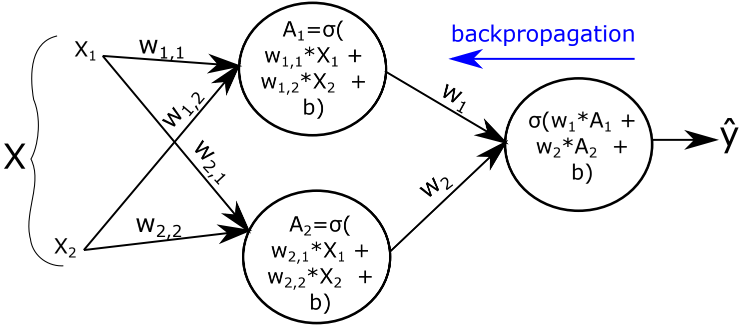


For w_1 and w_2 gradient descent can easily be applied

$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i} \tag{24}$$

We calculate the derivative with respect to w_i

$$\frac{\partial L}{\partial w_i} = (\hat{y} - y) * A_i \tag{25}$$



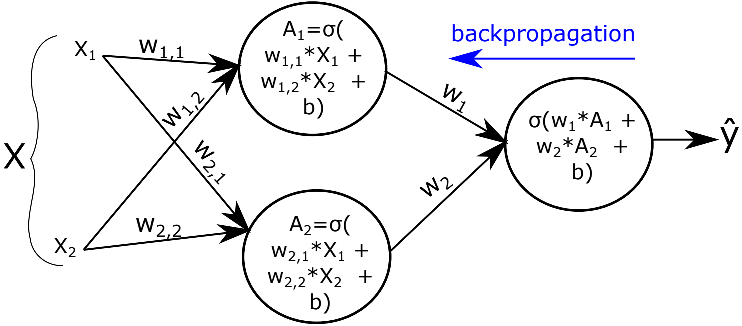
For $w_{i,1}$ (or $w_{i,2}$) the derivative cannot be calculated directly

$$\frac{\partial L}{\partial w_{i,1}} \tag{26}$$

According to the derivation theorem for compound functions

$$\frac{\partial L}{\partial w_{i,1}} = \frac{\partial L}{\partial A_i}\frac{\partial A_i}{\partial w_{i,1}} \tag{27}$$

Backpropagation (Backward pass)

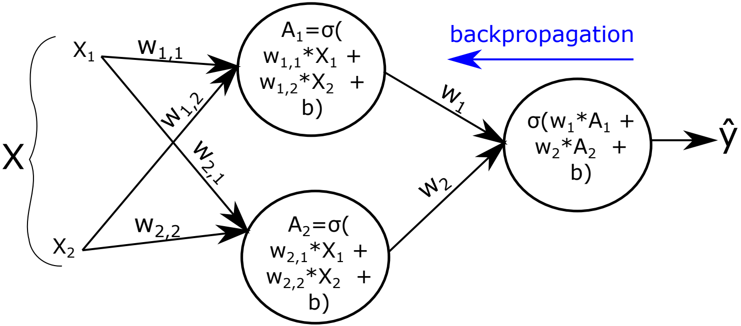


For $w_{i,1}$ (or $w_{i,2}$) the derivative cannot be calculated directly

$$\frac{\partial L}{\partial w_{i,1}} \tag{26}$$

According to the derivation theorem for compound functions

$$\frac{\partial L}{\partial w_{i,1}} = \frac{\partial L}{\partial A_i}\frac{\partial A_i}{\partial w_{i,1}} \tag{27}$$



$$\frac{\partial L}{\partial w_{i,1}} = \frac{\partial L}{\partial A_i}\frac{\partial A_i}{\partial w_{i,1}} \tag{28}$$

Equation (28) allows to calculate derivatives for any number of layers $\implies$ allows to train a multi-layer network

Gradient descent for weight of a hidden neuron

The gradient descent can now be applied

$$w_{i,1} = w_{i,1} - \alpha * X_1 * w_i * A_i(1 - A_i)(\hat{y} - y) \tag{29}$$

We can see how the error term $(\hat{y} - y)$ (calculated for the output layer) is **propagated** to perform the gradient descent on a neuron belonging to a hidden layer

Propagated $\implies$ **Backpropagation** of the gradient (or error)

By repeating this process for each hidden layer, a multi-layer network is created: the error spreads from one layer to another.

Gradient descent for weight of a hidden neuron

The gradient descent can now be applied

$$w_{i,1} = w_{i,1} - \alpha * X_1 * w_i * A_i(1 - A_i)(\hat{y} - y) \tag{29}$$

We can see how the error term $(\hat{y} - y)$ (calculated for the output layer) is **propagated** to perform the gradient descent on a neuron belonging to a hidden layer

Propagated $\Longrightarrow$ **Backpropagation** of the gradient (or error)

By repeating this process for each hidden layer, a multi-layer network is created: the error spreads from one layer to another.

Gradient descent for weight of a hidden neuron

The gradient descent can now be applied

$$w_{i,1} = w_{i,1} - \alpha * X_1 * w_i * A_i(1 - A_i)(\hat{y} - y) \tag{29}$$

We can see how the error term $(\hat{y} - y)$ (calculated for the output layer) is **propagated** to perform the gradient descent on a neuron belonging to a hidden layer

Propagated $\implies$ **Backpropagation** of the gradient (or error)

By repeating this process for each hidden layer, a multi-layer network is created: the error spreads from one layer to another.

Gradient descent for weight of a hidden neuron

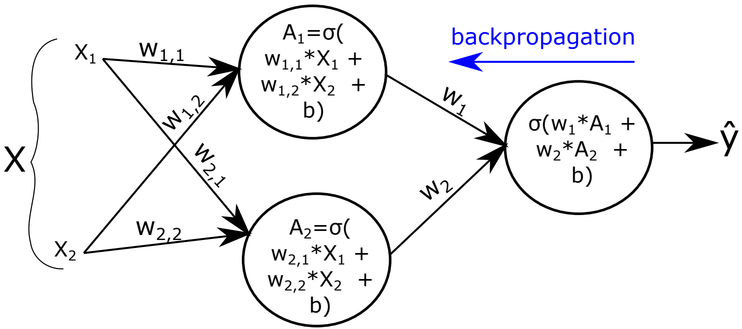
The gradient descent can now be applied

$$w_{i,1} = w_{i,1} - \alpha * X_1 * w_i * A_i(1 - A_i)(\hat{y} - y) \tag{29}$$

We can see how the error term $(\hat{y} - y)$ (calculated for the output layer) is **propagated** to perform the gradient descent on a neuron belonging to a hidden layer

Propagated $\implies$ **Backpropagation** of the gradient (or error)

By repeating this process for each hidden layer, a multi-layer network is created: the error spreads from one layer to another.



A hidden neuron receives from the previous layer: $z = W * X + b$

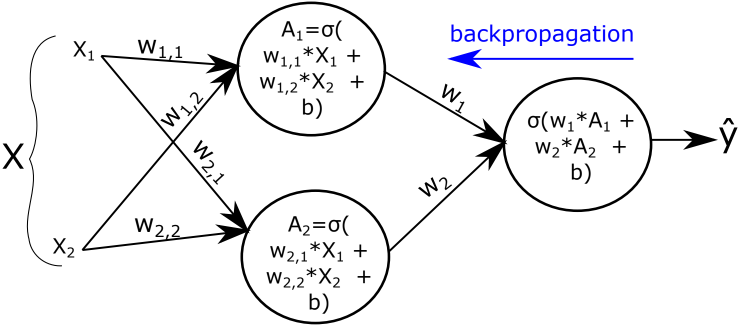
This neuron performs a non-linear interaction between its inputs

This requires that the output (or activation) of the neuron: $A = f(z)$

With $f(.)$ **being a non-linear function for example:** $\sigma(.), tanh(.)$

If $f(.)$ **is linear the network will learn only linear decisions**

Activation functions



A hidden neuron receives from the previous layer: $z = W * X + b$

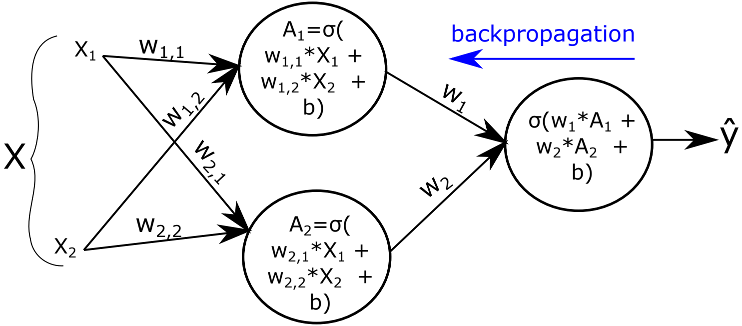
This neuron performs a non-linear interaction between its inputs

This requires that the output (or activation) of the neuron: $A = f(z)$

With $f(.)$ **being a non-linear function for example:** $\sigma(.), tanh(.)$

If $f(.)$ **is linear the network will learn only linear decisions**

Activation functions



A hidden neuron receives from the previous layer: $z = W * X + b$

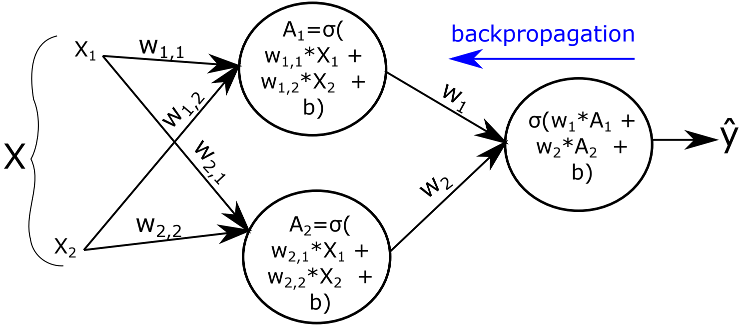
This neuron performs a non-linear interaction between its inputs

This requires that the output (or activation) of the neuron: $A = f(z)$

With $f(.)$ **being a non-linear function for example:** $\sigma(.), tanh(.)$

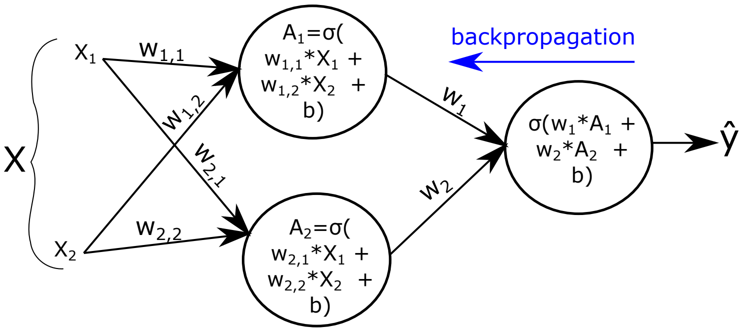
If $f(.)$ **is linear the network will learn only linear decisions**

Activation functions



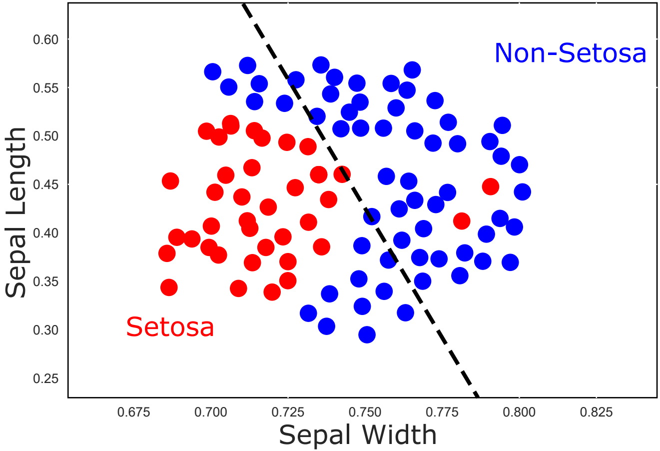
A hidden neuron receives from the previous layer: $z = W * X + b$
This neuron performs a non-linear interaction between its inputs
This requires that the output (or activation) of the neuron: $A = f(z)$
With $f(.)$ **being a non-linear function for example:** $\sigma(.), tanh(.)$
If $f(.)$ **is linear the network will learn only linear decisions**

Activation functions

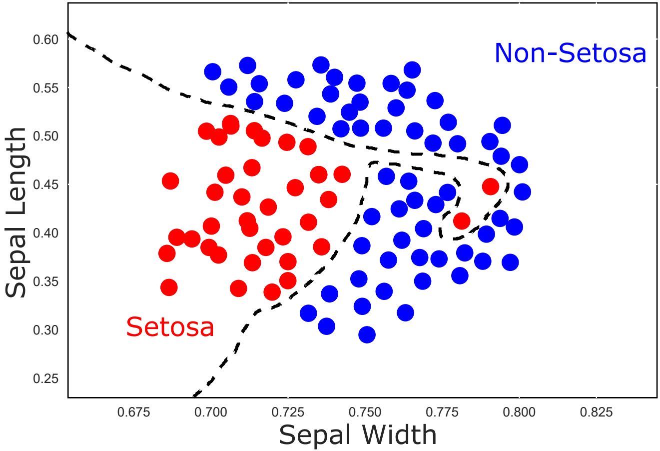


A hidden neuron receives from the previous layer: $z = W * X + b$
This neuron performs a non-linear interaction between its inputs
This requires that the output (or activation) of the neuron: $A = f(z)$
With $f(.)$ **being a non-linear function for example:** $\sigma(.), tanh(.)$
If $f(.)$ **is linear the network will learn only linear decisions**

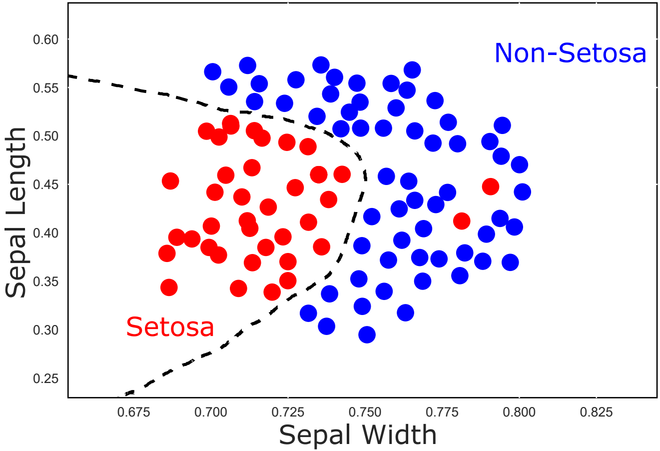
Underfitting



Overfitting



Good generalization: neither overfitting nor underfitting



SCikit-learn implementation

Scikit-learn

```
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix

df = pd.read_csv('data/iris.csv', header=0)
X = df.iloc[:, 0:4]
y = df.select_dtypes(include=[object])

le = preprocessing.LabelEncoder()
y = y.apply(le.fit_transform)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
mlp.fit(X_train, y_train.values.ravel())
predictions = mlp.predict(X_test)
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

Scikit-learn

- Visualization of the class boundaries

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.neural_network import MLPClassifier

n_classes = 3
plot_colors = "ryb"
plot_step = 0.02
iris = load_iris()
for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):
    X = iris.data[:, pair]
    y = iris.target

    clf = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000).fit(X, y
     )

    plt.subplot(2, 3, pairidx + 1)
    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, plot_step),
                         np.arange(y_min, y_max, plot_step))
    plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)
```

Scikit-learn

- Visualization of the class boundaries

```
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[1]])

    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                    cmap=plt.cm.RdYlBu, edgecolor='black', s=15)
plt.suptitle("Decision surface of a GaussianNB using paired features")
plt.legend(loc='lower right', borderpad=0, handletextpad=0)
plt.axis("tight")
plt.show()
```

Scikit-learn

- Visualization of the class boundaries