# Machine Learning –Logistic Regression and Iris Flowers

In this practical, you will learn **logistic regression** and **MLP**.  
and leading to the course project -also learn about **Kaggle** and how to use it.

**Kaggle**

  
Link: <https://www.kaggle.com/>  
A platform for ML (Machine Learning) and DL (Deep Learning) Competitions.  
It has Courses, Competitions, Datasets and much more.  
It is a good place for Data Scientists to start and sharpen their skills in real-world problems (with a lot of help from the community of Kaggle).

**Competitions**

Link: <https://www.kaggle.com/competitions>  
There are competitions in many areas of ML (CV (Computer Vision), etc.) and many topics of real-world problems (Health, Business, Energy, Historical, etc.).  
Some of the competitions have a high prize pool (hundred of thousands of dollars) and some are just for education and enhancing knowledge.

**Datasets**

Link: <https://www.kaggle.com/datasets>  
Kaggle datasets span across all areas of life.   
Users can publish their datasets and let others use it in their projects. Users can also keep a dataset private for their use.  
Datasets can be anything from pictures to audio files or CSV files. Trained models and python packages can sometimes be found here.

**Notebooks**

Link: <https://www.kaggle.com/notebooks>  
Kaggle is providing an internet Jupiter Notebook platform with [GPU (Graphics Processing Unit)](https://en.wikipedia.org/wiki/Graphics_processing_unit) and [TPU (Tensor Processing Unit)](https://en.wikipedia.org/wiki/Tensor_Processing_Unit).  
Kaggle Notebooks connects naturally to Kaggle Datasets and any available dataset can be an input to the notebooks.

**Discussion**

Link: <https://www.kaggle.com/discussion> There is a big community of Data Scientists, ML Developers, DL Developers, and Professionals from every aspect of ML.  
There is a big support for people that are new to the field, and even experienced ML Researchers can find new ideas and points of view in these forums.

**Courses**

Link: <https://www.kaggle.com/learn/overview>  
Kaggle courses can help you grasp the idea of ML conveniently and easily. The Courses are short and well guided, with some theory and a lot of practice.  
In a few hours, one can enter a new field in ML and start getting experience with the competitions on the site.

**Imports and Definitions**

Start with importing all the previous libraries.. math, numpy, pandas, seaborn, matplotlib… and defining the plot parameters, as we did last practical.

Then import sklearn libraries

*# sklearn imports*

**import** **sklearn**

**from** **sklearn** **import** metrics

**from** **sklearn** **import** datasets

**from** **sklearn** **import** pipeline

**from** **sklearn** **import** linear\_model

**from** **sklearn** **import** preprocessing

**from** **sklearn** **import** model\_selection

**Data Investigation and Preprocessing**

We use the iris dataset in this practical and perform a classification task.

As in last practical, start with exploring the data: print the data (iris\_data) description (“DESCR”) and display the data frame ('data', 'feature\_names' and 'target').

You should get a data table with 150 rows and 5 columns:

|  | **sepal length (cm)** | **sepal width (cm)** | **petal length (cm)** | **petal width (cm)** | **Class** |
| --- | --- | --- | --- | --- | --- |
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 |
| **...** | ... | ... | ... | ... | ... |
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | 2 |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | 2 |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | 2 |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 | 2 |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 | 2 |

The iris data has 150 samples - each corresponds to one iris flower:



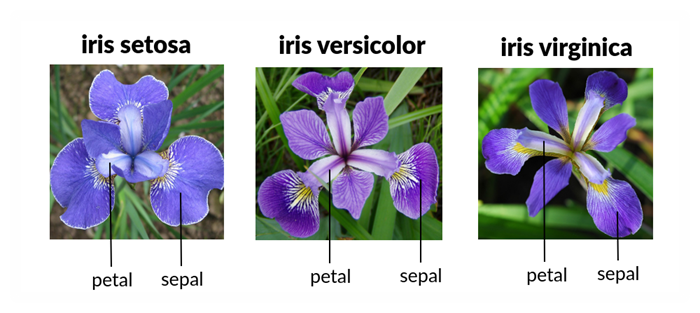
There are three types of iris flowers:

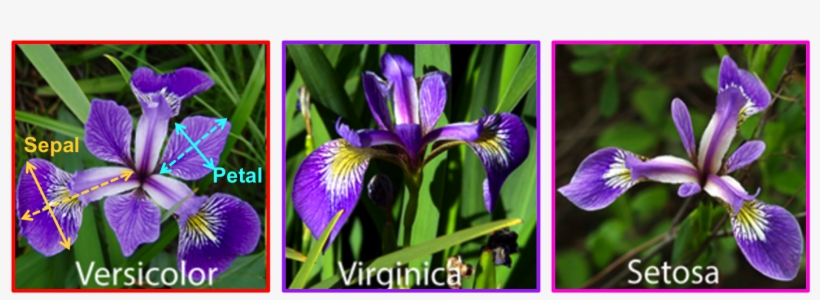
1. Setosa
2. Versicolour
3. Virginica

In order to classify the different types, we have 4 features.

Each flower has two types of leaves:

1. Sepal
2. Petal



Then each sepal and petal has different height and width:  
  
The height is the long, dashed line and the width is the short line.

Your task is to classify each flower, where the classes are the 3 types of Iris, based on these values.

Start with “feeling the data” using visualization:

Plot, using scatterplot the relations between sepal width and length for the 3 flower types (use colours to differentiate the types).

What does the figure explain? i.e. which class has wider sepals? Which has the longest sepal?

Now employ another visualization tool violinplot to see how petal length differs between the three classes (type of Iris).

The figure should display the difference between the classes in terms of petal length.

which class has shorter petals? Which class’ petals are more similar to each other?

Following the last practicals, you’re a Python graphics artist! A good visualization of the correlation between all the features can be portrayed by pairplot, - try it now.

Significant information can be drawn from the graphs:

1.Which of the classes is most separatable from the other two?

2. How are the petal length and petal width related?

3. How are the sepal length and sepal width related?

Having the qualitative observation, you can now check the correlation table using corr, and examine it visually using heatmap.

Which features have the highest correlation with the target (class)?

Let’s do some data preparation here: compute and insert a new feature petal size = petal width \* petal length. Check its correlations compared to the width and length alone.

How is the new feature - petal size correlated with the class?

Let’s move to machine learning now. Use it to compare the two dataframes, with the new feature and without it, to examine if it helped to get better predictions.

First, prepare the data for the machine classification: split the data to 80% train and 20% test sets.

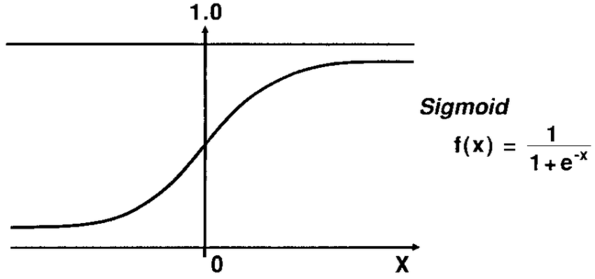
Now we’re ready for **Classification**

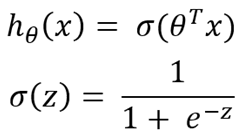
There are 2 “players” in data classification.

1. The Estimator – in our case: [LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
2. The Optimizer – in our case: [SGDClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html" \l "sklearn.linear_model.SGDClassifier) (using log loss)

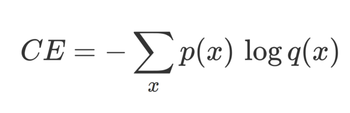
But, The Scikit-learn LogisticRegression estimator does not have a pure GD (or pure SGD) optimizer. In the Scikit-learn LogisticRegression variables an optimizer is called solver: and It has few other solvers, some of them based on SGD. – read about then.

We will use the SGDClassifier with log-loss.

To understand why we use the log-loss function instead of MSE and how we perform the classification, you need to understand the [Sigmoid function](https://en.wikipedia.org/wiki/Sigmoid_function):  
  
The sigmoid function values are in range (0, 1).  
A binary classification should outputs 0 if an item x does not belong to the class and 1 if it is from the class.  
All the values between 0 and 1 indicate how confident is the model in its decision.

In classification, the variable of the sigmoid function is not x but z, where z we is the linear hypothesis of a linear regression, with parameters θ:  


The sigmoid function is thus applied on the linear hypothesis and outputs a confidence number in the range [0, 1], and we get our new logistic regression hypothesis.

Now, we need a strategy for the learning: a loss function, that will give us the same [delta rule](https://en.wikipedia.org/wiki/Delta_rule) as the MSE.   
For logistic regression this loss function is the [CE (Cross-Entropy)](https://en.wikipedia.org/wiki/Cross_entropy) :  
  
where, *p* is the original labels of the data, and *q* is its approximation label which is the output of the hypothesis.

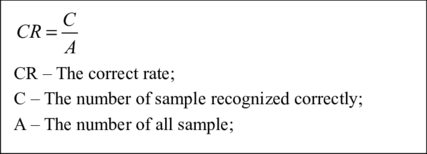
All this was for a binary classification. To use the logistic regression on multiple classes - like in our case, the 3 Iris classes - we apply the OVA (One Versus All) approach =training 3 different classifiers, one for each class.

In order to predict the class for a new data - we run the three classifiers on each new sample and return the class with the lowest loss value.

create the SGDClassifier and predict the probabilities of the train and test data. Pipeline the preprocessing (standatization), and the classification, defining the parameters: loss='log', alpha=0, learning\_rate='constant', eta0=0.01.

Print the first 5 predictions and the corresponding probabilities for the three classes, for the train and test sets.

The predictions and probabilities you printed just illustrate the operation of the classifier. We now need a performance (“score”) metric that will summarize how correct is this model.

The score function in Scikit-learn SGDClassifier is the mean accuracy for each label.  
The accuracy is defined as the ratio between the number of correct predictions and all the predictions:  
  
The accuracy values are in range [0, 1].  
The highest the score, the better the model.

Print the accuracy score and CE loss of the train and test.

Now check the same routines on the dataframe with the additional feature

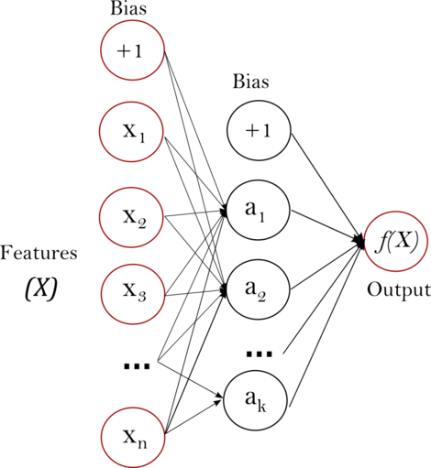
What’s the conclusion? Did the additional feature help the model to better recognize the Iris classes?

**Heads Up: Neural Networks!**

Another way for generating more features: The [NN](https://scikit-learn.org/stable/modules/neural_networks_supervised.html) (Neural Networks).

ANN’s come in different shapes and sizes.

In this practical we will use Scikit-learn [MLPClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html" \l "sklearn.neural_network.MLPClassifier) - a Multi-layer Perceptron, that looks like that:



All the components in this network are Neurons. The left-side neuronal layer receives the input features, the right-side neuron (there can be several) produces the results (In our case the classification result). There can be any number of layers between these two layers. The inner layers are basically transforming features into other features. The more layers, the more features are calculated there that the model has to learn.  
  
The new feature dimension (number of neuron in the middle layers) can be bigger or smaller than the original dimension.

If we want to keep the original features in addition to the new features, there’s an option called bypass (you will learn more about it in the NN class).

The major danger in ANN’s is the overfitting, which stems from too many features.

Let’s try that: import neural\_network and run MLP on the original data.

Import neural\_network and use MLP\_cls = neural\_network.MLPClassifier

Print the results (accuracy and CE loss)

How do these results compare to the LogisticRegression classification? How can this comparison be explained?

**More Information**

Explanation of binary cross-entropy (log loss):  
[Understanding binary cross-entropy / log loss: a visual explanation](https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a)

Wikipedia on Multilayer Perceptron:  
[Multilayer Perceptron](https://en.wikipedia.org/wiki/Multilayer_perceptron)

Explanation on Multilayer Perceptron:  
[Multi-Layer Perceptron (MLP)](https://medium.com/@xzz201920/multi-layer-perceptron-mlp-4e5c020fd28a)

Explanation of the differences between MSE and log-loss for Logistic Regression:  
[Why not Mean Squared Error(MSE) as a loss function for Logistic Regression?](https://towardsdatascience.com/why-not-mse-as-a-loss-function-for-logistic-regression-589816b5e03c)

Explanation of the derivative of the log-loss function:  
[The Derivative of Cost Function for Logistic Regression](https://medium.com/analytics-vidhya/derivative-of-log-loss-function-for-logistic-regression-9b832f025c2d)

Wikipedia on Entropy in Information Theory:  
[Entropy](https://en.wikipedia.org/wiki/Entropy_(information_theory))

A blog post on Neural Networks and their connection to linear and logistic regressions:  
[Understanding objective functions in neural networks.](https://towardsdatascience.com/understanding-objective-functions-in-neural-networks-d217cb068138)