ERHVERVSAKADEMI AARHUS

Machine Learning, Sept 1st, 2025.

Exercises Day 3.

Sila. Sept 1st, 2025.

Exercise 3.1.

A binary classifier is a classifier that separates two classes using a line, a plane or a hyper-plane.

In a linear model we will have that a prediction is made according to a formula like:

y = *w*[0]*x[0] + *w*[1]*x[1] +.... *w*[p]*x[p] + b > 0

If the function is smaller than zero we predict the class -1 (*Type 1*). If the function is larger than zero we predict the class +1 (*Type 2*).

There are many ways to find the coefficients *w,* and the intercept (b).

A decision boundary is found where we have y = 0.

Lets try this. And lets try it using logistic regression. We start by plotting some points:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model

# Array of points with a classification
X = np.array(np.matrix('6,350; 2.5, 400;4.5,500; 3.5,350;
2,300;4,600;7,300;5,500;5,400;6,400;3,400;4,500;1,200;3,400;7,700;3,550;2.5,650'))
y = np.array(np.matrix('0;0;1;0;0;1;1;1;0;1;0;0;0;0;1;1;0'))[:,0]

pos=np.where(y==1)
neg=np.where(y==0)

plt.plot(X[pos[0],0], X[pos[0],1], 'ro')
plt.plot(X[neg[0],0], X[neg[0],1], 'bo')
plt.xlim([min(X[:,0]),max(X[:,0])])
plt.ylim([min(X[:,1]),max(X[:,1])])
plt.plot
plt.show()
```

We can then use sklearn to help us find the weights for a model that we can use for classification:

You want to plot $\theta^T X = 0$, where X is the vector containing (1, x, y). That is, you want to plot the line defined by `theta[0] + theta[1]*x + theta[2]*y = 0`. Solve for y:

```
y = -(theta[0] + theta[1]*x)/theta[2]
```

```
logreg = linear_model.LogisticRegression(C=1000)
model = logreg.fit(X, y)
# model.coef_[0,0]*x + model.coef_[0,1]*y + model.intercept_[0] = 0
# y = - ( model.coef_[0,0]*x +  model.intercept_[0]) / model.coef_[0,1]
xx = np.linspace(1, 7)
yy = -  (model.coef_[0,0] / model.coef_[0,1]) * xx - ( model.intercept_[0]  /  model.coef_[0,1])
plt.plot(xx, yy, 'k-')
```

Afterwards, lets try it with some slightly different set of points, like below:

```
X = np.array(np.matrix('4,450;5,600;6,700;4.5,550;4.9,500;5,650;5.5,500; 5.25,525; 4.25,625; 4.75,575'))
y = np.array(np.matrix('0;1;1;0;0;1;0;1;1;1'))[:,0]
```

The trade-off parameter of logistic regression that determines the strength of the regularization is called C, and higher values of C correspond to less regularization (where we can specify the regularization function).C is actually the Inverse of regularization strength(lambda). Higher C values risk overfitting. **Try adjusting C.**

Exercise 3.2.

It is possible to use the techniques above on more realistic datasets.

One popular dataset is the “exam dataset”, where we get scores for students on two exams (each exam gives 0..100 points), along with information on whether the students passed the course (sounds sort of familiar I guess...). The beginning of the dataset looks like this:

```
34.62365962451697,78.0246928153624,0
30.28671076822607,43.89499752400101,0
35.84740876993872,72.90219802708364,0
60.18259938620976,86.30855209546826,1
79.0327360507101,75.3443764369103,1
45.08327747668339,56.3163717815305,0
61.10666453684766,96.51142588489624,1
75.02474556738889,46.55401354116538,1
76.09878670226257,87.42056971926803,1
84.43281996120035,43.53339331072109,1
95.86155507093572,38.22527805795094,0
75.01365838958247,30.60326323428011,0
82.30705337399482,76.48196330235604,1
69.36458875970939,97.71869196188608,1
39.53833914367223,76.03681085115882,0
53.9710521485623,89.20735013750205,1
69.07014406283025,52.74046973016765,1
```

67.94685547711617,46.67857410673128,0

70.66150955499435,92.92713789364831,1

76.97878372747498,47.57596364975532,1

67.37202754570876,42.83843832029179,0

89.67677575072079,65.79936592745237,1

50.534788289883,48.85581152764205,0

34.21206097786789,44.20952859866288,0

77.9240914545704,68.9723599933059,1

62.27101367004632,69.95445795447587,1

80.1901807509566,44.82162893218353,1

93.114388797442,38.80067033713209,0

61.83020602312595,50.25610789244621,0

38.78580379679423,64.99568095539578,0

61.379289447425,72.80788731317097,1

85.40451939411645,57.05198397627122,1

52.10797973193984,63.12762376881715,0

52.04540476831827,69.43286012045222,1

40.23689373545111,71.16774802184875,0

54.63510555424817,52.21388588061123,0

33.91550010906887,98.86943574220611,0

64.17698887494485,80.90806058670817,1

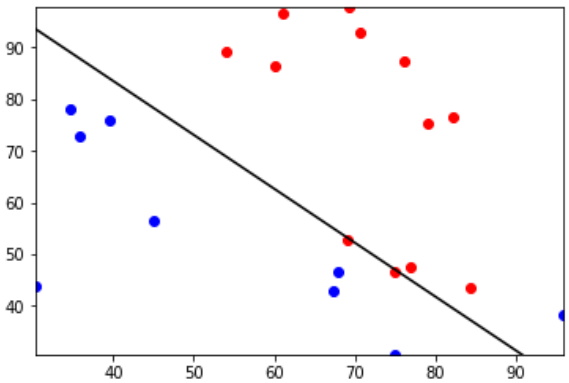
74.78925295941542,41.57341522824434,0

This clearly sounds like something we can do logistic regression on.

Transforming the data (some of it) to the right format:

```
X = np.array(np.matrix('34.62365962451697,78.0246928153624;30.28671076822607,43.89499752400101;
35.84740876993872,72.90219802708364;60.18259938620976,86.30855209546826;79.0327360507101,75.3443764369103;45.08327747668339,56.3163717815305;
61.10666453684766,96.51142588489624; 75.02474556738889,46.55401354116538; 76.09878670226257,87.42056971926803;84.43281996120035,43.53339331072109;
95.86155507093572,38.22527805795094; 75.01365838958247,30.60326323428011; 82.30705337399482,76.48196330235604; 69.36458875970939,97.71869196188608;
39.53833914367223,76.03681085115882; 53.9710521485623,89.20735013750205; 69.07014406283025,52.74046973016765; 67.94685547711617,46.67857410673128;
70.66150955499435,92.92713789364831; 76.97878372747498,47.57596364975532;67.37202754570876,42.83843832029179'))

y = np.array(np.matrix('0;0;0;1;1;0;1;1;1;1;0;0;1;1;0;1;1;0;1;1;0'))[:,0]
```

We are almost good to go. Try to plot the data. Then, make small adjustments to the program from the previous exercise, so that you can use it to get a decision boundary with the help of logistic regression (looking something like this):



Experiment with the C value for the logistic regression. Which values is the best?

Exercise 3.3.

When we work with sklearn's methods for logistic regression we have to (can) set certain values for the model. One of them is the C value. Also labelled the regularization parameter, $C = 1/\lambda$ (Notice: Here, the parameter controlling is not lambda, as in other linear models, but the inverse: C. The higher C is, the less the model is regularized).

Lambda (λ) controls the trade-off between allowing the model to increase it's complexity as much as it wants with trying to keep it simple. For example, if λ is very low or 0, the model will have enough power to increase it's complexity (overfit) by assigning big values to the weights for each parameter. If, in the other hand, we increase the value of λ, the model will tend to underfit, as the model will become too simple.

The parameter decides the strength of the regularization. Higher values of C corresponds to less regularization. In other words: When you use high values for the parameter C, the logistic regression algorithm will try to fit the training data as best as possible. It is important that each data point is classified correctly. While lower values will cause the algorithm to try to adjust to a majority of data points with a simple model. Perhaps to simple, underfitting the data.

Using low values of C will cause the algorithms to try to adjust to the "majority" of data points, while using a higher value of C stresses the importance that each individual data point be classified correctly.

In this exercise we will try to find a reasonable value for C.

First we will look at some points:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model

# Array of points with a classification
X = np.array(np.matrix('2,300;4,600;7,300;5,500;5,400;6,400;3,400;4,500;1,200;3,400;7,700;3,550;2.5,650'))
y = np.array(np.matrix('0;1;1;1;0;1;0;0;0;0;1;1;0'))[:,0]

pos=np.where(y==1)
neg=np.where(y==0)

plt.plot(X[pos[0],0], X[pos[0],1], 'ro')
plt.plot(X[neg[0],0], X[neg[0],1], 'bo')
plt.xlim([min(X[:,0]),max(X[:,0])])
plt.ylim([min(X[:,1]),max(X[:,1])])

plt.show()
```

Try to run this.

As usual, it is a good idea to take this step by step so that you know what is going on.

Just before we plot the data, using plt.show(), we will then insert some code that will try to make a classifier for our points:

```
logreg = linear_model.LogisticRegression(C=10000)

h = .02  # step size in the mesh

# we fit the data.
logreg.fit(X, y)

score= logreg.score(X, y)

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Ravel() return a contiguous flattened array.
# Predict the color for each x,y point
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot. Reshape point to xx format
Z = Z.reshape(xx.shape)

# figure(Num = our figure number, figsize=(1,1)) creates an inch-by-inch image
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
```

Again, try to understand the code, and what it is doing.

Then: First you will have to investigate the score value. **What is this?**

Print it.

Now you will have to change the C value. Tune it.

Notice how things change as you tune C.

Try out at least 5-10 C values to get a good understanding of what is going on here.

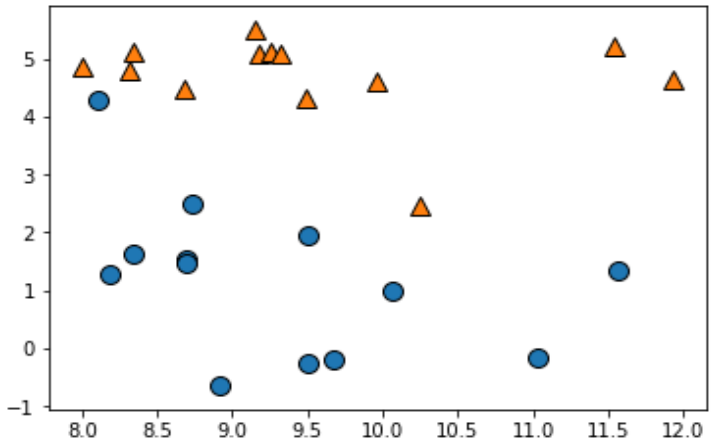
Are the statements in the introduction to this exercise correct?

Exercise 3.4.

There are a lot of small and synthetic (meaning made-up) datasets available on the internet, that are useful when we want to highlight certain aspects of machine learning algorithms.

Andreas Mueller (author of the book “Introduction to ML with Python”) has made the “forge dataset” available on:

https://github.com/amueller/introduction_to_ml_with_python/tree/master/mglearn



We won't go into the details of how he actually makes the dataset, but it should be apparent that it will make a good test case for logistic regression.

A version of the script (forge_dataset) is also available on Canvas.

Run it with different values in the make_forge method.

So, we now only need some code that can make a classification of the data.

Take a look at the code (below), and see, if you can make it work. And, equally important, understand the idea behind (in broad strokes) what the code is doing, and how you might be able to adjust it for use when dealing with similar problems?

```
# Use logistic regression for classification

model = LogisticRegression().fit(X, y)


x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5

y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5


# evenly spaced numbers over a specified interval

xx = np.linspace(x_min, x_max, 1000)

yy = np.linspace(y_min, y_max, 1000)


#coordinate matrices from coordinate vectors

X1, X2 = np.meshgrid(xx, yy)

#contiguous flattened array

X_grid = np.c_[X1.ravel(), X2.ravel()]
```

```
#Predict confidence scores for samples.
#A plane of equal probability, where the model has determined
#both target classes are equally likely.
decision_values = model.decision_function(X_grid)

#A contour plot displays line with equal likelyhood, draw decision boundary
axes.contour(X1, X2, decision_values.reshape(X1.shape), levels=[0],
        colors="black", alpha=1, linewidths=None,
        linestyles="solid", zorder=5)
```

Again, try non-default values for C.

I.e. re-introduce a C value setting in the code. And try out different values, 3-4 different values.

Re-do with different forge sets.

I.e. different values for random_state & n_samples in the make_forge method.

Exercise 3.5.

(Preferably) Discuss the code and results in this exercise in groups of 2-3, in order to make sure that you all know what is going on. Make extra “print-statements” in the code to clarify the code.

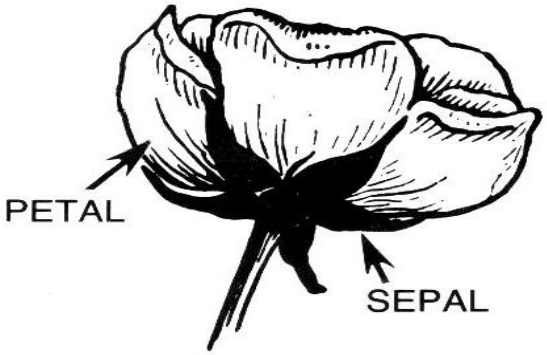
We will then take a look at the ”Iris Data Set”.

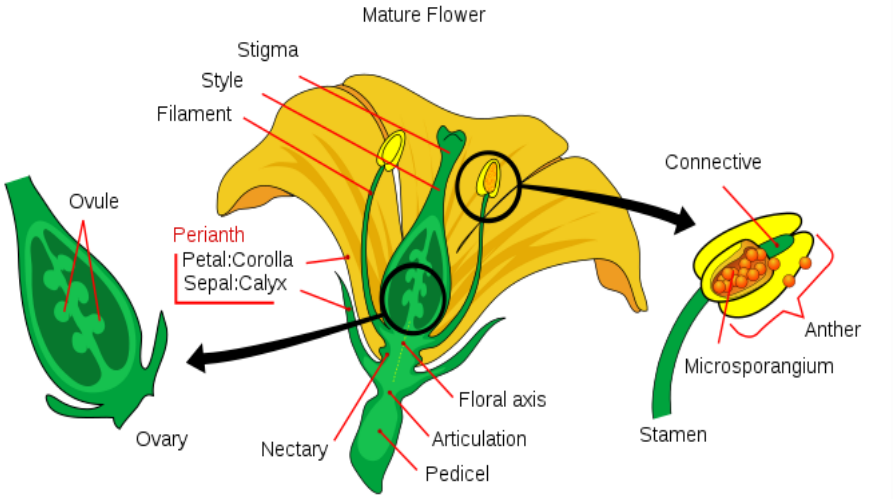
https://archive.ics.uci.edu/ml/datasets/Iris

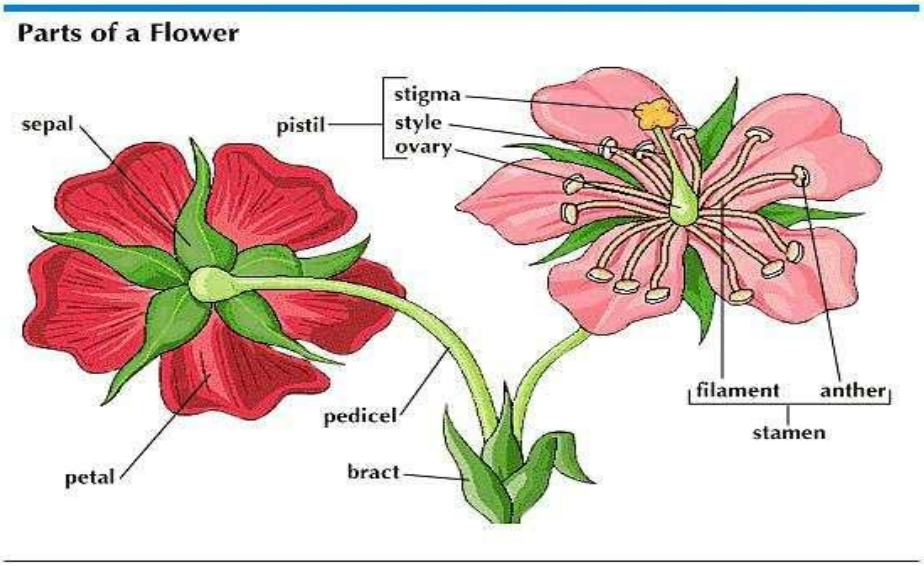
”This is perhaps the best known database to be found in the pattern recognition literature”

”A good project because it is so well understood”.

- The Iris Dataset is a classification problem, allowing you to practice with perhaps an easier type of supervised learning algorithm.
- It only has 4 attributes and 150 rows, meaning it is small and easily fits into memory (and a screen or A4 page).
- All of the numeric attributes are in the same units and the same scale, not requiring any special scaling or transforms to get started.







First we are going to load the data set:

```
import pandas

# Load dataset
```

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pandas.read_csv(url, names=names)
```

If this causes any problems, you might want to download the iris.data file to your local machine, and then let the url point to your local folder.

Or use:

```
# from sklearn.datasets import load_iris
iris = load_iris()
# datasets.load_iris()
```

Print the dataset:

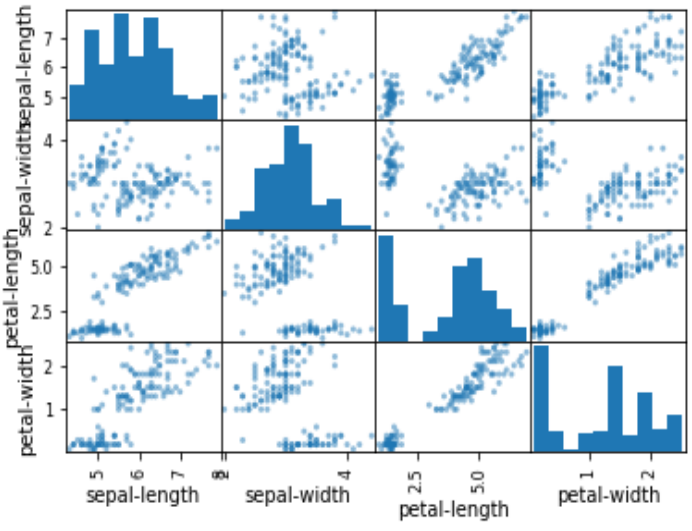
```
print(dataset.shape)
# head
print(dataset.head(20))

# descriptions
print(dataset.describe())
```

Will give you a first glimpse of what the dataset contains. Histograms can also be helpful:

```
# histograms
dataset.hist()
plt.show()
```

Just as scatter plots

```
# scatter plot matrix
scatter_matrix(dataset)
plt.show()
```



We will now try to take a closer look at (only) the petal-width, and see if we can make a classification based on that.

```
array = dataset.values
```

```
X = array[:,3]  # petal width
Y = array[:,4]  # classification

# Make y into numbers
a_enc = pandas.factorize(Y)
yvalues = a_enc[0]

y = []

for i in yvalues:
  if i == 2:
     y = np.append(y,[1])
  else:
     y = np.append(y,[0])

# l2 is default,
# Intuitively, the model will be adjusted to minimize single outlier case,
# at the expense of many other common examples
log_reg = LogisticRegression(penalty="l2")
# Shape, gives a new shape to an array without changing its data. In order to use algorithm
log_reg.fit(X.reshape(-1, 1),y)

plt.plot(X,y,"b.")

X_new = np.linspace(0,3,1000).reshape(-1,1)
y_proba = log_reg.predict_proba(X_new)

plt.plot(X_new,y_proba[:,1],"g-",label="Iris-Virginica")
plt.plot(X_new,y_proba[:,0],"b--",label="Not Iris-Virginca")
plt.xlabel("Petal width", fontsize=14)
plt.ylabel("Probability", fontsize=14)
plt.legend(loc="upper left", fontsize=14)
plt.show()
```

As usual, you are here asked to test and verify the code. Try out a few examples (by hand) from the dataset, and see if it gives the expected results.

Instead, of just printing the first 20 data elements, print the whole dataset.

You might also want to change the default parameter, l2, to l1 regularization, and see if this gives a better graph in your opinion (without going into the precise math behind the l1, and l2 terms).

Next, it would be interesting to see a plot for both petal-width and petal-length.

The code is probably going to look something like this:

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt

iris = datasets.load_iris()

X = iris["data"][:,(2,3)]  # petal length, petal width
yy = iris["target"]

y = []
for i in yy:
  if i == 2:
     y = np.append(y,[1])
  else:
     y = np.append(y,[0])

#model = LogisticRegression(multi_class="multinomial", solver="lbfgs", C=5)
model = LogisticRegression(C=1000)
model.fit(X,y)

plt.plot(X[:, 0][y==1], X[:, 1][y==1], "y.", label="Virginica")
plt.plot(X[:, 0][y==0], X[:, 1][y==0], "b.", label="Setosa or versicolor")

plt.legend(loc="upper left", fontsize=14)

plt.xlabel("Petal length", fontsize=14)
plt.ylabel("Petal width", fontsize=14)

xx = np.linspace(1, 6)
yy = -  (model.coef_[0,0] / model.coef_[0,1]) * xx - ( model.intercept_[0]  /  model.coef_[0,1])

plt.plot(xx, yy, 'k-')
plt.show()
```

Verify that the plot supports the findings from the previous plot? Make changes to the program that solidifies your understanding of the plot.

There are many options (for parameters) to play around with for Logistic regression in sklearn. Try a few of them, and see if this changes the results. C values etc.

Write down you findings.

See:

http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html