02 - Classification

From lesson we learned that 3 approaches are possible for classification:

- **Discriminant function approach**:
  - model a *function* that maps inputs to classes
  - fit model to data
- **Probabilistic discriminative approach**:
  - model a *conditional probability* $P(C_k|x)$
  - fit model to data
- **Probabilistic generative approach**:
  - model *likelihood* $P(x|C_k)$ and *prior* $P(C_k)$
  - fit models to data
  - infer posterior $P(C_k|x) = \frac{P(C_k)P(x|C_k)}{P(x)}$

the difference with the previous is that here we only model the prior and the likelihood, the posterior comes automatically (in the previous method we model directly the posterior)

Iris dataset for classification

In this session, we take into account again the Iris dataset. This time we are more interested in the discrimination of the sample class, i.e., either Setosa, Versicolor or Virginica.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

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

In [3]:
```
dataset.head()
```

Out[3]:

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [4]:
```
dataset['class'].unique()
```

Out[4]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

We do not have any metric over the space of the classes, i.e., it is not possible to order them. In this case one **could not** consider regression techniques.

Let us start with discriminating between Setosa and non-Setosa flowers according to the sepal length and width.

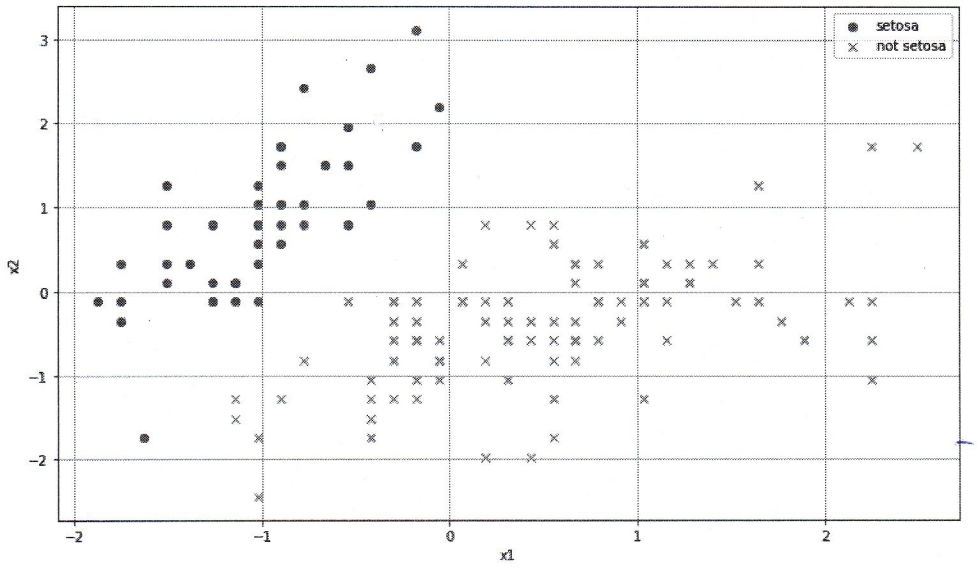
In [5]:
```
from scipy.stats import zscore
from sklearn.utils import shuffle

X = zscore(dataset[['sepal-length', 'sepal-width']].values)
t = dataset['class'].values == 'Iris-setosa'
X, t = shuffle(X, t, random_state=0) # this time we have to do it!
```

(boolean index)

we shuffle both data and target (to keep them aligned)

In [6]:
```
setosa = X[t]
not_setosa = X[~t]
```

In [7]:
```
plt.figure(figsize=(12,7))
plt.scatter(setosa[:, 0], setosa[:, 1], label='setosa')
plt.scatter(not_setosa[:, 0], not_setosa[:, 1], label='not setosa', marker='x')

plt.xlabel('x1')
plt.ylabel('x2')
plt.grid()
plt.legend()
plt.show()
```



linearly separable

Discriminant Function Approach: the Perceptron

At first, let us perform a classification with a perceptron classifier:

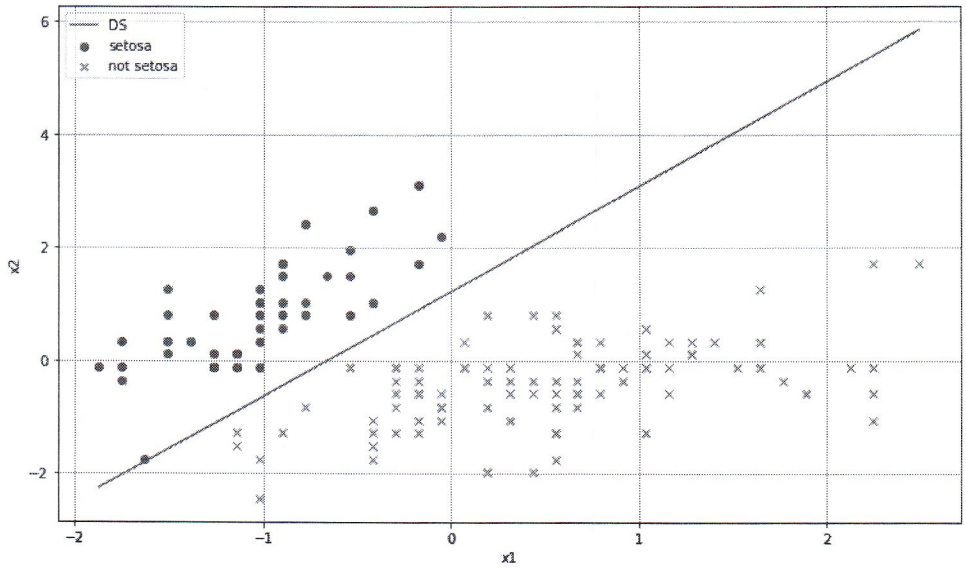
- Hypothesis space: $y(\mathbf{x}_n) = sgn(\mathbf{w}^T\mathbf{x}_n) = sgn(w_0 + x_{n1}w_1 + x_{n2}w_2)$;
- Loss measure: Distance of misclassified points from the separating surface $L_P(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^T\mathbf{x}_n C_n$;
- Optimization method: Online Gradient Descent;

where $sgn(\cdot)$ is the sign function.

In [8]:
```
from sklearn.linear_model import Perceptron
perc_classifier = Perceptron(alpha=1, shuffle=False, random_state=0)
perc_classifier.fit(X, t)
```

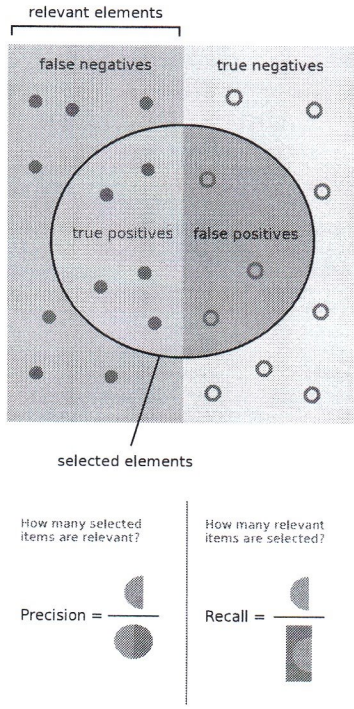
Out[8]:
```
Perceptron(alpha=1, class_weight=None, early_stopping=False, eta0=1.0,
           fit_intercept=True, max_iter=1000, n_iter_no_change=5, n_jobs=None,
           penalty=None, random_state=0, shuffle=False, tol=0.001,
           validation_fraction=0.1, verbose=0, warm_start=False)
```

In [9]:
```
plt.figure(figsize=(12,7))
plt.scatter(setosa[:, 0], setosa[:, 1], label='setosa')
plt.scatter(not_setosa[:, 0], not_setosa[:, 1], label='not setosa', marker='x')

# Plot the DS

coef = perc_classifier.coef_.flatten() # weights
w0 = perc_classifier.intercept_ # bias
w1 = coef[0]
w2 = coef[1]

step = 100
ds_x1 = np.linspace(X[:,0].min(), X[:,0].max(), step)
# Compute x2 componenent given some x1:
# w^T x + x0 = 0 -> w0 + w1 * x1 + w2 * x2 = 0 -> x2 = - (w0 + w1*x1) / w2
ds_x2 = [-(w0 + w1*x1) / w2 for x1 in ds_x1]
plt.plot(ds_x1, ds_x2, label='DS')

plt.xlabel('x1')
plt.ylabel('x2')
plt.grid()
plt.legend()
plt.show()
```



Evaluating Classification

To evaluate the performances of the chosen method, we need to compute the *confusion matrix* which tells us the number of points which have been correctly classified and those which have been misclassified.



	Actual Class: 1	Actual Class: 0
Predicted Class: 1	tp	fp
Predicted Class: 0	fn	tn

true/false negative/positive
wrong/right prediction (correctness) prediction

Based on this matrix we can evaluate:

- Accuracy: $Acc = \frac{tp+tn}{N}$ fraction of the samples correctly classified in the dataset;
- Precision $Pre = \frac{tp}{tp+fp}$ fraction of samples correctly classified in the positive class among the ones classified in the positive class;
- Recall: $Rec = \frac{tp}{tp+fn}$ fraction of samples correctly classified in the positive class among the ones belonging to the positive class;
- F1 score: $F1 = \frac{2 \cdot Pre \cdot Rec}{Pre+Rec}$ harmonic mean of the precision and recall;

where tp is the number of true positives, fp is the number of false positives, fn are the false negatives and tn are the true negatives. Equivalently, we can look at the meaning of Precision and Recall by looking at the figure above.

Remember that:

- The higher these figures of merits the better the algorithm is performing.
- These performance measures are **not** symmetric, but depends on the class we selected as positive.
- Depending on the **application** one might switch the classes to have measures which better evaluate the predictive power of the classifier.

In [10]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
```

In [11]:
```
t_pred = perc_classifier.predict(X)
```

In [12]:
```
confusion_matrix(t, t_pred)
```

Out[12]:
```
array([[100,   0],
       [  0,  50]])
```

In [13]:
```
accuracy_score(t, t_pred)
```

Out[13]:
```
1.0
```

In [14]:
```
precision_score(t, t_pred)
```

Out[14]:
```
1.0
```

In [15]:
```
recall_score(t, t_pred)
```

Out[15]:
```
1.0
```

In [16]:
```
f1_score(t, t_pred)
```

Out[16]:
```
1.0
```

Implementing the Perceptron classifier

The loss function is the distance of the misclassified from the boundary. Minimized how? Gradient descent (online gradient descent)

In [17]:
```
w = np.ones(3) # initialization of the weights
n_epochs = 10
for epoch in range(n_epochs):
  for i, (x_i, t_i) in enumerate(zip(X, t)):
    # correct t_i to be in {-1, 1} - the target of the perceptron is +1/-1
    corr_t_i = 1 if t_i else -1
    ext_x = np.concatenate([np.ones(1), x_i.flatten()]) # we add "1" to the vector of features just for the scalar product to work properly
    if np.sign(w.dot(ext_x)) != corr_t_i:
      w = w + ext_x * corr_t_i
```

"extended - x" (because it has 1)

if the target predicted ≠ the update

Notice that this procedure will stop if the classes are linearly separable, while it does not stop if the two classes are overlapping.

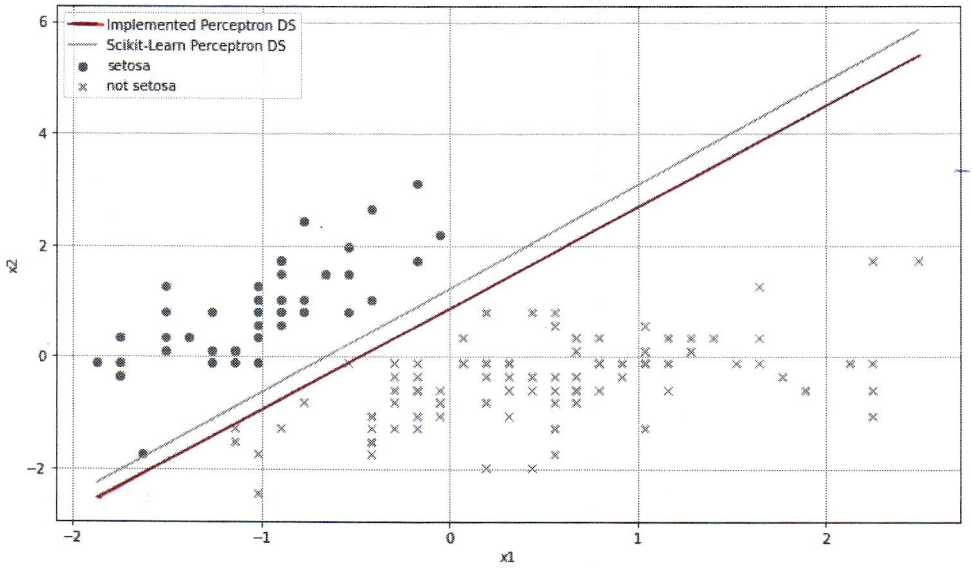
Moreover, we do not know how long the procedure will take to reach convergence.

This makes impossible to distinguish between a procedure which is *slowly converging* to a *non-linearly separable* setting. !

In [18]:
```
# Since we would rewrite the same code, it is a good idea to write a function
def plot_ds(X, w, step=100, label='DS'):
  ds_x1 = np.linspace(X[:,0].min(), X[:,0].max(), step)
  ds_x2 = [-(w[0] + w[1]*x1) / w[2] for x1 in ds_x1]
  plt.plot(ds_x1, ds_x2, label=label)
```

In [19]:
```
plt.figure(figsize=(12,7))
plt.scatter(setosa[:, 0], setosa[:, 1], label='setosa')
plt.scatter(not_setosa[:, 0], not_setosa[:, 1], label='not setosa', marker='x')

# Implemented Perceptron
plot_ds(X, w, label='Implemented Perceptron DS')

# Sklearn Perceptron
coef = perc_classifier.coef_.flatten() # weights
w0 = perc_classifier.intercept_ # bias
perc_w = np.array([w0, coef[0], coef[1]])
plot_ds(X, perc_w, label='Scikit-Learn Perceptron DS')

plt.xlabel('x1')
plt.ylabel('x2')
plt.grid()
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: VisibleDeprecationWarning: Creating an n
darray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with differe
nt lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creati
ng the ndarray
  # This is added back by InteractiveShellApp.init_path()
```



they're different but both divide correctly

Probabilistic Discriminative Approach: Logistic Regression

Let us change the methods for the classification task and use a Logistic regression classifier with two classes:

- Hypothesis space: $y_n = y(x_n) = \sigma(w_0 + x_{n1}w_1 + x_{n2}w_2)$;
- Loss measure: Loglikelihood $L(\mathbf{w}) = -\sum_{n=1}^{N}[C_n \ln y_n + (1 - C_n)\ln(1 - y_n)]$;
- Optimization method: Gradient Descent;

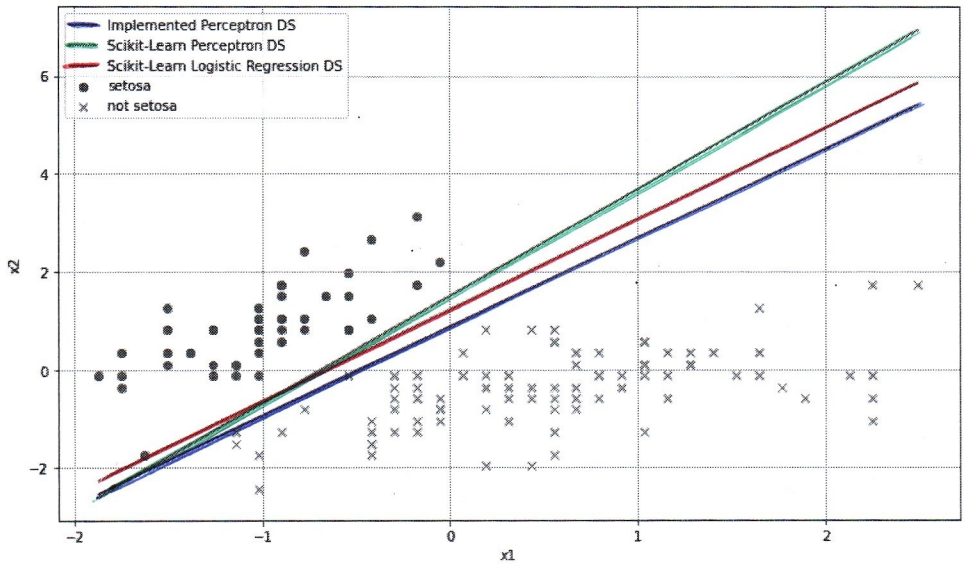
where the sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$.

In [20]:
```
from sklearn.linear_model import LogisticRegression

log_classifier = LogisticRegression(penalty='none') # regularization is applied as default
log_classifier.fit(X, t)
```

Out[20]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [21]:
```
# compare perceptron, by hand and logistic regression
plt.figure(figsize=(12,7))
plt.scatter(setosa[:, 0], setosa[:, 1], label='setosa')
plt.scatter(not_setosa[:, 0], not_setosa[:, 1], label='not setosa', marker='x')

# Implemented Perceptron
plot_ds(X, w, label='Implemented Perceptron DS')

# Sklearn Perceptron
coef = perc_classifier.coef_.flatten() # weights
w0 = perc_classifier.intercept_ # bias
perc_w = np.array([w0, coef[0], coef[1]])
plot_ds(X, perc_w, label='Scikit-Learn Perceptron DS')

# Sklearn Logistic Regression
coef = log_classifier.coef_.flatten() # weights
w0 = log_classifier.intercept_ # bias
log_w = np.array([w0, coef[0], coef[1]])
plot_ds(X, log_w, label='Scikit-Learn Logistic Regression DS')

plt.xlabel('x1')
plt.ylabel('x2')
plt.grid()
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: VisibleDeprecationWarning: Creating an n
darray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with differe
nt lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creati
ng the ndarray
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: VisibleDeprecationWarning: Creating an n
darray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with differe
nt lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creati
ng the ndarray
```



If we perform the $logit(x) = \log(\frac{x}{1-x})$ transformation to the output we have:

$$logit(y_n) = w_0 + x_{n1}w_1 + x_{n2}w_2,$$

and, thus, we have the same statistical characterization of the parameters $\mathbf{w}$ as we had in the linear regression if we consider as output a specific transformation of the target, i.e., we can perform hypothesis testing on the significance of the parameters. ≠ perceptron

Multiple Classes

In [22]:
```
multi_t = dataset['class']
multi_log_classifier = LogisticRegression()
multi_log_classifier.fit(X, multi_t)
```

Out[22]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Probabilistic Generative Approach: Naive Bayes

Generative models have the purpose of modeling the joint pdf of the couple input/output $p(C_k, \mathbf{x})$, which allows us to generate also **new data** from what we learned.

This is different from the probabilistic discriminative models, in which we are only interested in computing the probabilities that a given input is coming from a specific class $p(C_k|\mathbf{x})$, which is not sufficient to produce new samples.

Conversely, we will see how it is possible to generate new samples if we are provided with an approximation of the joint input/output distribution $p(C_k, \mathbf{x})$.

In this case, the Naive Bayes method considers the **naive assumption** that each input is conditionally (w.r.t. the class) independent from each other. If we consider the Bayes formula we have:

$$\begin{aligned}
p(C_k|\mathbf{x}) &= \frac{p(C_k)\, p(\mathbf{x}|C_k)}{p(\mathbf{x})} \\
&\propto p(x_1, \dots, x_M, C_k) \\
&= p(x_1|x_2, \dots, x_M, C_k)p(x_2, \dots, x_M, C_k) \\
&= p(x_1|x_2, \dots, x_M, C_k)p(x_2|x_3, \dots, x_M, C_k)p(x_3, \dots, x_n, C_k) \\
&= p(x_1|x_2, \dots, x_M, C_k) \dots p(x_M|C_k)p(C_k) \\
&= p(x_1|C_k) \dots p(x_M|C_k)p(C_k) \\
&= p(C_k) \prod_{j=1}^{M} p(x_j|C_k).
\end{aligned}$$

naive

The decision function, which maximises the Maximum A Posteriori probability, is the following:

$$y(\mathbf{x}) = \arg\max_k p(C_k) \prod_{j=1}^{M} p(x_j|C_k),$$

where as usual we do not consider the normalization factor $p(\mathbf{x})$.

In a specific case we have to define a prior distribution for the classes $p(C_k)\ \forall k$ and a distribution to compute the likelihood of the considered samples $p(x_j|C_k)\ \forall J,\ \forall k$.

In the case of continuous variable one of the usual assumption is to use Gaussian distributions for each variable $p(x_j|C_k) = \mathcal{N}(x_j; \mu_{jk}, \sigma^2_{jk})$ and either a uniform prior $p(C_k) = \frac{1}{K}$ or a multinomial prior based on the samples proportions $p(C_k) = \frac{\sum_{i=1}^{N} I\{\mathbf{x}_n \in C_k\}}{N}$, where $I\{\cdot\}$ is the indicator function.

The complete model of Naive Bayes is:

- Hypothesis space: $y_n = y(x_n) = \arg\max_k p(C_k) \prod_{j=1}^{M} p(x_j|C_k)$;
- Loss measure: Log likelihood;
- Optimization method: MLE.

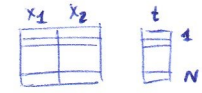
In [23]:
```
from sklearn.naive_bayes import GaussianNB

gnb_classifier = GaussianNB()
gnb_classifier.fit(X, t)
t_pred = gnb_classifier.predict(X)

print(accuracy_score(t, t_pred))
print(recall_score(t, t_pred))
print(precision_score(t, t_pred))
print(confusion_matrix(t, t_pred))
```

```
0.9933333333333333
0.98
1.0
[[100   0]
 [  1  49]]
```
one misclassified

Generate new data

Using the estimated priors $p(C_k)$ and likelihoods $p(x_j|C_k) = \mathcal{N}(x_j; \mu_{jk}, \sigma^2_{jk})$ it is possible to generate new data.

In [24]:
```
N = 100

new_samples = np.empty((N, 2))
new_t = np.empty(N, dtype=bool)

for i in range(N):
  # Based on the class priors, we sample a class
  class_ = np.random.choice([0,1], p=gnb_classifier.class_prior_)
  new_t[i] = class_

  # For each feature, we have a normal distribution of its likelihood given the class

  # theta: mean of each feature per class (n_classes, n_features)
  thetas = gnb_classifier.theta_[class_, :]

  # sigma: variance of each feature per class (n_classes, n_features)
  sigmas = gnb_classifier.sigma_[class_, :]

  # sample x1
  new_samples[i,0] = np.random.normal(thetas[0], sigmas[0], 1)
  # sample x2
  new_samples[i,1] = np.random.normal(thetas[1], sigmas[1], 1)

# divide samples by class
new_setosa = new_samples[new_t, :]
new_not_setosa = new_samples[~new_t, :]
```
we create empty structures



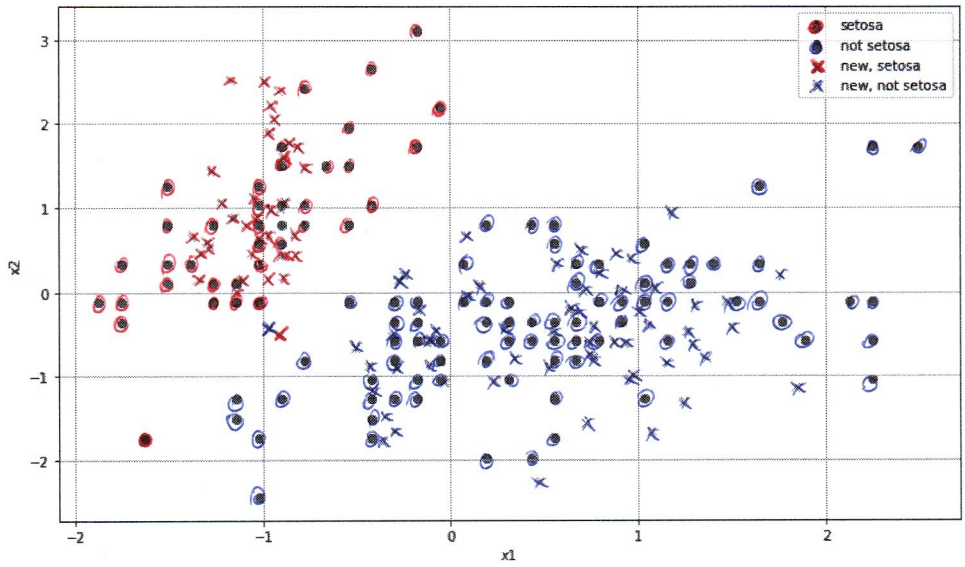
we generate the target using the prior that we fixed for the classes

random choice among the ones proposed in the vector (in this case [0,1]), choice made according to the multinomial prior (in this case: gnb-classifier.class_prior_)

we recover θ, σ of the attributes given the class (a couple (θ,σ) for the attribute x_1 and a couple (θ,σ) for the attribute x_2, both the (θ,σ)'s are of the class chosen at the beginning)

In [25]:
```
plt.figure(figsize=(12,7))

# plot real samples
plt.scatter(setosa[:, 0], setosa[:, 1], label='setosa', color='red')
plt.scatter(not_setosa[:, 0], not_setosa[:, 1], label='not setosa', color='blue')

# plot generated samples
plt.scatter(new_setosa[:, 0], new_setosa[:, 1], label='new, setosa', color='red', marker='x', alpha=0.3)
plt.scatter(new_not_setosa[:, 0], new_not_setosa[:, 1], label='new, not setosa', color='blue', marker='x

plt.xlabel('x1')
plt.ylabel('x2')
plt.grid()
plt.legend()
plt.show()
```

This can be useful for data augmentation (all generative algorithms can)



Notice that the Naive Bayes is **not** a Bayesian method.

Indeed, the priors we compute are estimated from data, and not updated using likelihoods.

This makes Naive Bayes a method which uses the Bayes theorem to model the independence among the input, given the classes.

Homework

Implementing Logistic Regression

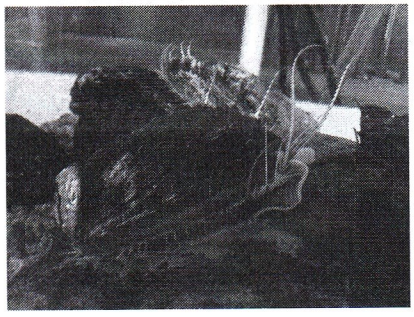
Implement the learning algorithm for the logistic regression on the Iris dataset by relying on the batch gradient descent optimization.

In which situation the use of this optimization algorithm is a good idea?

In [26]:
```
from scipy.stats import zscore
from sklearn.utils import shuffle

X = zscore(dataset[['sepal-length', 'sepal-width']].values)
t = dataset['class'].values == 'Iris-setosa'
X, t = shuffle(X, t, random_state=0) # this time we have to do it!

### WRITE YOUR CODE HERE ###
```

Classifying molluscs - Part 1

Have you ever dreamt of becoming a marine biologist?



Now, you can (almost) realize your dream by classifying molluscs (abalones) with the Abalone dataset!

Train a **perceptron** which uses the Length and Height of the shell to predict if an Abalone is *Male* or *Female* (remove the infants from the dataset analysed).

In [27]:
```
abalone_df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data',
                         names=['Sex', 'Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight',

### WRITE YOUR CODE HERE ###
```

Classifying molluscs - Part 2

Train a **logistic regression** which uses the Length and Height of the shell to predict if an Abalone is *Male* or *Female* (remove the infants from the dataset analysed).

Plot the separating hyperplane obtained by the logistic regression.

In [28]:
```
### WRITE YOUR CODE HERE ###
```

Find the error

- Describe the process and purpose of what is implemented in this snippet.
- Tell if the method is sound or if it is necessary to modify the procedure.
- After fixing the error, try to evaluate the performance: is it the method sound? Can you still find some problem?

In [29]:
```
from sklearn import linear_model

# Load iris dataset
X = dataset['sepal-length'].values
t = dataset['class'].values == 'Iris-setosa'
X, t = shuffle(X, t, random_state=0) # this time we have to do it!

# Select 2 set of features
phi = zscore(np.array([X, X**2]).T)
X = zscore(X).reshape(-1, 1)

# fit a regression
lin_model = linear_model.LinearRegression()
lin_model.fit(X, t)
lin_pred = lin_model.predict(X) > 0.5

# fit a regression with a quadratic feature
qua_model = linear_model.LinearRegression()
qua_model.fit(phi, t)
qua_pred = qua_model.predict(phi) > 0.5

n = len(dataset)

# Compute the adjusted R2
m = X.shape[1] + 1
lin_R2 = accuracy_score(t, lin_pred)
lin_adj_R2 = 1-(1-lin_R2)*(n-1)/(n-m)

# Compute the adjusted R2
m = X.shape[1] + 1
qua_R2 = accuracy_score(t, qua_pred)
qua_adj_R2 = 1-(1-qua_R2)*(n-1)/(n-m)

if qua_adj_R2 > lin_adj_R2:
  t_pred = lin_pred
else:
  t_pred = qua_pred
```

4 Classification

Exercise 4.1

Which of the following is an example of *qualitative variable*?

1. Height
2. Age
3. Speed
4. Colour

Provide a method to convert the qualitative ones into quantitative one, without introducing further structure over the data.

Exercise 4.2

Suppose we collect data for a group of workers with variables hours spent working x_1, number of completed projects x_2 and receive a bonus t. We fit a logistic regression and produce estimated coefficients: $w_0 = -6$, $w_1 = 0.05$ and $w_2 = 1$.

Estimate the probability that a worker who worked for 40h and completed 3.5 projects gets an bonus.

How many hours would that worker need to spend working to have a 50% chance of getting an bonus?

Do you think that values of z in $\sigma(z)$ lower than -6 make sense in this problem? Why?

Exercise 4.3

Derive for logistic regression, the gradient descent update for a batch of K samples.

Do we have assurance about converge to the optimum?

Exercise 4.4

Tell if the following statement about the perceptron algorithm for classification are true

or false.

1. Shuffling the initial data influences the perceptron optimization procedure;
2. We are guaranteed that, during the learning phase, the perceptron loss function is decreasing over time;
3. There exists a unique solution to the minimization of the perceptron loss;
4. The choice of a proper learning rate α might speed up the learning process.

Motivate your answer.

✗ Exercise 4.5

You are working on a spam classification system using logistic regression. "Spam" is a positive class ($y = 1$) and "not spam" is the negative class ($y = 0$). You have trained your classifier and there are $N = 1000$ samples. The confusion matrix is:

	Actual Class: 1	Actual Class: 0
Predicted Class: 1	85	890
Predicted Class: 0	15	10

What is the classifier recall? What about the $F1$ score? What would you try to improve in such a system? Should we aim at solving a specific issue?

✗ Exercise 4.6

Suppose you have trained a logistic regression classifier which is output is y_n. Currently, you predict 1 if $y_n > \tau$, and predict 0 if $y_n < \tau$, where currently the threshold is $\tau = 0.5$.

Suppose you decrease the threshold to $\tau = 0.3$. Which of the following are true? Check all that apply and provide a motivation.

1. The classifier is likely to now have higher precision on the training set.
2. The classifier is likely to have unchanged precision and recall, but higher accuracy on the training set.
3. The classifier is likely to now have higher recall on the training set.
4. The classifier is likely to have unchanged precision and recall, but lower accuracy on the training set.

What happens on a generic test set?

When it is a good idea to change the threshold from $\tau = 0.5$ to a lower value?

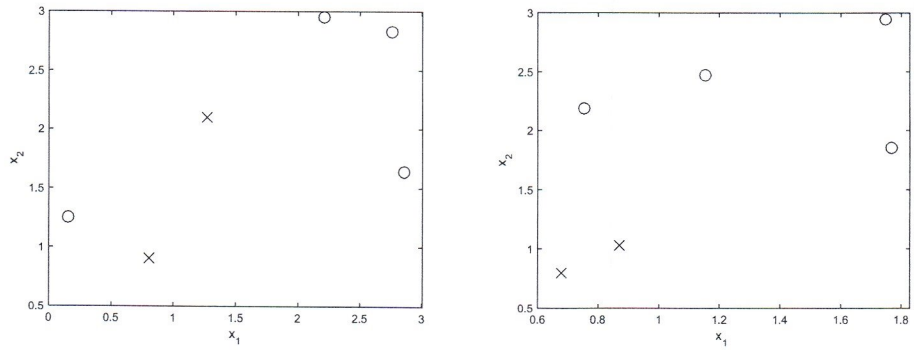
X Exercise 4.7

Which of the following is NOT a linear function in x:

1. $f(x) = a + b^2 x$;
2. $\delta_k(x) = \frac{x\mu}{\sigma^2} - \frac{\mu^2}{2\sigma^2} + \log(\pi)$;
3. $logit(P(y = 1|x))$ where $P(y = 1|x)$ is a logistic regression;
4. $P(y = 1|x)$ from logistic regression;
5. $g(x) = \frac{x-1}{x+1}$;
6. $h(x) = \frac{x^2-1}{x+1}$.

X Exercise 4.8

Consider the following datasets:



and consider the online stochastic gradient descend algorithm to train a perceptron. Does the learning procedure terminates? If so, how many steps we require to reach convergence? Provide motivations for your answers.

What about the Logistic regression?

Exercise 4.9

Starting from the formula of the softmax classifier:

$$y_k(x) = \frac{\exp(w_k^T x)}{\sum_j \exp(w_j^T x)},$$

derive the formula for the sigmoid logistic regression for the two classes problem.

Exercise 4.10

Consider one at a time the following characteristics for an ML problem:

1. Large dataset (big data scenario);
2. Embedded system;
3. Prior information on data distribution;
4. Learning in a Real-time scenario.

Provide motivations for the use of either a parametric or non-parametric method in the above situations.

Exercise 4.11

Consider a classification problem having more than two classes. Proposed a method to deal with multiple classes in each of the following methods:

1. Naive Bayes;
2. Perceptron;
3. Logistic regression;
4. K-NN.

Motivate your answers.

Exercise 4.12

Consider the following dataset to implement a spam filter function:

"pills"	"fee"	"kittens"	Url Presence	"PoliMi"	spam
0	1	0	0	1	0
0	0	1	1	0	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	0	0	0	0
1	1	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	1

where we enumerate the presence of specific word or of an URL in 8 different e-mails and the corresponding inclusion in the spam or non-spam class.

1. Estimate a Naive Bayes classifier, choosing the proper distributions for the classes priors and the feature posteriors.

2. Predict the probability of the following samples to belong to the spam and no-spam classes.

"pills"	"fee"	"kittens"	Url Presence	"PoliMi"
1	1	0	1	0
0	1	1	0	1

Answers

Answer of exercise 4.1

1. Quantitative variable: we are able to order heights and they usually are in a finite set (in $\{50, \dots, 230\}$ cm);
2. Quantitative variable: they are ordered values, all of them being integer numbers;
3. Quantitative variable: this variable takes values in the real numbers;
4. Qualitative variable: since we do not have an ordering over the colours (in general cases), we need to convert the set of the available colours into binary variables.

Given the set C ($|C| = p$) of all possible colours we would like to consider we create a new variable for each colour c. This variable for colour c is equal to one in the case a sample x_i has colour $c_i = c$ and zero otherwise. Clearly we need to create p new variables and for each sample, of which only one is equal to one.

Answer of exercise 4.2

The logistic model provides as output the probability of getting a bonus, thus:

$$P(t = 1|\mathbf{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$$

where $x_1 = 40$ and $x_2 = 3.5$

$$P(t = 1|\mathbf{x}) = \sigma(-6 + 0.05 \cdot 40 + 1 \cdot 3.5) = sigmoid(-0.5) = 0.3775$$

To have a $\alpha\%$ chance of having a bonus we need to invert the sigmoidal function, while in this case we know that we have 50% chance when the argument of the sigmoid is equal to zero, thus:

$$\begin{aligned} w_0 + w_1 \hat{x} + w_2 x_2 &= 0 \\ -6 + 0.05 \cdot \hat{x} + 3.5 &= 0 \\ \hat{x} = \frac{2.5}{0.05} &= 50 \end{aligned}$$

Since all the considered variables are positive definite, it makes only sense to consider values greater than -6 as predictions.

Answer of exercise 4.4

1. TRUE: the learning procedure is influenced by both the initial parameter we consider and the order we present the data to it.
2. FALSE: we are guaranteed that the error (loss) on the currently considered data does not increase.
3. FALSE: if the data are linearly separable, there is an infinite number of linear boundaries able to provide the same loss performance, which are all equivalent solutions for the perceptron.
4. FALSE: the parameter vector norm does not influence the result of the discrimination, thus the use of a generic parameter $\alpha > 0$ would work.

Answer of exercise 4.5

$$Rec = \frac{tp}{tp + fn} = \frac{85}{85 + 15} = 0.85$$
$$Pre = \frac{tp}{tp + fp} = \frac{85}{85 + 890} = 0.087$$
$$F1 = \frac{2 \cdot Rec \cdot Pre}{Pre + Rec} = 2 \cdot 0.85 \cdot 0.0870.85 + 0.087 \approx 0.16$$

In this case it is more harmful to put some non-spam mails in the spam folder, thus we would rather decrease the number of fp than decreasing the fn one.

Answer of exercise 4.6

If we move the threshold τ down we likely have these influence on the confusion matrix:

	Actual Class: 1	Actual Class: 0
Predicted Class: 1	$\uparrow tp$	$\uparrow fp$
Predicted Class: 0	$\downarrow fn$	$\downarrow tn$

If we consider $Rec = \frac{tp}{tp+fn}$ we have that the tp is increasing (numerator) and that $tp + fn$ (number of sample in the positive class) can not change (denominator), thus we will have most likely have a higher recall.

If we consider $Acc = \frac{tp+tn}{N}$ we have that the tp are increasing and tn are decreasing (numerator) and that N can not change (denominator), thus we cannot say if it is likely to decrease or increase. For instance, if the original classifier was able to classify perfectly the data would have likely a lower accuracy.

If we consider $Pre = \frac{tp}{tp+fp}$ we have that the tp are increasing (numerator) and that both tp and fp are increasing. Thus, the denominator increases of a larger amount than the numerator, thus, we will have most likely a decrease in the precision.

Summarizing, the only true statement is the (3).

If the training an test are likely to to have the same distribution, the same effect will occur also to a test set. While the increase and decrease in the quantities for the training set is almost sure (at least they remain the same) on the test set these changes occurs on average.

Since we are likely to increase the recall, we are encouraged to use such a strategy in the case we would like to have a low number of false negatives, i.e., if these errors are critical for the application.

Answer of exercise 4.7

1. Linear;
2. Linear;
3. Linear since $logit(P(y = 1|x)) = \mathbf{w}^T\mathbf{x}$;
4. Nonlinear;
5. Nonlinear;
6. Linear since can be simplified, thus is linear (if we exclude the value $x = -1$).

Answer of exercise 4.8

The perceptron learning algorithm is guaranteed to converge in the case it exists a linear separation surface. In this case, we are able to reduce the classification error to zero, otherwise the optimization procedure does not stop. We do not have any assurance on the convergence rate, since it depends on the starting point for the parameter, and on the ordering of the points we consider for training.

In the first case (left) we are sure it does not converge, while in the second case (right) the online stochastic gradient descend will eventually converge.

Since the loss function for the logistic regression is convex, it has a single minimum point, which is reached by the gradient descent algorithm.

Answer of exercise 4.9

Since we are considering only two classes we have that summation is only over two

parameter vectors $\mathbf{w}_1$ and $\mathbf{w}_2$. If we consider class C_1 we may write:

$$\begin{aligned} y_1(x) &= \frac{\exp(\mathbf{w}_1^T x)}{\exp(\mathbf{w}_1^T x) + \exp(\mathbf{w}_2^T x)} \\ &= \frac{\frac{\exp(\mathbf{w}_1^T x)}{\exp(\mathbf{w}_1^T x)}}{\frac{\exp(\mathbf{w}_1^T x) + \exp(\mathbf{w}_2^T x)}{\exp(\mathbf{w}_1^T x)}} \\ &= \frac{1}{1 + \exp[(\mathbf{w}_2 - \mathbf{w}_1)^T x]}. \end{aligned}$$

Similarly for class C_2 we have the same formula with $(\mathbf{w}_1 - \mathbf{w}_2)^T$. Since we are considering a probability distribution it is not necessary to consider both $y_1(x)$ and $y_2(x)$. Indeed, $y_2(x) = 1 - y_1(x)$, which is why we just need to store a single parameter vector $\mathbf{w} = \mathbf{w}_2 - \mathbf{w}_1$ and the formula for the two class classifier has a single parameter vector:

$$y_1(x) = \frac{1}{1 + \exp(\mathbf{w}^T x)}.$$

Answer of exercise 4.10

1. PARAMETRIC: in the case we have a large dataset it is better to have a model which is able to capture the characteristics of the problem than by basing on the entire dataset to provide a prediction.

2. NON-PARAMETRIC: some of the algorithm we considered in machine learning requires computationally expensive training phases. If the entire system has to work on an embedded system, we should either perform the training phase on a different device or use non parametric methods, which does not require a learning phase.

3. PARAMETRIC: an easy way for introducing a-priori information on the dataset we have in a learning methods is to include them in the model. Since non-parametric only are based on data, it is not trivial to include prior knowledge in them.

4. PARAMETRIC/NON-PARAMETRIC: since we want a fast way of performing tasks a non-parametric method is probably a good idea, since it does not require to have a training phase. On the contrary, if we are able to provide an online method for training a parametric method, we could also consider parametric methodologies.

Answer of exercise 4.11

1. The Naive Bayes algorithm natively supports the existence of multiple classes. Indeed, it models the posterior probability of each class given the sample.
2. For K classes, we can consider K 1 vs. all models and give as label the one providing the maximum value for $\mathbf{w}^T\mathbf{x} + w_0$.
3. For K classes, we can consider K 1 vs. all logistic regressions and consider as a class the one providing the highest probability.
4. We might use majority voting to decide the class. We need to carefully choose the way we are breaking ties, since this might be crucial in the case of many classes.

Answer of exercise 4.12

1. The proper models in this case uses Bernoulli variables both as prior distributions and for the posteriors. More specifically we estimate the probabilities with the MLE, which is the common empirical expected value:

$$
\begin{aligned}
P(C_1) &= \frac{5}{8} & P(C_2) &= \frac{3}{8} \\
P(x_1 = 0 \mid C_1) &= \frac{5}{5} & P(x_1 = 1 \mid C_1) &= \frac{0}{5} \\
P(x_2 = 0 \mid C_1) &= \frac{4}{5} & P(x_2 = 1 \mid C_1) &= \frac{1}{5} \\
P(x_3 = 0 \mid C_1) &= \frac{2}{5} & P(x_3 = 1 \mid C_1) &= \frac{3}{5} \\
P(x_4 = 0 \mid C_1) &= \frac{4}{5} & P(x_4 = 1 \mid C_1) &= \frac{1}{5} \\
P(x_5 = 0 \mid C_1) &= \frac{3}{5} & P(x_5 = 1 \mid C_1) &= \frac{2}{5} \\
P(x_1 = 0 \mid C_2) &= \frac{1}{3} & P(x_1 = 1 \mid C_2) &= \frac{2}{3} \\
P(x_2 = 0 \mid C_2) &= \frac{1}{3} & P(x_2 = 1 \mid C_2) &= \frac{2}{3} \\
P(x_3 = 0 \mid C_2) &= \frac{3}{3} & P(x_3 = 1 \mid C_2) &= \frac{0}{3} \\
P(x_4 = 0 \mid C_2) &= \frac{1}{3} & P(x_4 = 1 \mid C_2) &= \frac{2}{3} \\
P(x_5 = 0 \mid C_2) &= \frac{2}{3} & P(x_5 = 1 \mid C_2) &= \frac{1}{3}
\end{aligned}
$$

2. The probability is the product of the priors and the posteriors of each feature:
   - First sample: $P(C_1|x) \propto P(C_1)P(x_1 = 1 \mid C_1)P(x_2 = 1 \mid C_1)P(x_3 = 0 \mid C_1)P(x_4 =$

$1 \mid C_1)P(x_5 = 0 \mid C_1) = \frac{5}{8} \cdot 0 \cdot \dots = 0$, therefore it belongs to C_2 for the trained NB model.

- Second sample: $P(C_2|x) \propto P(C_2)P(x_1 = 0 \mid C_2)P(x_2 = 1 \mid C_2)P(x_3 = 1 \mid C_2)P(x_4 = 0 \mid C_2)P(x_5 = 1 \mid C_2) = \frac{3}{8} \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot 0 \ldots = 0$, therefore it belongs to C_1 for the trained NB model.