Master M2 MVA - Computational Statistics

TP 1 : Reminder on Markov Chains
Stochastic Gradient Descent

Realized By: Eya GHAMGUI

Exercice 2: Invariant Distribution

We define a Markov chain $(Xn)_{n\geq 0}$ with values in $[0,1]$ as follows : given the current value $X_n (n\in\mathbb{N})$ of the chain,

- If $X_n = \frac{1}{m}$ (for some positive integer m), we let :

$$\begin{cases} X_{n+1} = \frac{1}{m+1} & \text{with probability} \quad 1 - X_n^2 \\ X_{n+1} \sim u([0,1]) & \text{with probability} \quad X_n^2 \end{cases}$$

- If not, $X_{n+1} \sim u([0,1])$

Question 4

Let $x = \frac{1}{m}$ with $m \geq 2$

a) Let $n \in \mathbb{N}^*$. Compute $P^n(x, \frac{1}{n+m})$ in terms of m and n.

- If $n = 1$:

$$\begin{aligned} P\,(x, \frac{1}{m+1}) &= P\,(\frac{1}{m}, \frac{1}{m+1}) \\ &= P\,(X_{n+1} = \frac{1}{m+1} | X_n = \frac{1}{m}) \\ &= 1 - X_n^2 = 1 - (\frac{1}{m})^2 \end{aligned}$$

- If $n = 2$:

$$\begin{aligned} P^2\,(x, \frac{1}{m+2}) &= P\,(P\,(x, \frac{1}{m+2})) \\ &= \int P\,(y, \frac{1}{m+2}) P\,(x, dy) \\ &= \int P\,(y, \frac{1}{m+2}) \left[x^2 \int_{dy\cap[0,1]} dt + (1-x^2)\ \ \delta_{\frac{1}{m+1}}(dy) \right] \end{aligned}$$

The last equality is resulted from the question 1 of the exercise, in the case when $x = \frac{1}{m}$ and δ_α is the Dirac measure at α.
We have also: $\int_{dy\cap[0,1]} dt = 0$ because this is an integral over $dy \cap [0,1]$.
We obtain:

$$\begin{aligned} P^2\,(x, \frac{1}{m+2}) &= \int P\,(y, \frac{1}{m+2})(1-x^2)\ \ \delta_{\frac{1}{m+1}}(dy) \\ &= (1-x^2) \int P\,(y, \frac{1}{m+2}) \delta_{\frac{1}{m+1}}(dy) \\ &= (1-(\frac{1}{m})^2) \int P\,(y, \frac{1}{m+2}) \delta_{\frac{1}{m+1}}(dy) \\ &= (1-(\frac{1}{m})^2) P\,(\frac{1}{m+1}, \frac{1}{m+2}) \end{aligned}$$

Using the previous result: $P\,(\frac{1}{m+1}, \frac{1}{m+2}) = 1 - (\frac{1}{m+1})^2$

Thus:

$$\begin{aligned} P^2\,(x, \frac{1}{m+2}) &= (1-(\frac{1}{m})^2)(1-(\frac{1}{m+1})^2) \\ &= \prod_{i=0}^{1} (1-(\frac{1}{m+i})^2) \end{aligned}$$

- Now, let's take:

$$P^n\,(x, \frac{1}{m+n}) = \prod_{i=0}^{n-1} (1-(\frac{1}{m+i})^2)$$

Let's suppose that this result is true until n, we want to show that it is true for $n+1$:

$$\begin{aligned} P^{n+1}\,(x, \frac{1}{m+n+1}) &= P\,(P^n\,(x, \frac{1}{m+n+1})) \\ &= \int P^n\,(y, \frac{1}{m+n+1}) P(x, dy) \\ &= \int P^n\,(y, \frac{1}{m+n+1}) \left[x^2 \int_{dy\cap[0,1]} dt + (1-x^2)\ \ \delta_{\frac{1}{m+1}}(dy) \right] \\ &= (1-(\frac{1}{m})^2) \int P^n\,(y, \frac{1}{m+n+1}) \delta_{\frac{1}{m+1}}(dy) \\ &= (1-(\frac{1}{m})^2) P^n\,(\frac{1}{m+1}, \frac{1}{m+n+1}) \end{aligned}$$

Let's take $m' = m+1$, Thus,

$$P^{n+1}\,(x, \frac{1}{m+n+1}) = (1-(\frac{1}{m})^2) P^n\,(\frac{1}{m'}, \frac{1}{m'+n})$$

Using the recurrence condition:

$$\begin{aligned} P^{n+1}\,(x, \frac{1}{m+n+1}) &= (1-(\frac{1}{m})^2)\ \ \prod_{i=0}^{n-1} (1-(\frac{1}{m'+i})^2) \\ &= (1-(\frac{1}{m})^2)\ \ \prod_{i=0}^{n-1} (1-(\frac{1}{m+i+1})^2) \\ &= (1-(\frac{1}{m})^2)\ \ \prod_{i=1}^{n} (1-(\frac{1}{m+i})^2) \\ &= \prod_{i=0}^{n} (1-(\frac{1}{m+i})^2) \end{aligned}$$

Using recurrence, we have shown that:

$$\boxed{P^n\,(x, \frac{1}{m+n}) = \prod_{i=0}^{n-1} (1-(\frac{1}{m+i})^2)}$$

b) Do we have $\lim_{n\to+\infty} P^n(x, A) = \pi(A)$ when $A = \bigcup_{q\in N} \left\{ \frac{1}{m+1+q} \right\}$?

- **First Part:**

π : the uniform distribution on [0,1].

Thus,

$$\begin{aligned} \pi(A) &= \int_{A\cap[0,1]} dt \\ &= \int_{\bigcup_{q\in N} (\frac{1}{m+1+q})\cap[0,1]} dt \\ &= \sum_{q\in\mathbb{N}} \int_{\frac{1}{m+1+q}\cap[0,1]} dt \\ &= 0 \end{aligned}$$

- **Second Part:**

$$\begin{aligned} P^n\,(x, A) &= P^n\,(x, \bigcup_{q\in N} (\frac{1}{m+1+q})) \\ &= \sum_{q\in\mathbb{N}} P^n\,(x, \frac{1}{m+1+q}) \\ &= \sum_{q\in\mathbb{N}\,,\, q\neq n-1} P^n\,(x, \frac{1}{m+1+q}) + P^n\,(x, \frac{1}{m+n}) \geq P^n\,(x, \frac{1}{m+n}) \end{aligned}$$

We have:

$$P^n\,(x, \frac{1}{m+n}) = \prod_{i=0}^{n-1} (1-(\frac{1}{m+i})^2)$$

Also,

$$\begin{aligned} m \geq 2 &\Longrightarrow m+i \geq 2+i \\ &\Longrightarrow 1-(\frac{1}{m+i})^2 \geq 1-(\frac{1}{i+2})^2 \\ &\Longrightarrow \prod_{i=0}^{n-1} (1-(\frac{1}{m+i})^2) \geq \prod_{i=0}^{n-1} (1-(\frac{1}{i+2})^2) \end{aligned}$$

Then,

$$\begin{aligned} P^n\,(x, \frac{1}{m+n}) \geq \prod_{i=0}^{n-1} (1-(\frac{1}{i+2})^2) &= \prod_{i=0}^{n-1} (\frac{(2+i)^2-1}{(2+i)^2}) \\ &= \prod_{i=0}^{n-1} (\frac{(i+1)(i+3)}{(i+2)^2}) \\ &= \prod_{i=0}^{n-1} \frac{i+1}{i+2} \frac{i+3}{i+2} \\ &= \prod_{i=0}^{n-1} \frac{i+1}{i+2} \prod_{i=0}^{n-1} \frac{i+3}{i+2} \\ &= \prod_{i=1}^{n} \frac{i}{i+1} \prod_{i=2}^{n+1} \frac{i+1}{i} \\ &= \frac{1}{2} \frac{n+2}{n+1} \\ &= \frac{1}{2}(1+\frac{1}{n+1}) \end{aligned}$$

$$\begin{aligned} &\Longrightarrow P^n\,(x, A) \geq \frac{1}{2}(1+\frac{1}{n+1}) \\ &\Longrightarrow \lim_{n\to+\infty} P^n(x, A) \geq \lim_{n\to+\infty} \frac{1}{2}(1+\frac{1}{n+1}) = \frac{1}{2} \\ &\Longrightarrow \lim_{n\to+\infty} P^n(x, A) \geq \frac{1}{2} \end{aligned}$$

- **Conclusion:**

$$\boxed{\lim_{n\to+\infty} P^n(x, A) \neq \pi(A)}$$

Exercice 3: Stochastic Gradient Learning in Neural Networks

Import Libraries

In [1]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import random

sns.set()
```

Question 1

Describe the stochastic gradient descent algorithm for minimizing the empirical risk and implement it.

The goal of optimization algorithms is to minimize the risk function $R(w)$. Thus, we can note the optimization problem as:

$$\underset{w}{Min}\ \ R(w) \quad \equiv \quad \underset{w}{Min}\ \ E_z[J(w,z)] \quad \equiv \quad \underset{w}{Min} \quad \int (y - w^t x)^2\, \mathrm{dP}(x,y)$$

To solve this problem, the stochastic gradient descent algorithm is good solution. It is an extension of the gradient descent algorithm. The intuition behind this method is that by following a descent direction in expectation, we are able to get close to the optimal solution. That is, this method calculate the gradient using just a random small part of observations instead of all of them. That's why, we can say that this approach can reduce the computational time when compared to the gradient descent. In our case, we suppose that we have independent input-output samples $\{z_i = (x_i, y_i)\}_{i=1}^n$. Instead of using the continous expression of the risk, we will define the empirical risk. The problem is then defined as:

$$\underset{w}{Min}\ \ R_n(w) \quad \equiv \quad \underset{w}{Min}\ \ \frac{1}{n} \sum_{i=1}^{n} (y_i - w^t x_i)^2$$

The stochastic gradient descent algorithm will be as follow:

> - Start from an initial vector $w_0 \in \mathbb{R}^d$
> - At each step $k = 0, 1, 2, \dots n_{iter}$:
>   - Choose a step size $\epsilon_k > 0$ and random index $i \in \mathbb{N}$
>   - Set $w_{k+1} = w_k - \epsilon_k \nabla_w R(f(x_i, w), y_i)$

Here, we have chosen the end condition to be the total number of iteration and $\epsilon_k = \frac{1}{k^\alpha}$ with $\alpha \in]\frac{1}{2}, 1[$.

Now, we should calculate the value of the gradient. We have:

$R(f(x_i, w), y_i) = (y_i - w^t x_i)^2$

$$\begin{aligned} \Longrightarrow \nabla_w R(f(x_i, w), y_i) &= \nabla_w (y_i - w^t x_i)^2 \\ &= -2(y_i - w^t x_i) x_i \end{aligned}$$

Thus, the update equation will be as follow:

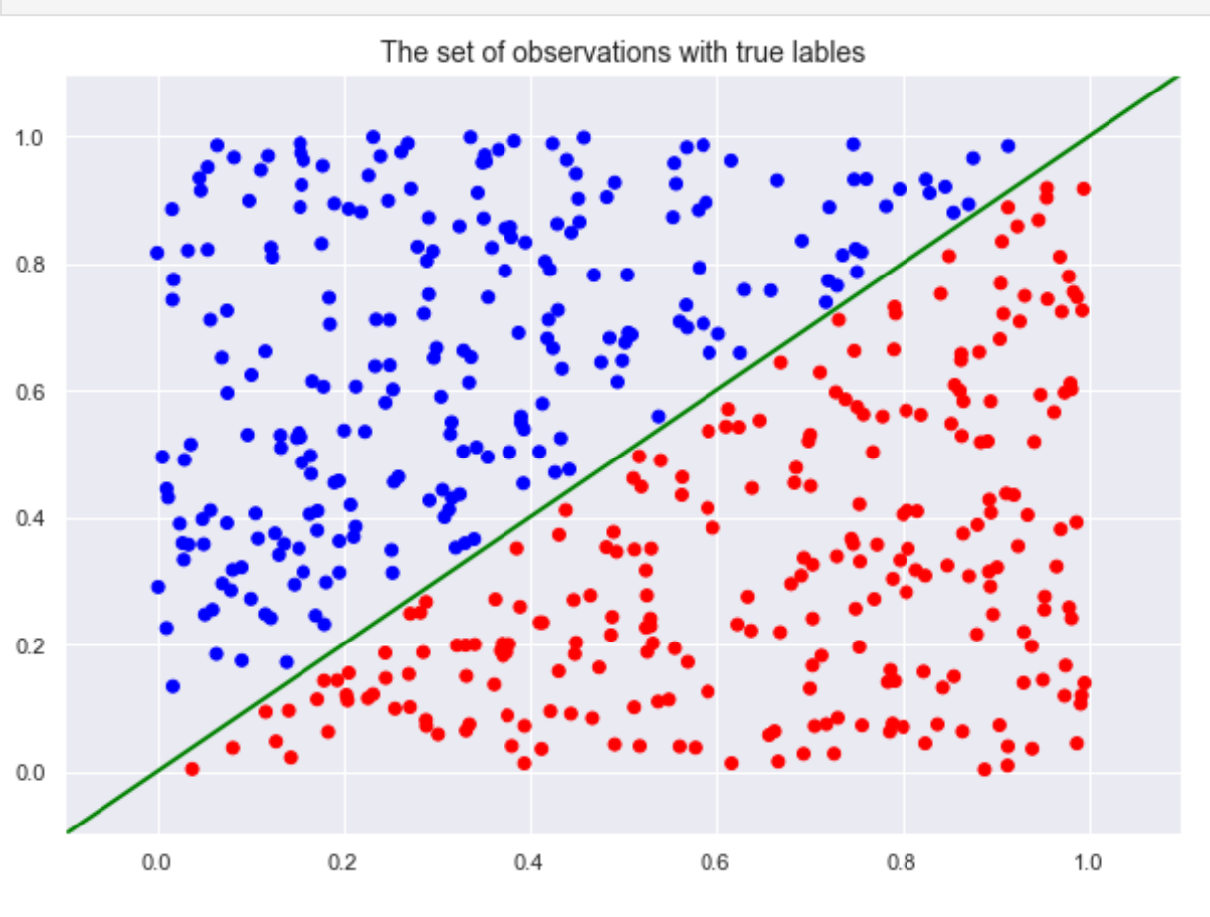
$$\boxed{w_{k+1} = w_k + 2\epsilon_k (y_i - w^t x_i) x_i = w_k + \frac{2}{k^\alpha}(y_i - w^t x_i) x_i}$$

In [2]:
```
# Implementation of Stochastic Gradient Descent

def SGD ( x , y , w_0 , alpha , n_iter ):

    N = y.shape[0]
    n = x.shape[0]
    w_k = w_0

    for k in range(1,n_iter+1):

        i = np.random.randint(N)
        xx = x[:,i].reshape((n,1))

        w_k1 = w_k + 2 * ((1/k)**alpha) * (y[i] - np.dot(w_k.T , xx)) * xx
        w_k = w_k1

    return (w_k1)
```

Question 2

Sample a set of observations $\{z_i\}_{i=1}^n$ by generating a collection of random points x_i of $\mathbb{R}^2$, $\bar{w} \in \mathbb{R}^2$ seen as the normal vector of an hyperplane, a straight line here, and assigning the label y_i according to the side of the hyperplane the point x_i is.

In [3]:
```
# Choose the parameters

## The number of samples
n=500

## We have chosen the expression of the hyperplane as y = a x + b = x
a = 1
b = 0

# Calculate the normal vector of the hyperplane
w = np.array([[-1],[1]])
w = w / np.sqrt(np.dot(w.T ,w))

# Generate randomly the set of observations
x = [random.uniform(0,1) for i in range(n)]
y = [random.uniform(0,1) for i in range(n)]

# Create the Dataset
x1 = []
y1 = []

for i in range(n):
    if (y[i] - a * x[i] - b > 0.02) or (y[i] - a * x[i] - b < - 0.02) :
        x1.append(x[i])
        y1.append(y[i])

n = len(x1)
x = x1
y = y1

X = np.array ([x,y])
Y = np.array([1 if y[i] - a * x[i] - b > 0 else -1 for i in range(n)])

# Plot the set of observations with their true labels
plt.figure(figsize=(10,7))

colors = ['blue' if y[i] - a * x[i] - b > 0 else 'red' for i in range(n) ] ## Generate colors
plt.scatter(x, y, c=colors) ## The observations
plt.plot([-1, 2], [-1* a + b, 2 * a + b], c='green', linewidth =2) ## Plot the hyperplane

plt.title("The set of observations with true lables" , fontsize = 14)
plt.xlim((-0.1,1.1))
plt.ylim((-0.1,1.1))
plt.show()
```



Interpretations:

> The green line corresponds to the hyperplane chosen to separate the data into two classes. The blue points correspond to the observations with the label $y_i = 1$. They are located above the hyperplane. The red points below the hyperplane correspond to the observation with the label $y_i = -1$.

Question 3

Test the algorithm you wrote at the first question over these observations. What is the vector w^* estimated? Is it far from $\bar{w}$?

1. Calculate the estimated vector

In [4]:
```
# Create an initial random vector w0
w_0 = np.random.rand(2,1)

# Initiate the parameters
alpha = 0.6     ## This value is chosen because it gives better performance
n_iter = 1000   ## This value is chosen randomly

# Calculate the estimated vector using Stochastic Gradient Descent method
w_est = SGD (X, Y, w_0 , alpha , n_iter)

print('The estimated vector :' )
print(w_est)
```

```
The estimated vector :
[[-2.01279254]
 [ 1.93994036]]
```

2. Compute the distance between the normal vector and the estimated vector

In [5]:
```
# Normalize the estimated vector to bring it in the same scale as the normal vector
w_est = w_est / np.sqrt(np.dot(w_est.T , w_est))

# Compute the distance between vectors
d =  np.linalg.norm(w-w_est)

print('The euclidean distance between the two vectors = ', d)
```

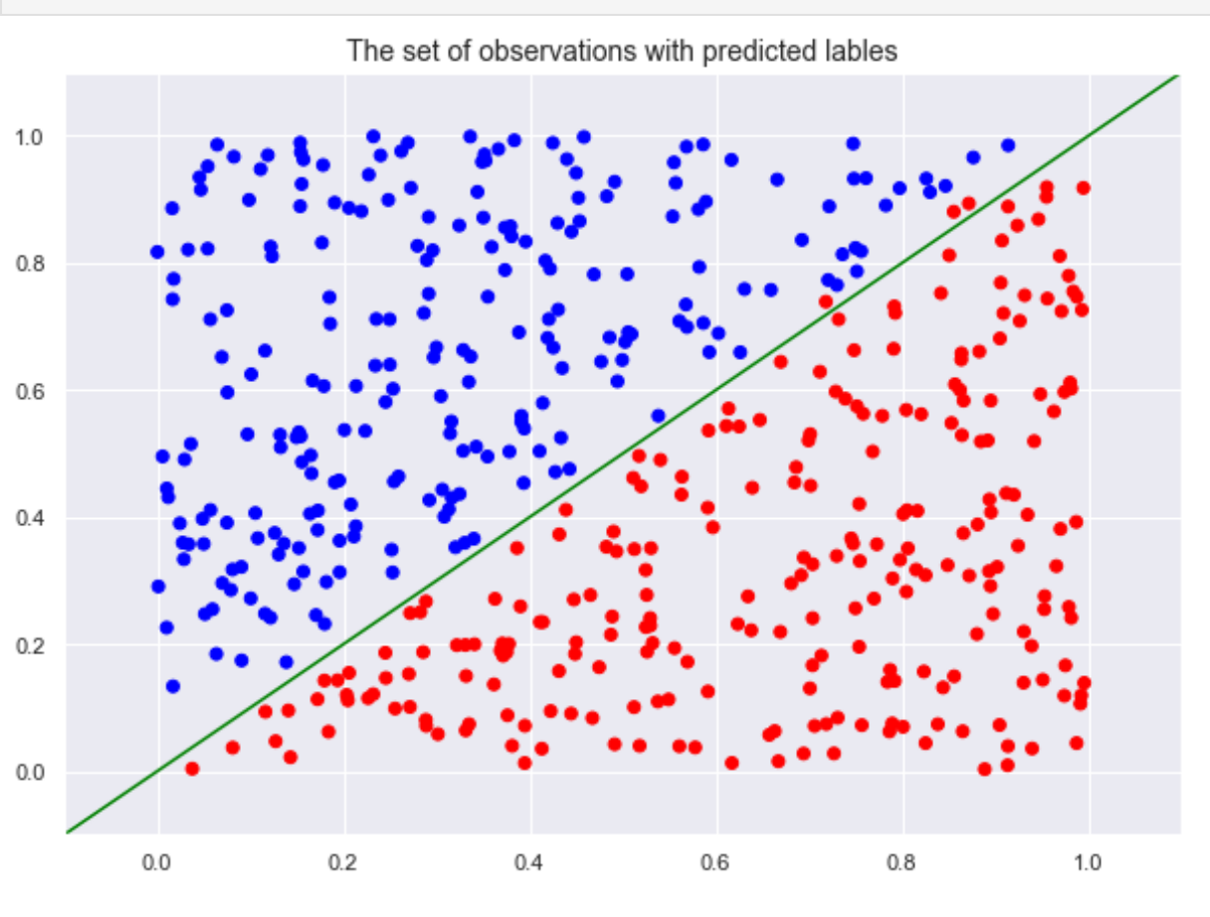
```
The euclidean distance between the two vectors =  0.018428491984333956
```

Interpretations:

> From the previous results, we can say that the Stochastic Gradient Descent algorithm succeeded in estimating the normal vector of the separation hyperplane. Indeed, the Euclidean distance between these two vectors is arround 0.018, which is considered small. We can say that the estimated vector w^* is not far from the value $\bar{w}$.
>
> $\Longrightarrow$ We conclude that the Stochastic Gradient Descent algorithm converges to the optimal solution.

3. Plot the labels predicted

In [6]:
```
# Calculate the predicted labels
y_est = np.dot(w_est.T , X).T

# Plot the predicted labes
plt.figure(figsize=(10,7))

colors_est = ['blue' if  y_est[i] > 0 else 'red' for i in range(n) ]
plt.scatter(x,y, c=colors_est)
plt.plot([-1, 2], [-1* a + b, 2 * a + b], c='green')

plt.title('The set of observations with predicted lables', fontsize = 14)
plt.xlim((-0.1,1.1))
plt.ylim((-0.1,1.1))
plt.show()
```



Interpretations:

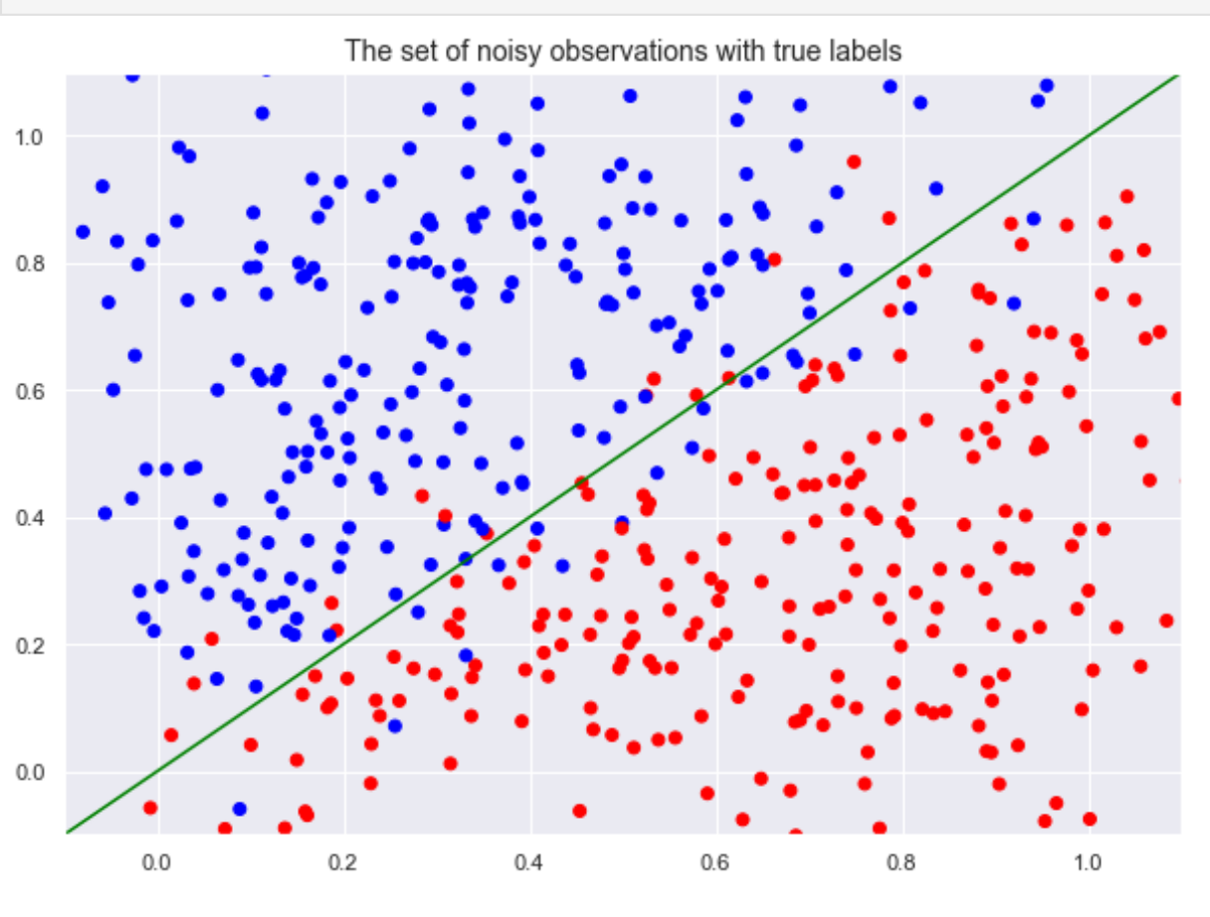
> From the previous plot, we can notice that almost all the predicted labels are correct. There is only a small number of errors, i.e. the red points are above the green line. We also remark that the errors made are only for observations close to the hyperplane. I think this is due to the small difference between the normal vector and the predicted vector.

Question 4

Noise your observations $\{z_i\}_{i=1}^n$ with an additive Gaussian noise and perform the optimisation again. Compare with the result of question three.

1. Add Gaussian noise to the observations

In [7]:
```
# Generate Gaussian noise
eps = np.array(np.random.normal(0,0.1,(2,n)))

# Add the noise to the observations
X_noisy = X + eps

# Plot the noisy observations
plt.figure(figsize=(10,7))

plt.plot([-1, 2], [-1* a + b, 2 * a + b], c='green')
plt.scatter(X_noisy[0,:], X_noisy[1,:], c=colors)

plt.title("The set of noisy observations with true labels" , fontsize = 14)
plt.xlim((-0.1,1.1))
plt.ylim((-0.1,1.1))

plt.show()
```



2. Estimate the normal vector of the hyperplane

In [8]:
```
# Create an initial random vector w0
w_0 = np.random.rand(2,1)

# Initiate the parameters
alpha = 0.6
n_iter = 1000

# Calculate the estimated vector using Stochastic Gradient Descent method
w_est = SGD (X_noisy, Y, w_0 , alpha , n_iter)

print('The estimated vector :' )
print(w_est)
```

```
The estimated vector :
[[-1.70148396]
 [ 1.71887094]]
```

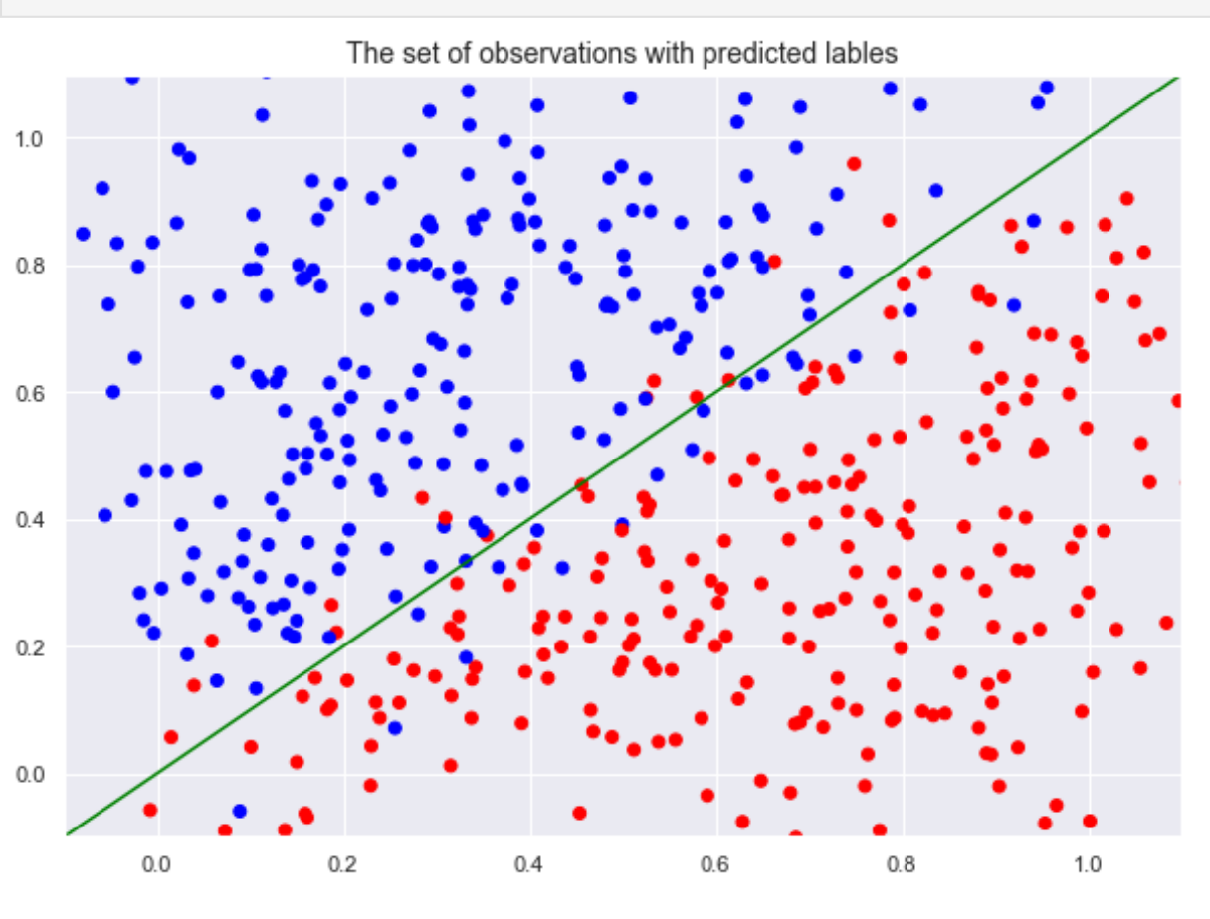
3. Compute the distance between the normal vector and the estimated vector

In [9]:
```
# Normalize the estimated vector to bring it in the same scale as the normal vector
w_est = w_est / np.sqrt(np.dot(w_est.T , w_est))

print('The euclidean distance between the two vectors = ', np.linalg.norm(w-w_est))
```

```
The euclidean distance between the two vectors =  0.005083335281238399
```

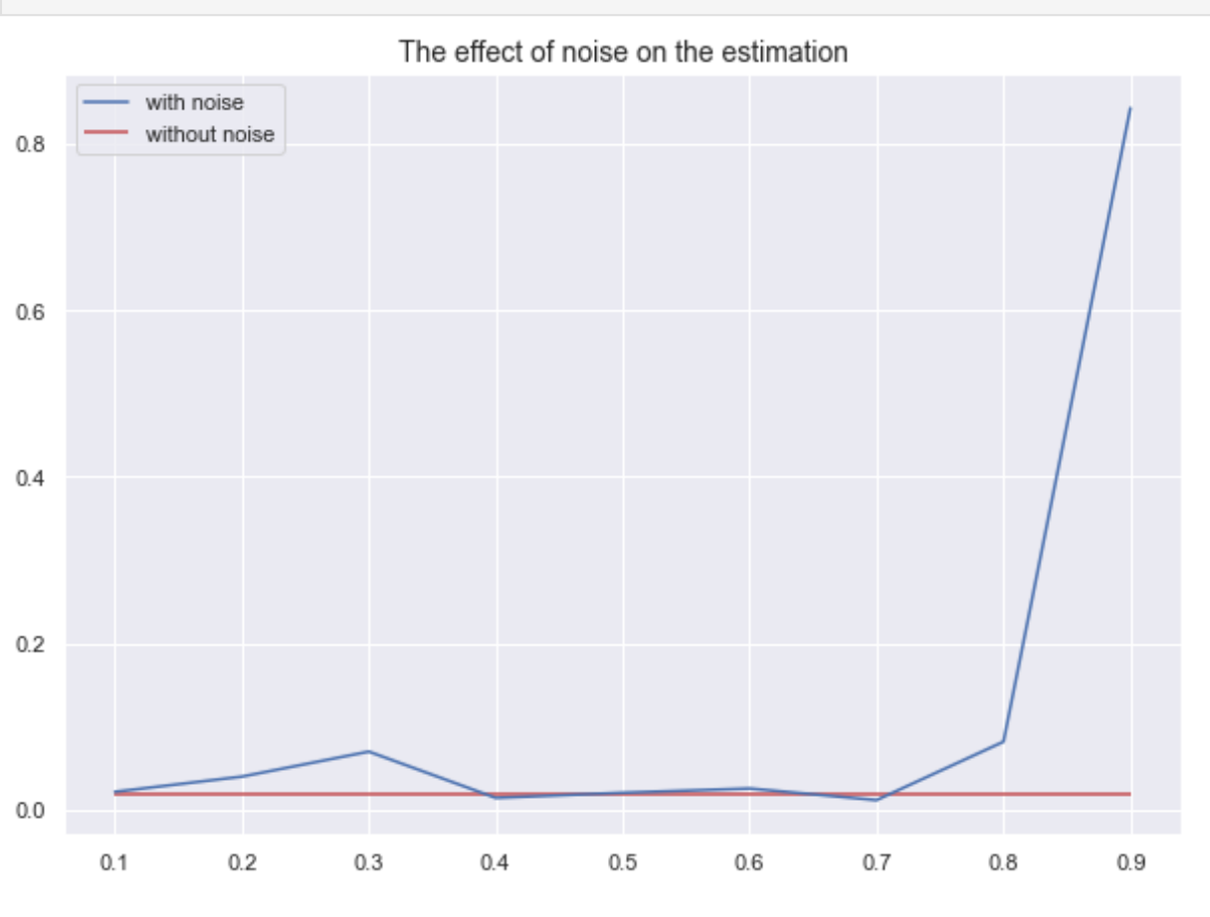
4. Plot the labels predicted

In [10]:
```
# Calculate the predicted labels
y_est = np.dot(w_est.T , X).T

# Plot the predicted labes
plt.figure(figsize=(10,7))

colors_est = ['blue' if  y_est[i] > 0 else 'red' for i in range(n) ]
plt.scatter(X_noisy[0,:], X_noisy[1,:], c=colors_est)
plt.plot([-1, 2], [-1* a + b, 2 * a + b], c='green')

plt.title('The set of observations with predicted lables', fontsize = 14)
plt.xlim((-0.1,1.1))
plt.ylim((-0.1,1.1))
plt.show()
```



5. Study the effect of noise on the estimate

In [11]:
```
# Here, we will plot the distance between the normal vector and the estimated vector as a function of noise

noise = np.arange(0.1,1,0.1)
dist = []

for i in noise :
    eps = np.array(np.random.normal(0,i,(2,n)))
    X_noisy = X + eps

    w_est = SGD (X_noisy, Y, w_0 , alpha , n_iter)
    w_est = w_est / np.sqrt(np.dot(w_est.T , w_est))

    dist.append(np.linalg.norm(w-w_est))

plt.figure(figsize=(10,7))
plt.title("The effect of noise on the estimation" , fontsize = 14)
plt.plot(noise, dist,label ='with noise' )
plt.hlines(y=d, xmin=0.1, xmax=0.9, color='r' ,label='without noise')
plt.legend()
plt.show()
```



Interpretations and Comparison:

> From the previous graph, we can notice that the distance between the two vectors is almost constant for noise values less than 0.7 and increases for higher noise values.
>
> For a small value of noise, the algorithm gives an estimate close to the normal vector in term of distance. We can say that this algorithm remains robust to noise and correctly estimates the parameters of the hyperplane. Moreover, these values are very close to the distance of the estimate using the noiseless data.
>
> We can also notice that for the value of noise equal to 0.4 and 0.7, the algorithm performs better than with the data without noise. Thus, even if the data is noisy, this noise helps the algorithm to get closer to the optimal solution.
>
> For large values of noise, the algorithm gives estimates with a large distance to the normal vecor. In this case, the estimate is worse than the result without noise. Thus, the noise has an effect on the estimate and the algorithm start to diverge.

Question 5

Test the algorithm on the Breast Cancer Wisconsin (Diagnostic) Data Set [WSM95]:

http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29 .

1. Data Retrieval

In [12]:
```
# Read the data
data = pd.read_csv("wdbc.data" ,index_col=False ,header = None)

# Extract observations and labels
x = data.iloc[:,2:]
y = (2 * (data.iloc[:,1]=='M').astype('int') -1 ).values

# Split the data into training and test data
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=0)

# Normalize the data
scaler = Normalizer().fit(x_train)
x_train= scaler.transform(x_train)
x_test = scaler.transform(x_test)

x_train = x_train.T
x_test = x_test.T
```

2. Estimate the vector of the separating hyperplane

In [13]:
```
# Initiate the parameters
n_parameters = x_train.shape[0]
w_0 = np.random.rand(n_parameters,1)
alpha = 0.6
n_iter = 1000

# Calculate the estimated vector using Stochastic Gradient Descent method
w_est = SGD (x_train, y_train, w_0 , alpha , n_iter)
w_est = w_est / np.sqrt(np.dot(w_est.T , w_est))
```

3. Predict test data

In [14]:
```
y_pred = np.dot(w_est.T , x_test).reshape((y_test.shape[0],))
y_pred = np.where(y_pred > 0,1,-1)
```

4. Performance of the algorithm

In [15]:
```
print('The accuracy of the algorithm = ' , np.round(accuracy_score(y_test, y_pred)*100, 3),' %')
```

```
The accuracy of the algorithm =  69.591  %
```

Interpretation:

> This algorithm gives a good result on this dataset. In fact, the accuracy of the algorithm is arround 69.591%