# MSE - T-MachLe - PW04

October 11, 2017

#### 0.0.1 MSE - T-MachLe

# 1 PW 04

{dorian.magnin, noemien.kocher}@master.hes-so.ch

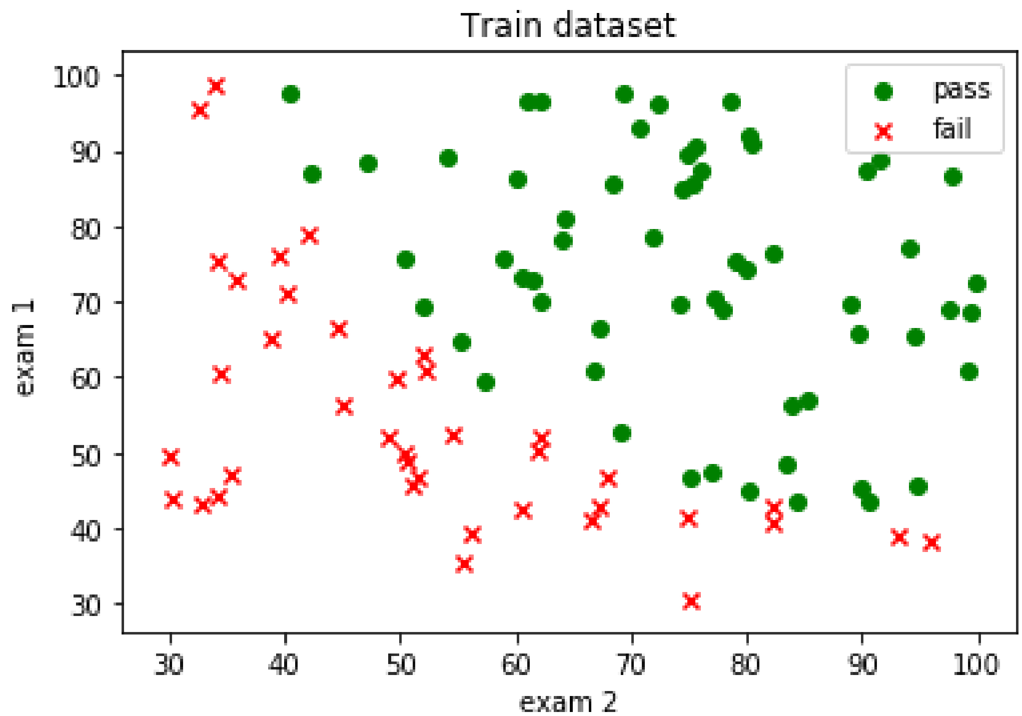
## 1.1 Exercice 1 Classification system

### 1.1.1 a. Getting started

**a) + b)**

In [158]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from numpy.linalg import inv

dataset = pd.read_csv('ex1-data-train.csv',
                      header=0,
                      names=['x1', 'x2', 'y'])

x1 = dataset['x1'].values
x2 = dataset['x2'].values
y = dataset['y'].values
N = len(x1)

ones_x1 = [x1[i] for i in range(0, N) if y[i] == 1]
ones_x2 = [x2[i] for i in range(0, N) if y[i] == 1]

zero_x1 = [x1[i] for i in range(0, N) if y[i] == 0]
zero_x2 = [x2[i] for i in range(0, N) if y[i] == 0]

plt.scatter(ones_x1, ones_x2, marker="o", label="pass", color="green")
plt.scatter(zero_x1, zero_x2, marker="x", label="fail", color="red")
plt.legend()
plt.title("Train dataset")
plt.ylabel("exam 1")
```

```
plt.xlabel("exam 2")
plt.show()
```




**c)**

```
In [159]: import random

          def blindTest(x1, x2):
              test = random.random()
              return 1 if test > 0.5 else 0
```

**d)**

```
In [160]: true_guess = 0
          for i in range(0, len(x1)):
              guess = blindTest(x1[i], x2[i])
              if guess == y[i]:
                  true_guess = true_guess + 1

          print("Performance: ", true_guess/N)

Performance:  0.42424242424242425
```

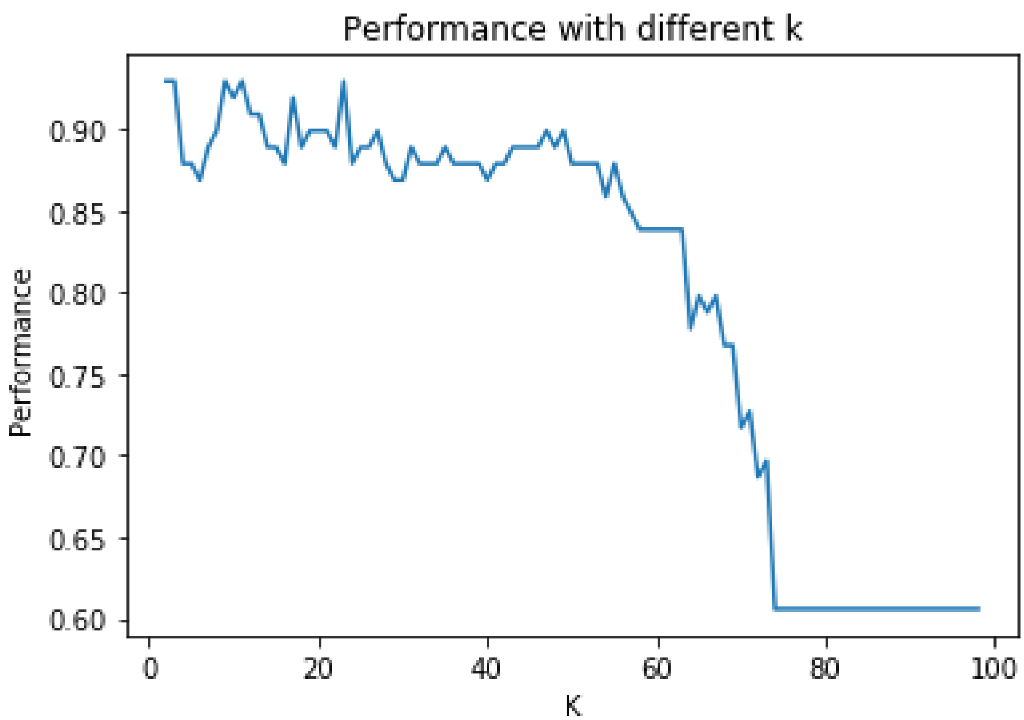
### 1.1.2 b. K-nn classifier

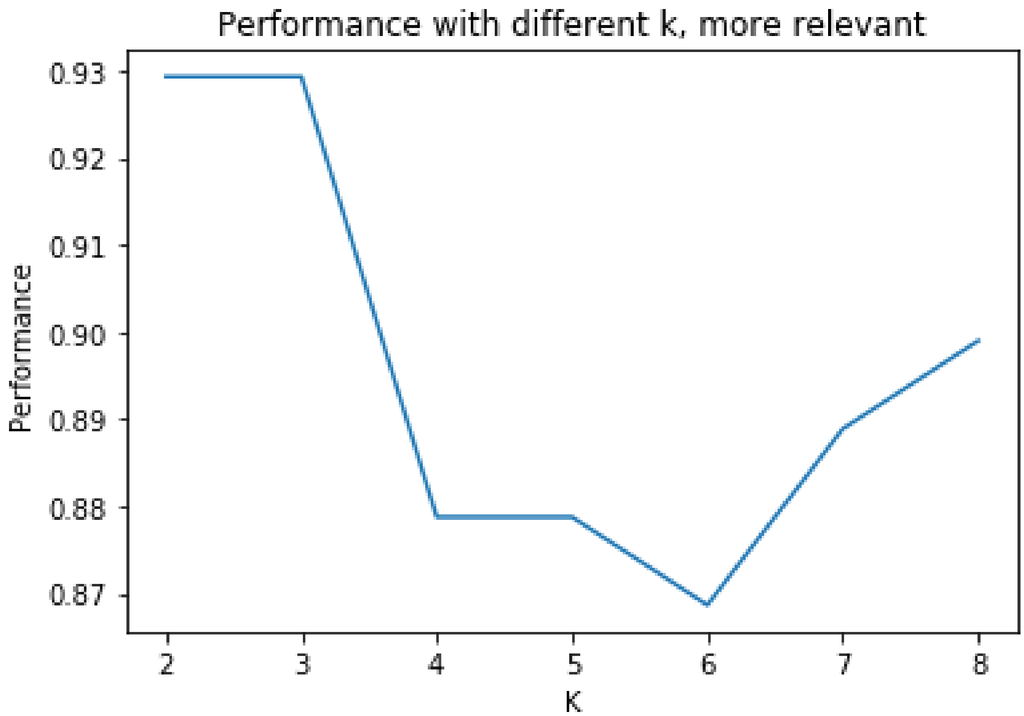
Nous remarquons que les meilleurs résultats sont avec un `k=2` ou `k=3`. Comme critère en cas d'égalité, nous prenons la catégorie du point le plus proche.

Pour le fun, nous avons fait varié le k jusqu'à `N`. On remarque que jusqu'environ 70 la performance est stable.

```
In [165]: from sklearn.neighbors import NearestNeighbors

          X = []
          for i in range(0, N):
              X.append([x1[i], x2[i]])

          def distEucl(point1, point2):
              return np.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)


          max = N-1

          dists, indices = NearestNeighbors(n_neighbors=max+1,
                                            algorithm='ball_tree',
                                            metric="euclidean").fit(X).kneighbors(X)


          measures = []
          ks = []

          for k in range(2,max+1):

              true_guess = 0

              # Pour chaque point
              for i in range(0, N):
                  fails = 0
                  passs = 0
                  # Pour chaque voisin du point
                  for neighborI in indices[i][1:k]:
                      if y[neighborI] == 0:
                          fails = fails + 1
                      else:
                          passs = passs + 1
                  if fails > passs:
                      knnresult = 0
                  elif fails == passs: # On pourrait choisir un autre critère
                      knnresult = y[indices[i][1]]
                  else:
                      knnresult = 1
                  if knnresult == y[i]:
```

```
            true_guess = true_guess + 1

        measures.append(true_guess/N)
        ks.append(k)
        #print("With k =", k)
        #print("Performance: ", true_guess/N)

    #print("ks:", ks)
    #print("measures: ", measures)
    plt.plot(ks, measures)
    plt.title("Performance with different k")
    plt.xlabel("K")
    plt.ylabel("Performance")
    plt.show()

    plt.plot(ks[:7], measures[:7])
    plt.title("Performance with different k, more relevant")
    plt.xlabel("K")
    plt.ylabel("Performance")
    plt.show()
```






### 1.1.3 c. Bayes - Histogram

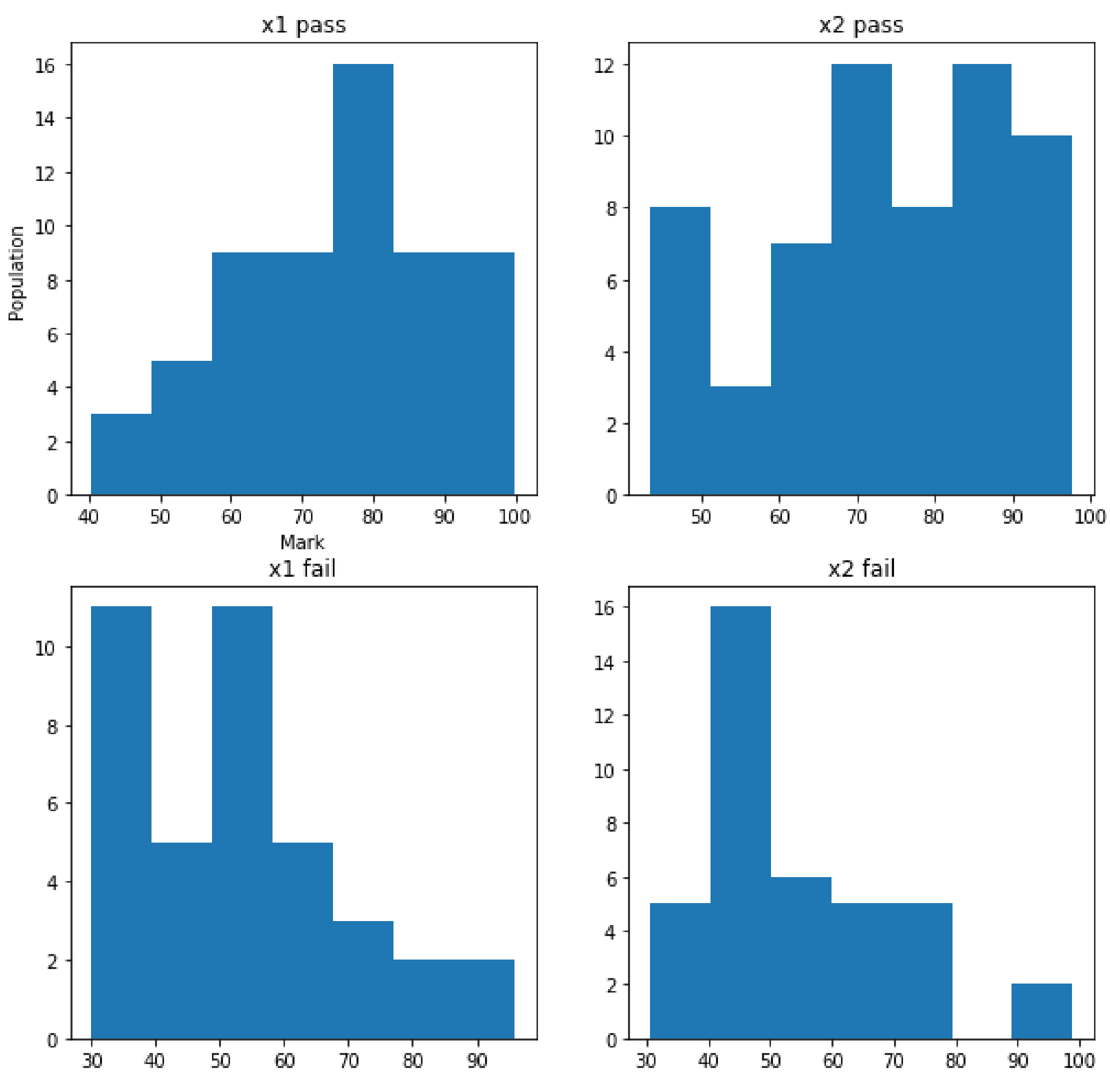
**a) Compute the priors of both classes P(C0) and P(C1).**

```
In [166]: n_pass = 0
          n_fail = 0
          for i in range(0, N):
              if y[i] == 0:
                  n_fail = n_fail + 1
              if y[i] == 1:
                  n_pass = n_pass + 1

          prior_pass = n_pass/N
          prior_fail = n_fail/N

          print("Prior pass: ", prior_pass)
          print("Prior fail: ", prior_fail)
```

```
Prior pass:  0.6060606060606061
Prior fail:  0.3939393939393939
```

**b) Compute histograms of x1 and x2 for each class (total of 4 histograms). Plot these histograms. Advice : use the numpy histogram(a,bins='auto') function.**

In [168]:
```
f1, ((ax1, ax2),(ax3, ax4)) = plt.subplots(2, 2, figsize=(10,10))

ax1.hist(ones_x1, bins='auto')
ax1.set_title("x1 pass")
ax1.set_ylabel("Population")
ax1.set_xlabel("Mark")
ax2.hist(ones_x2, bins='auto')
ax2.set_title("x2 pass")
ax3.hist(zero_x1, bins='auto')
ax3.set_title("x1 fail")
ax4.hist(zero_x2, bins='auto')
ax4.set_title("x2 fail")


plt.show()
```



**c) Use the histograms to compute the likelihoods p(x1|C0), p(x1|C1), p(x2|C0) and p(x2|C1). For this define a function likelihoodHist(x,histValues,edgeValues) that returns the likelihood of x for a given histogram (defined by its values and bin edges as returned by the numpy histogram() function).**

```
In [169]: def likelihoodHist(x, histValues, edgeValues):
              for i in range(0, len(histValues)):
                  if edgeValues[i] == x:
                      return histValues[i]
                  if edgeValues[i] >= x:
                      return histValues[i-1]
              return 0


          #histValues, edgeValues = np.histogram(ones_x1, bins='auto')
          #for i in range(0, N):
          #    print(likelihoodHist(x1[i], histValues, edgeValues))
```

**d) Implement the classification decision according to Bayes rule and compute the performance of the system on the training set :** — using only feature x1
— using only feature x2
— using x1 and x2 making the naive Bayes hypothesis of feature independence, i.e. `p(X|Ck) = p(x1|Ck) ů p(x2|Ck)`
Which system is the best ?

```
In [170]: print("Only using feature 1")

          true_guess = 0
          for i in range(0, N):
              histValues, edgeValues = np.histogram(zero_x1)
              prob_fail = likelihoodHist(x1[i], histValues, edgeValues) * prior_fail

              histValues, edgeValues = np.histogram(ones_x1)
              prob_pass = likelihoodHist(x1[i], histValues, edgeValues) * prior_pass

              knnresult = 0
              if prob_pass > prob_fail:
                  knnresult = 1

              if knnresult == y[i]:
                  true_guess = true_guess + 1


          print("Performance: ", true_guess/N)
```

```python
# ----------

print("Using feature 2")

true_guess = 0
for i in range(0, N):
    histValues, edgeValues = np.histogram(zero_x2)
    prob_fail = likelihoodHist(x2[i], histValues, edgeValues) * prior_fail

    histValues, edgeValues = np.histogram(ones_x2)
    prob_pass = likelihoodHist(x2[i], histValues, edgeValues) * prior_pass

    knnresult = 0
    if prob_pass > prob_fail:
        knnresult = 1

    if knnresult == y[i]:
        true_guess = true_guess + 1


print("Performance: ", true_guess/N)


# ----------

print("Using 2 features")

true_guess = 0
for i in range(0, N):
    histValues, edgeValues = np.histogram(zero_x1)
    prob_fail_x1 = likelihoodHist(x1[i], histValues, edgeValues) * prior_fail

    histValues, edgeValues = np.histogram(ones_x1)
    prob_pass_x1 = likelihoodHist(x1[i], histValues, edgeValues) * prior_pass

    histValues, edgeValues = np.histogram(zero_x2)
    prob_fail_x2 = likelihoodHist(x2[i], histValues, edgeValues) * prior_fail

    histValues, edgeValues = np.histogram(ones_x2)
    prob_pass_x2 = likelihoodHist(x2[i], histValues, edgeValues) * prior_pass

    knnresult = 0
    if prob_pass_x1 * prob_pass_x2 > prob_fail_x1 * prob_fail_x2:
        knnresult = 1

    if knnresult == y[i]:
        true_guess = true_guess + 1
```

```
        print("Performance: ", true_guess/N)
```

```
Only using feature 1
Performance:  0.5959595959595959
Using feature 2
Performance:  0.7171717171717171
Using 2 features
Performance:  0.5858585858585859
```

On remarque que le meilleurs système est celui avec la feature x2.

### 1.1.4 c. Bayes - Univariate Gaussian distribution

```
In [171]: def likelihood(x, mu, sig):
              sig = sig * sig
              result = (1.0 / np.sqrt(2 * np.pi * sigma)) * \
                     np.exp( -( (x-mu)*(x-mu)/(2*sig) ) )

              return result



          print("Only using feature 1")

          true_guess = 0
          for i in range(0, N):

              prob_fail = likelihood(x1[i], np.mean(zero_x1), np.std(zero_x1)) * prior_fail

              prob_pass = likelihood(x1[i], np.mean(ones_x1), np.std(ones_x1)) * prior_pass

              knnresult = 0
              if prob_pass > prob_fail:
                  knnresult = 1

              if knnresult == y[i]:
                  true_guess = true_guess + 1


          print("Performance: ", true_guess/N)


          # ----------

          print("Using feature 2")
```

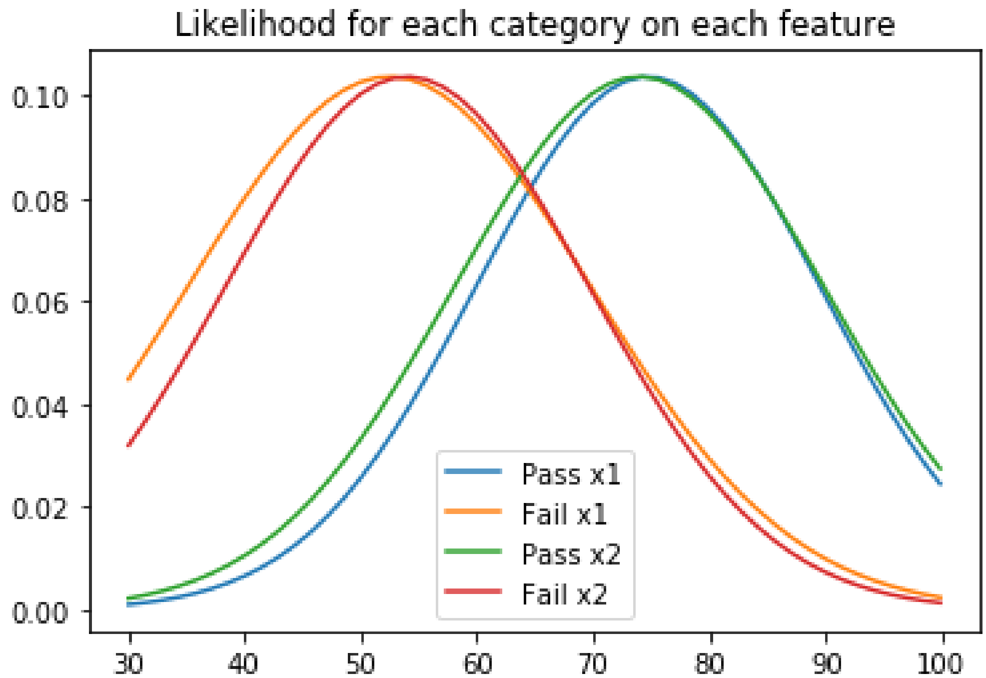
```python
true_guess = 0
for i in range(0, N):

    prob_fail = likelihood(x2[i], np.mean(zero_x2), np.std(zero_x2)) \
                * prior_fail

    prob_pass = likelihood(x2[i], np.mean(ones_x2), np.std(ones_x2)) \
                * prior_pass

    knnresult = 0
    if prob_pass > prob_fail:
        knnresult = 1

    if knnresult == y[i]:
        true_guess = true_guess + 1


print("Performance: ", true_guess/N)


# ----------

print("Using 2 features")

true_guess = 0
for i in range(0, N):

    prob_fail_x1 = likelihood(x1[i], np.mean(zero_x1), np.std(zero_x1)) \
                   * prior_fail

    prob_pass_x1 = likelihood(x1[i], np.mean(ones_x1), np.std(ones_x1)) \
                   * prior_pass

    prob_fail_x2 = likelihood(x2[i], np.mean(zero_x2), np.std(zero_x2)) \
                   * prior_fail

    prob_pass_x2 = likelihood(x2[i], np.mean(ones_x2), np.std(ones_x2)) \
                   * prior_pass

    knnresult = 0
    if prob_pass_x1 * prob_pass_x2 > prob_fail_x1 * prob_fail_x2:
        knnresult = 1

    if knnresult == y[i]:
        true_guess = true_guess + 1


print("Performance: ", true_guess/N)
```

```
Only using feature 1
Performance:  0.7878787878787878
Using feature 2
Performance:  0.7575757575757576
Using 2 features
Performance:  0.9292929292929293
```

Les résultats sont encourageants.

```
In [172]: import matplotlib.mlab as mlab
          from scipy.stats import norm
          (mu, sigma) = norm.fit(ones_x1)

          print("Mu: ", mu, " - Sigma: ", sigma)

          mu = np.average(ones_x1)
          sigma = np.std(ones_x1)

          print("Mu: ", mu, " - Sigma: ", sigma)

          x = np.linspace(np.min(x1), np.max(x1))
          y = [likelihood(x1, mu, sigma) for x1 in x]
          plt.plot(x,y, label="Pass x1")

          y = [likelihood(x1, np.average(zero_x1), np.std(zero_x1)) for x1 in x]
          plt.plot(x,y, label="Fail x1")

          y = [likelihood(x1, np.average(ones_x2), np.std(ones_x2)) for x1 in x]
          plt.plot(x,y, label="Pass x2")

          y = [likelihood(x1, np.average(zero_x2), np.std(zero_x2)) for x1 in x]
          plt.plot(x,y, label="Fail x2")

          #plt.hist(ones_x1, bins='auto')
          plt.legend()
          plt.title("Likelihood for each category on each feature")
          plt.show()
```

```
Mu:  74.7189226966  - Sigma:  14.7876272134
Mu:  74.7189226966  - Sigma:  14.7876272134
```


Likelihood for each category on each feature
0.10
0.08
0.06
0.04
0.02
0.00
30
40
50
60
70
80
90
100
Pass x1
Fail x1
Pass x2
Fail x2