zumz3a

April 12, 2018

0.1 Uczenie maszynowe UMZ 2017/2018

1 3. Naiwny klasyfikator bayesowski, drzewa decyzyjne

1.0.1 Cz 1

1.1 3.1. Naiwny klasyfikator bayesowski

- Naiwny klasyfikator bayesowski jest algorytmem dla problemu klasyfikacji wieloklasowej.
- Naszym celem jest znalezienie funkcji uczcej $f\colon x \mapsto y$, gdzie y oznacza jedn ze zdefiniowanych wczeniej klas.
- Klasyfikacja probabilistyczna polega na wskazaniu klasy o najwyszym prawdopodobiestwie:
$$\hat{y} = \arg\max_{y} P(y \mid x)$$
- Klasyfikatory probabilistyczne klasyfikatory bayesowskie naiwny klasyfikator bayesowski

1.1.1 Twierdzenie Bayesa

$$P(y_k \mid x) = \frac{P(x \mid y_k) \cdot P(y_k)}{\sum_i P(x \mid y_i)\, P(y_i)}$$

1.1.2 Twierdzenie Bayesa

$$\underbrace{P(y_k \mid x)}_{\text{prawd. a posteriori}} = \frac{\overbrace{P(x \mid y_k)}^{\text{model klasy}} \cdot \overbrace{P(y_k)}^{\text{prawd. a priori}}}{\underbrace{\sum_i P(x \mid y_i)\, P(y_i)}_{\text{wyraenie normalizacyjne}}}$$

1.1.3 Rola wyraenia normalizacyjnego w twierdzeniu Bayesa

Przykad: obserwacja nietypowa ma mae prawdopodobiestwo wzgldem dowolnej klasy, wyraenie normalizacyjne sprawia, e to prawdopodobiestwo staje si porównywalne z prawdopodobiestwami typowych obserwacji, ale nie wpywa na klasyfikacj!

1.1.4 Klasyfikatory dyskryminatywne a generatywne

- Klasyfikatory generatywne tworz model rozkadu prawdopodobiestwa dla kadej z klas.
- Klasyfikatory dyskryminatywne wyznaczaj granic klas (*decision boundary*) bezporednio.
- Naiwny klasyfikator baywsowski jest klasyfikatorem generatywnym (poniewa wyznacza $P(x \mid y)$).
- Wszystkie klasyfikatory generatywne s probabilistyczne, ale nie na odwrót.
- Regresja logistyczna jest przykadem klasyfikatora dyskryminatywnego.

1.1.5 Zaoenie niezalenoci dla naiwnego klasyfikatora bayesowskiego

- Naiwny klasyfikator bayesowski jest *naiwny*, poniewa zakada, e poszczególne cechy s niezalene od siebie:

$$P(x_1, \ldots, x_n \mid y) = \prod_{i=1}^{n} P(x_i \mid x_1, \ldots, x_{i-1}, y) = \prod_{i=1}^{n} P(x \mid y)$$

- To zaoenie jest bardzo przydatne ze wzgldów obliczeniowych, poniewa bardzo czsto mamy do czynienia z ogromn liczb cech (bitmapy, sowniki itp.)

1.1.6 Naiwny klasyfikator bayesowski – przykad

In [10]:
```
# Przydtne importy

import ipywidgets as widgets
import matplotlib.pyplot as plt
import numpy as np
import pandas

%matplotlib inline
```

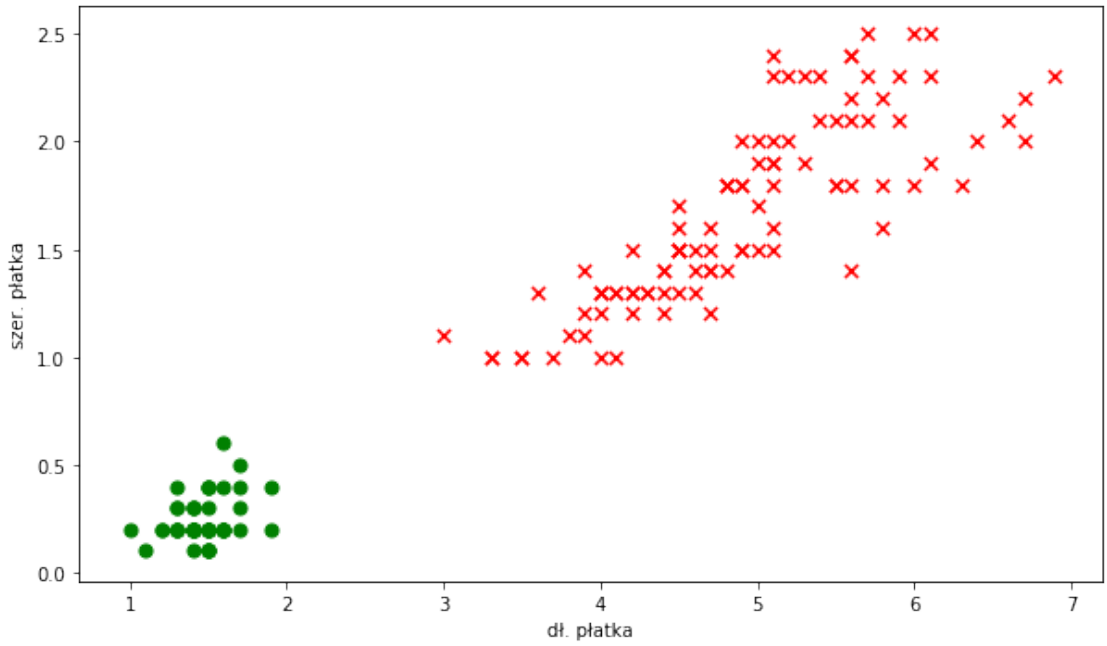
In [11]:
```
# Wczytanie danych (gatunki kosaców)

data_iris_setosa = (
    pandas.read_csv('iris.csv', usecols=['p.d.', 'p.sz.', 'Gatunek'])
    .apply(lambda x: [x[0], x[1], 1 if x[2] == 'Iris-setosa' else 0], axis=1))
data_iris_setosa.columns = ['d. patka', 'szer. patka', 'Iris setosa?']

m, n_plus_1 = data_iris_setosa.values.shape
n = n_plus_1 - 1
Xn = data_iris_setosa.values[:, 0:n].reshape(m, n)

X = np.matrix(np.concatenate((np.ones((m, 1)), Xn), axis=1)).reshape(m, n_plus_1)
Y = np.matrix(data_iris_setosa.values[:, 2]).reshape(m, 1)
```

In [12]:
```
classes = [0, 1]
count = [sum(1 if y == c else 0 for y in Y.T.tolist()[0]) for c in classes]
prior_prob = [float(count[c]) / float(Y.shape[0]) for c in classes]

print 'count: ', {c: count[c] for c in classes}
print 'prior prob.:', {c: prior_prob[c] for c in classes}
```

```
count:  {0: 100, 1: 50}
prior prob.: {0: 0.6666666666666666, 1: 0.3333333333333333}
```

In [13]:
```
# Wykres danych (wersja macierzowa)
def plot_data_for_classification(X, Y, xlabel, ylabel):
    fig = plt.figure(figsize=(16*.6, 9*.6))
    ax = fig.add_subplot(111)
    fig.subplots_adjust(left=0.1, right=0.9, bottom=0.1, top=0.9)
    X = X.tolist()
    Y = Y.tolist()
    X1n = [x[1] for x, y in zip(X, Y) if y[0] == 0]
    X1p = [x[1] for x, y in zip(X, Y) if y[0] == 1]
    X2n = [x[2] for x, y in zip(X, Y) if y[0] == 0]
    X2p = [x[2] for x, y in zip(X, Y) if y[0] == 1]
    ax.scatter(X1n, X2n, c='r', marker='x', s=50, label='Dane')
    ax.scatter(X1p, X2p, c='g', marker='o', s=50, label='Dane')

    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.margins(.05, .05)
    return fig
```

In [14]:
```
fig = plot_data_for_classification(X, Y, xlabel=u'd. patka', ylabel=u'szer. patka')
```



In [15]:
```
XY = np.column_stack((X, Y))
XY_split = [XY[np.where(XY[:,3] == c)[0]] for c in classes]
```

```
        X_split = [XY_split[c][:,0:3] for c in classes]
        Y_split = [XY_split[c][:,3] for c in classes]

        X_mean = [np.mean(X_split[c], axis=0) for c in classes]
        X_std = [np.std(X_split[c], axis=0) for c in classes]
        print X_mean
        print X_std
```

```
[matrix([[ 1.   ,  4.906,  1.676]]), matrix([[ 1.   ,  1.464,  0.244]])]
[matrix([[ 0.        ,  0.8214402 ,  0.42263933]]), matrix([[ 0.        ,  0.17176728,  0.1061
```

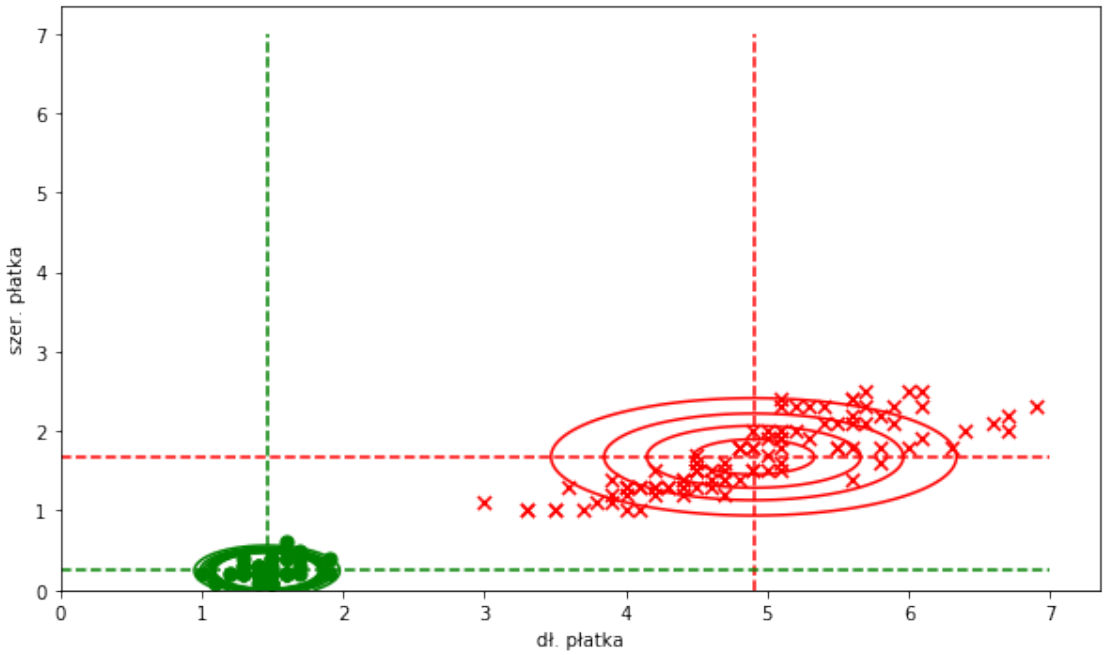
In [16]:
```
# Rysowanie rednich
def draw_means(fig, means, xmin=0.0, xmax=7.0, ymin=0.0, ymax=7.0):
    class_color = {0: 'r', 1: 'g'}
    classes = range(len(means))
    ax = fig.axes[0]
    mean_x1 = [means[c].item(0, 1) for c in classes]
    mean_x2 = [means[c].item(0, 2) for c in classes]
    for c in classes:
        ax.plot([mean_x1[c], mean_x1[c]], [xmin, xmax],
                color=class_color.get(c, 'c'), linestyle='dashed')
        ax.plot([ymin, ymax], [mean_x2[c], mean_x2[c]],
                color=class_color.get(c, 'c'), linestyle='dashed')
```

In [17]:
```
from scipy.stats import norm

# Prawdopodobiestwo klasy dla pojedynczej cechy
def prob(x, c, feature, mean, std):
    return norm(mean[c].item(0, feature), std[c].item(0, feature)).pdf(x)

# Prawdopodobiestwo klasy
def class_prob(x, c, mean, std):
    return prob(x[1], c, 1, mean, std) * prob(x[2], c, 2, mean, std)
```

In [18]:
```
# Wykres prawdopodobiestw klas
def plot_prob(fig, X_mean, X_std, classes, xmin=0.0, xmax=7.0, ymin=0.0, ymax=7.0):
    class_color = {0: 'r', 1: 'g'}
    ax = fig.axes[0]
    x1, x2 = np.meshgrid(np.arange(xmin, xmax, 0.02),
                         np.arange(xmin, xmax, 0.02))
    for c in classes:
        fun1 = lambda x: prob(x, c, 1, X_mean, X_std)
        fun2 = lambda x: prob(x, c, 2, X_mean, X_std)
        p = fun1(x1) * fun2(x2)
        plt.contour(x1, x2, p, levels=np.arange(0.0, 1.0, 0.1),
                    colors=class_color.get(c, 'c'), lw=3)
```

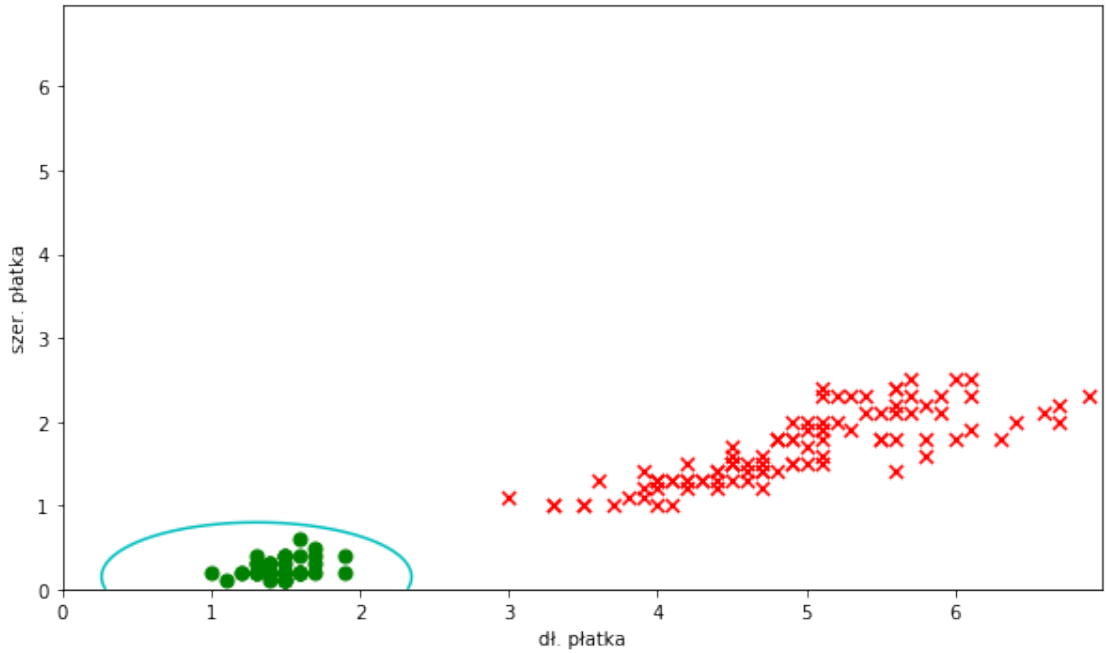
In [19]:
```
fig = plot_data_for_classification(X, Y, xlabel=u'd. patka', ylabel=u'szer. patka')
draw_means(fig, X_mean)
plot_prob(fig, X_mean, X_std, classes)
```

```
/home/pawel/.local/lib/python2.7/site-packages/matplotlib/contour.py:967: UserWarning: The foll
  s)
```



In [20]:
```
# Prawdopodobiestwo a posteriori
def posterior_prob(x, c):
    normalizer = sum(class_prob(x, c, X_mean, X_std)
                     * prior_prob[c]
                    for c in classes)
    return (class_prob(x, c, X_mean, X_std)
            * prior_prob[c]
            / normalizer)

# Wykres granicy klas dla naiwnego Bayesa
def plot_decision_boundary_bayes(fig, X_mean, X_std, xmin=0.0, xmax=7.0, ymin=0.0, yma
    ax = fig.axes[0]
    x1, x2 = np.meshgrid(np.arange(xmin, xmax, 0.02),
                         np.arange(ymin, ymax, 0.02))
    p = [posterior_prob([1, x1, x2], c) for c in classes]
    p_diff = p[1] - p[0]
    plt.contour(x1, x2, p_diff, levels=[0.0], colors='c', lw=3);
```

```
In [21]: fig = plot_data_for_classification(X, Y, xlabel=u'd. patka', ylabel=u'szer. patka')
         plot_decision_boundary_bayes(fig, X_mean, X_std)
```



1.1.7 Dla porównania: regresja logistyczna na tych samych danych

```
In [39]: def powerme(x1,x2,n):
             X = []
             for m in range(n+1):
                 for i in range(m+1):
                     X.append(np.multiply(np.power(x1,i),np.power(x2,(m-i))))
             return np.hstack(X)

         # Funkcja logistyczna
         def safeSigmoid(x, eps=0):
             y = 1.0/(1.0 + np.exp(-x))
             if eps > 0:
                 y[y < eps] = eps
                 y[y > 1 - eps] = 1 - eps
             return y

         # Funkcja hipotezy dla regresji logistycznej
         def h(theta, X, eps=0.0):
             return safeSigmoid(X*theta, eps)

         # Funkcja kosztu dla regresji logistycznej
```

```
        def J(h,theta,X,y, lamb=0):
            m = len(y)
            f = h(theta, X, eps=10**-7)
            j = -np.sum(np.multiply(y, np.log(f)) +
                        np.multiply(1 - y, np.log(1 - f)), axis=0)/m
            if lamb > 0:
                j += lamb/(2*m) * np.sum(np.power(theta[1:],2))
            return j

        # Gradient funkcji kosztu
        def dJ(h,theta,X,y,lamb=0):
            g = 1.0/y.shape[0]*(X.T*(h(theta,X)-y))
            if lamb > 0:
                g[1:] += lamb/float(y.shape[0]) * theta[1:]
            return g

        # Funkcja klasyfikujca
        def classifyBi(theta, X):
            prob = h(theta, X)
            return prob
```

In [40]:
```
        # Przygotowanie danych dla wielomianowej regresji logistycznej

        data = np.matrix(data_iris_setosa)

        Xpl = powerme(data[:, 1], data[:, 0], n)
        Ypl = np.matrix(data[:, 2]).reshape(m, 1)
```

In [41]:
```
        # Metoda gradientu prostego dla regresji logistycznej
        def GD(h, fJ, fdJ, theta, X, y, alpha=0.01, eps=10**-3, maxSteps=10000):
            errorCurr = fJ(h, theta, X, y)
            errors = [[errorCurr, theta]]
            while True:
                # oblicz nowe theta
                theta = theta - alpha * fdJ(h, theta, X, y)
                # raportuj poziom bdu
                errorCurr, errorPrev = fJ(h, theta, X, y), errorCurr
                # kryteria stopu
                if abs(errorPrev - errorCurr) <= eps:
                    break
                if len(errors) > maxSteps:
                    break
                errors.append([errorCurr, theta])
            return theta, errors
```

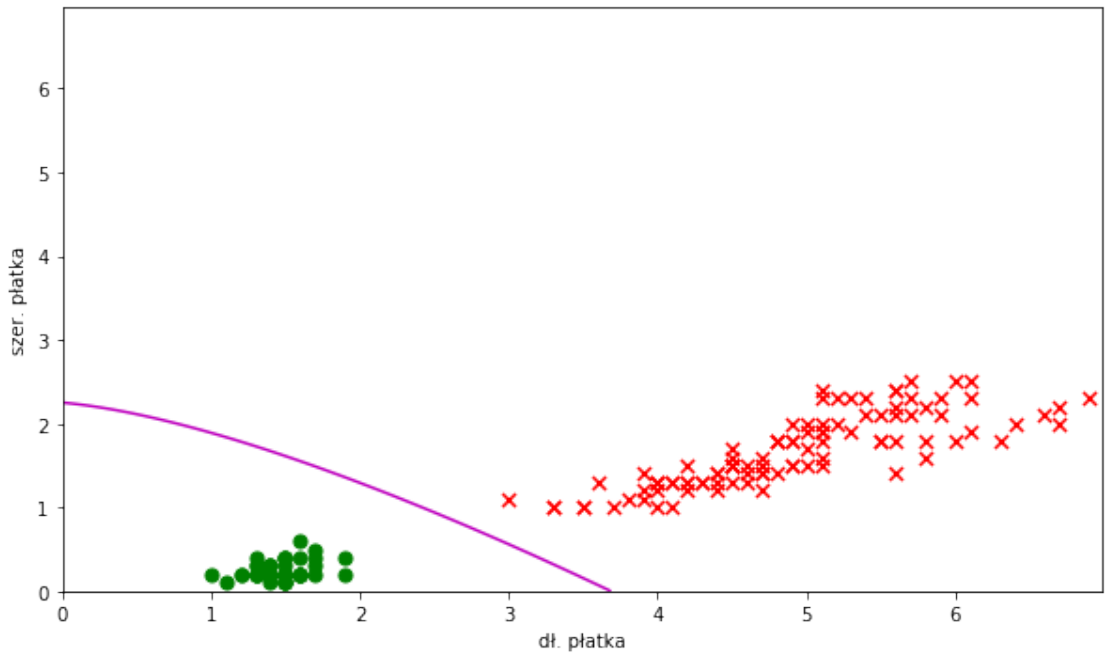
In [42]:
```
        # Uruchomienie metody gradientu prostego dla regresji logistycznej
        theta_start = np.matrix(np.zeros(Xpl.shape[1])).reshape(Xpl.shape[1], 1)
        theta, errors = GD(h, J, dJ, theta_start, Xpl, Ypl,
```

```
                                alpha=0.1, eps=10**-7, maxSteps=100000)
print(r'theta = {}'.format(theta))
```

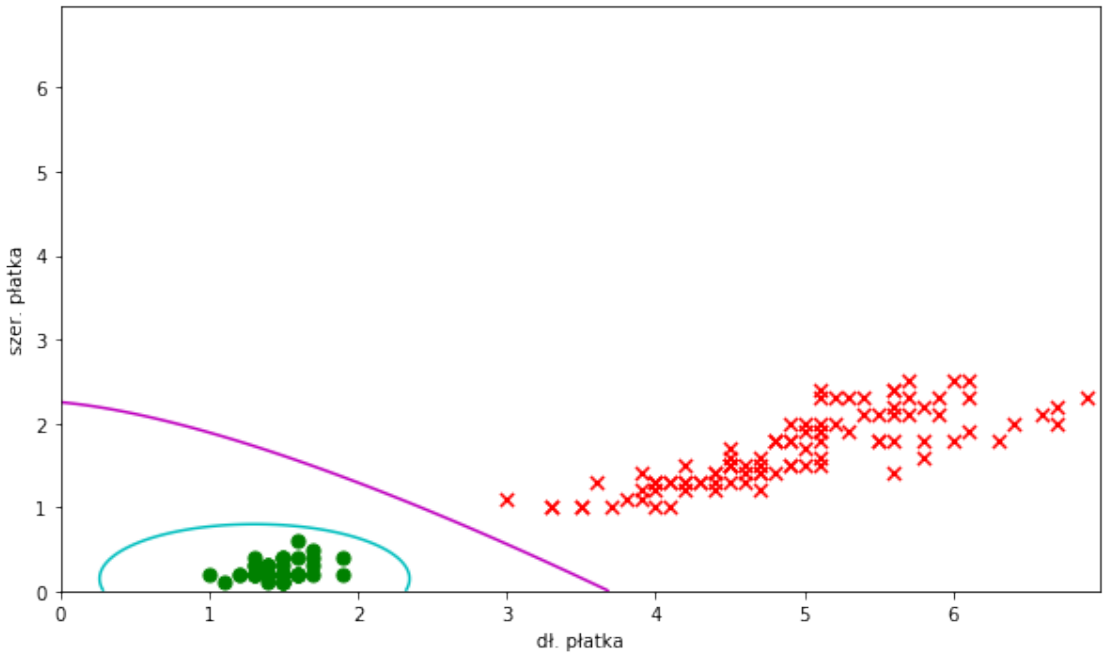
```
theta = [[ 4.01960795]
 [ 3.89499137]
 [ 0.18747599]
 [-1.3524039 ]
 [-2.00123783]
 [-0.87625505]]
```

In [43]:
```
# Wykres granicy klas
def plot_decision_boundary(fig, theta, Xpl, xmin=0.0, xmax=7.0):
    ax = fig.axes[0]
    xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.02),
                         np.arange(xmin, xmax, 0.02))
    l = len(xx.ravel())
    C = powerme(yy.reshape(l, 1), xx.reshape(l, 1), n)
    z = classifyBi(theta, C).reshape(int(np.sqrt(l)), int(np.sqrt(l)))

    plt.contour(xx, yy, z, levels=[0.5], colors='m', lw=3);
```

In [44]:
```
fig = plot_data_for_classification(Xpl, Ypl, xlabel=u'd. patka', ylabel=u'szer. patka
plot_decision_boundary(fig, theta, Xpl)
```



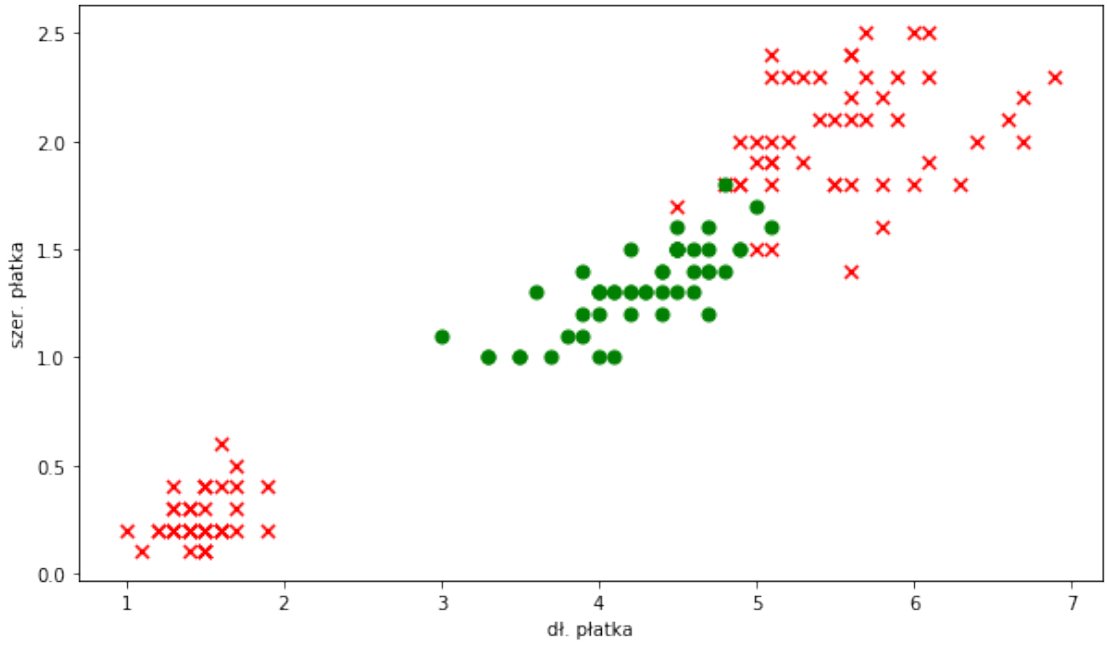
In [45]:
```
fig = plot_data_for_classification(Xpl, Ypl, xlabel=u'd. patka', ylabel=u'szer. patka
plot_decision_boundary(fig, theta, Xpl)
plot_decision_boundary_bayes(fig, X_mean, X_std)
```



1.1.8 Inny przykad

```
In [46]: # Wczytanie danych (gatunki kosaców)

         data_iris_versicolor = (
             pandas.read_csv('iris.csv', usecols=['p.d.', 'p.sz.', 'Gatunek'])
             .apply(lambda x: [x[0], x[1], 1 if x[2] == 'Iris-versicolor' else 0], axis=1))
         data_iris_versicolor.columns = ['d. patka', 'szer. patka', 'Iris versicolor?']

         m, n_plus_1 = data_iris_versicolor.values.shape
         n = n_plus_1 - 1
         Xn = data_iris_versicolor.values[:, 0:n].reshape(m, n)

         X = np.matrix(np.concatenate((np.ones((m, 1)), Xn), axis=1)).reshape(m, n_plus_1)
         Y = np.matrix(data_iris_versicolor.values[:, 2]).reshape(m, 1)

In [47]: classes = [0, 1]
         count = [sum(1 if y == c else 0 for y in Y.T.tolist()[0]) for c in classes]
         prior_prob = [float(count[c]) / float(Y.shape[0]) for c in classes]

         print 'count: ', {c: count[c] for c in classes}
         print 'prior prob.:', {c: prior_prob[c] for c in classes}

count:  {0: 100, 1: 50}
prior prob.: {0: 0.6666666666666666, 1: 0.3333333333333333}
```

In [48]:
```
fig = plot_data_for_classification(X, Y, xlabel=u'd. patka', ylabel=u'szer. patka')
```

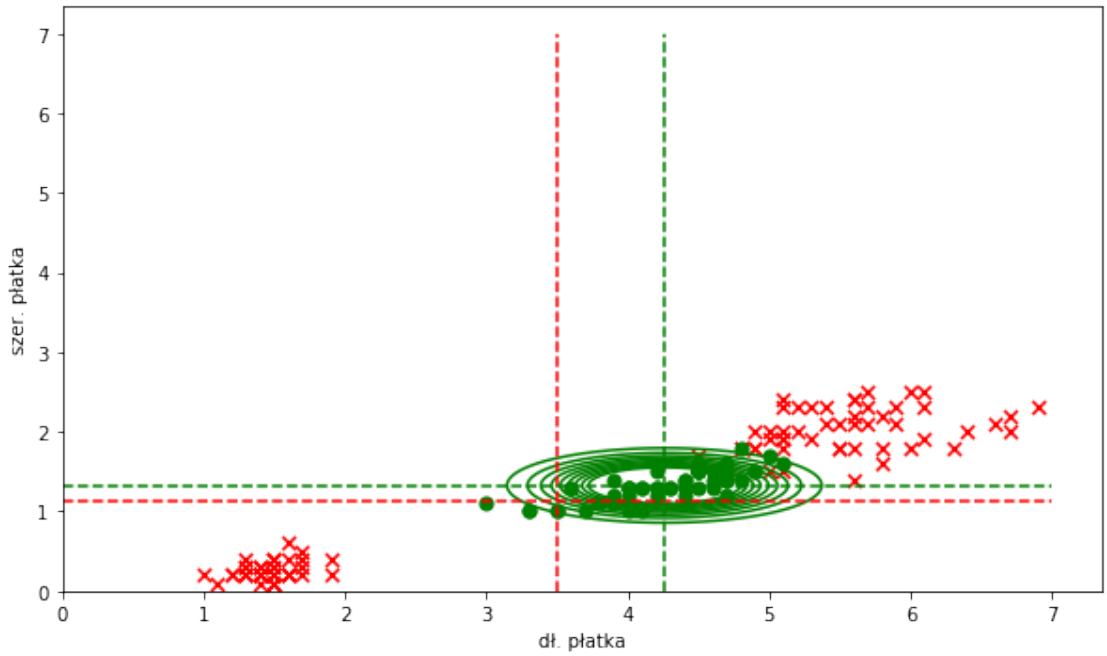


In [49]:
```
XY = np.column_stack((X, Y))
XY_split = [XY[np.where(XY[:,3] == c)[0]] for c in classes]
X_split = [XY_split[c][:,0:3] for c in classes]
Y_split = [XY_split[c][:,3] for c in classes]

X_mean = [np.mean(X_split[c], axis=0) for c in classes]
X_std = [np.std(X_split[c], axis=0) for c in classes]
print X_mean
print X_std
```

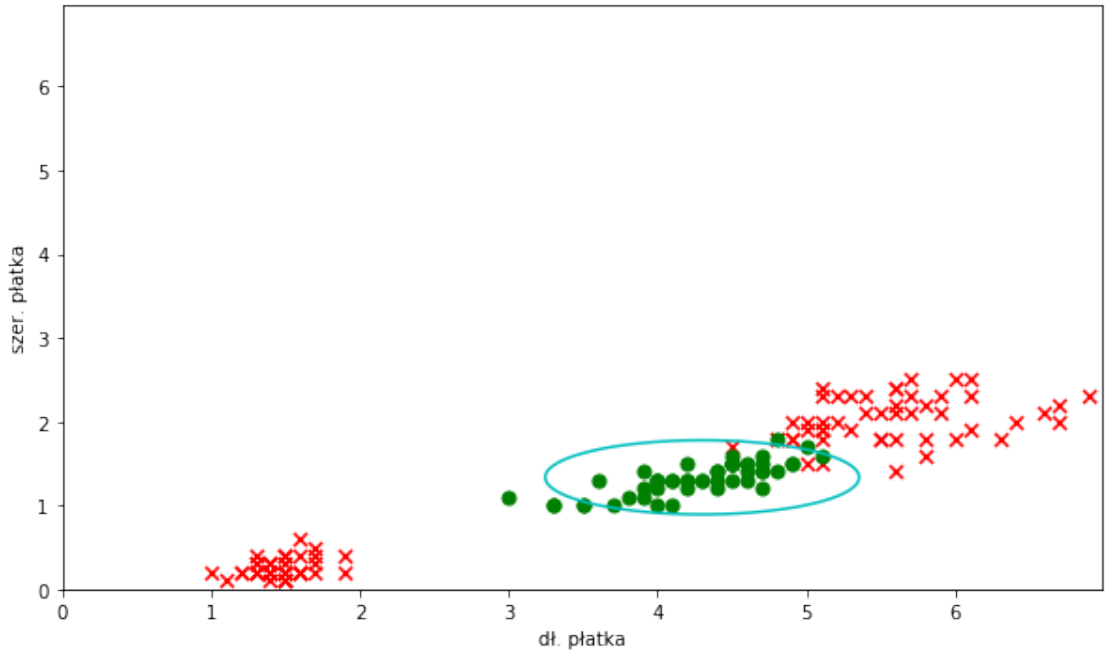
```
[matrix([[ 1.   ,  3.508,  1.135]]), matrix([[ 1.   ,  4.26 ,  1.326]])]
[matrix([[ 0.        ,  2.08373127,  0.91459007]]), matrix([[ 0.        ,  0.46518813,  0.1957
```

In [50]:
```
fig = plot_data_for_classification(X, Y, xlabel=u'd. patka', ylabel=u'szer. patka')
draw_means(fig, X_mean)
plot_prob(fig, X_mean, X_std, classes)
```

```
/home/pawel/.local/lib/python2.7/site-packages/matplotlib/contour.py:1180: UserWarning: No cont
  warnings.warn("No contour levels were found"
```



In [51]:
```
fig = plot_data_for_classification(X, Y, xlabel=u'd. patka', ylabel=u'szer. patka')
plot_decision_boundary_bayes(fig, X_mean, X_std)
```

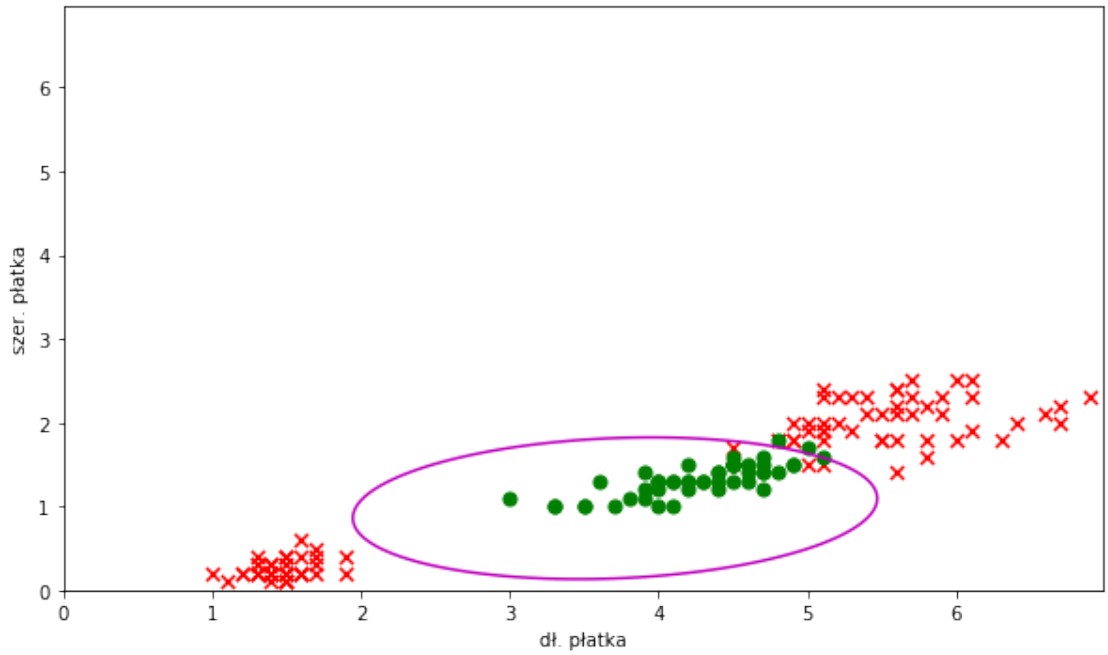


In [52]:
```
# Przygotowanie danych dla wielomianowej regresji logistycznej

data = np.matrix(data_iris_versicolor)

Xpl = powerme(data[:, 1], data[:, 0], n)
Ypl = np.matrix(data[:, 2]).reshape(m, 1)
```

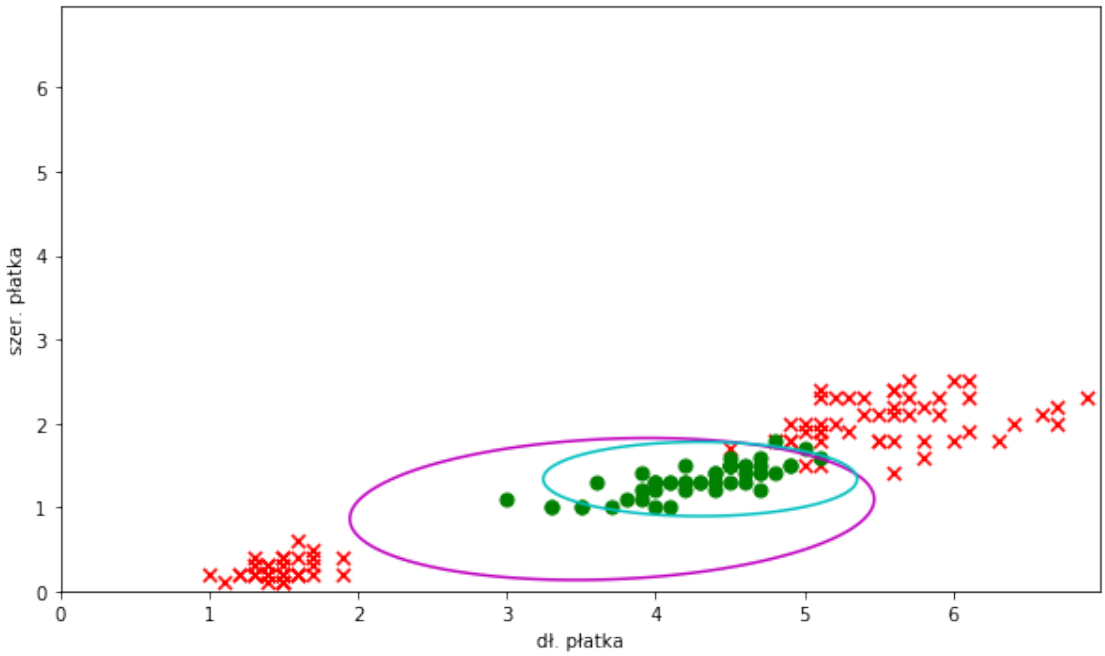
In [53]:
```
# Uruchomienie metody gradientu prostego dla regresji logistycznej
theta_start = np.matrix(np.zeros(Xpl.shape[1])).reshape(Xpl.shape[1], 1)
theta, errors = GD(h, J, dJ, theta_start, Xpl, Ypl,
                           alpha=0.05, eps=10**-7, maxSteps=100000)
print(r'theta = {}'.format(theta))
```

```
theta = [[-22.4168027 ]
 [ 12.03703048]
 [ 11.13804517]
 [ -1.75684025]
 [  1.01230657]
 [ -7.61403096]]
```

In [54]:
```
fig = plot_data_for_classification(Xpl, Ypl, xlabel=u'd. patka', ylabel=u'szer. patka
plot_decision_boundary(fig, theta, Xpl)
```



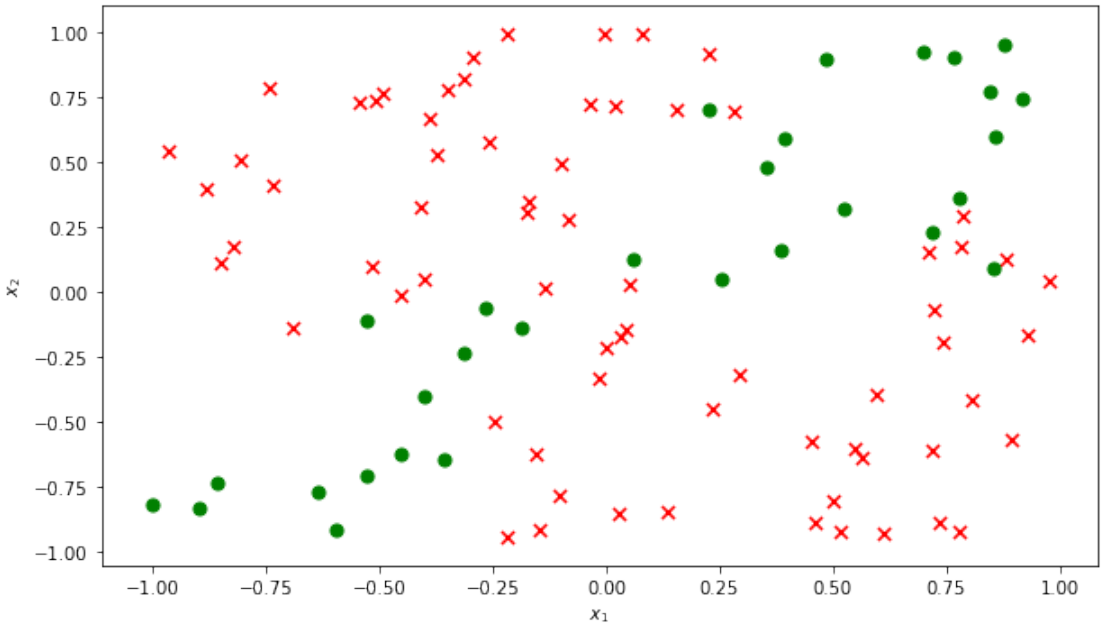
In [55]:
```
fig = plot_data_for_classification(Xpl, Ypl, xlabel=u'd. patka', ylabel=u'szer. patka
plot_decision_boundary(fig, theta, Xpl)
plot_decision_boundary_bayes(fig, X_mean, X_std)
```



1.1.9 Kiedy naiwny Bayes nie dziaa?

In [56]:
```
# Wczytanie danych
import pandas
import numpy as np

alldata = pandas.read_csv('bayes_nasty.tsv', sep='\t')
data = np.matrix(alldata)

m, n_plus_1 = data.shape
n = n_plus_1 - 1
Xn = data[:, 1:]

Xbn = np.matrix(np.concatenate((np.ones((m, 1)), Xn), axis=1)).reshape(m, n_plus_1)
Xbnp = powerme(data[:, 1], data[:, 2], n)
Ybn = np.matrix(data[:, 0]).reshape(m, 1)
```

In [57]:
```
fig = plot_data_for_classification(Xbn, Ybn, xlabel=r'$x_1$', ylabel=r'$x_2$')
```



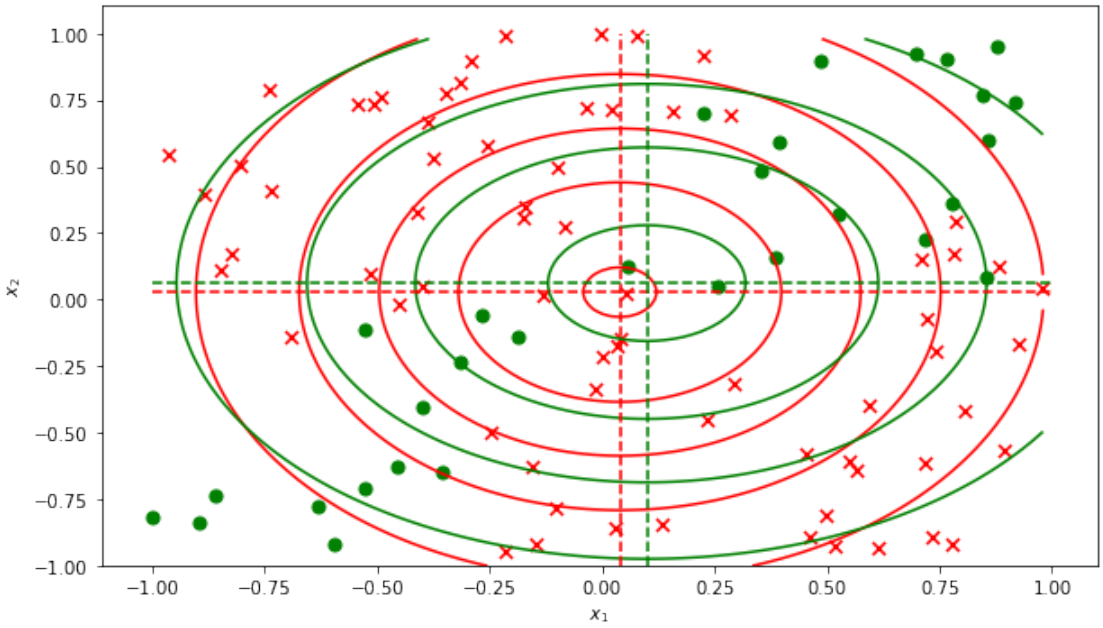

```
In [58]: classes = [0, 1]
         count = [sum(1 if y == c else 0 for y in Ybn.T.tolist()[0]) for c in classes]
         prior_prob = [float(count[c]) / float(Ybn.shape[0]) for c in classes]

         print 'count: ', {c: count[c] for c in classes}
         print 'prior prob.:', {c: prior_prob[c] for c in classes}
```

```
count:  {0: 69, 1: 30}
prior prob.: {0: 0.696969696969697, 1: 0.30303030303030304}
```

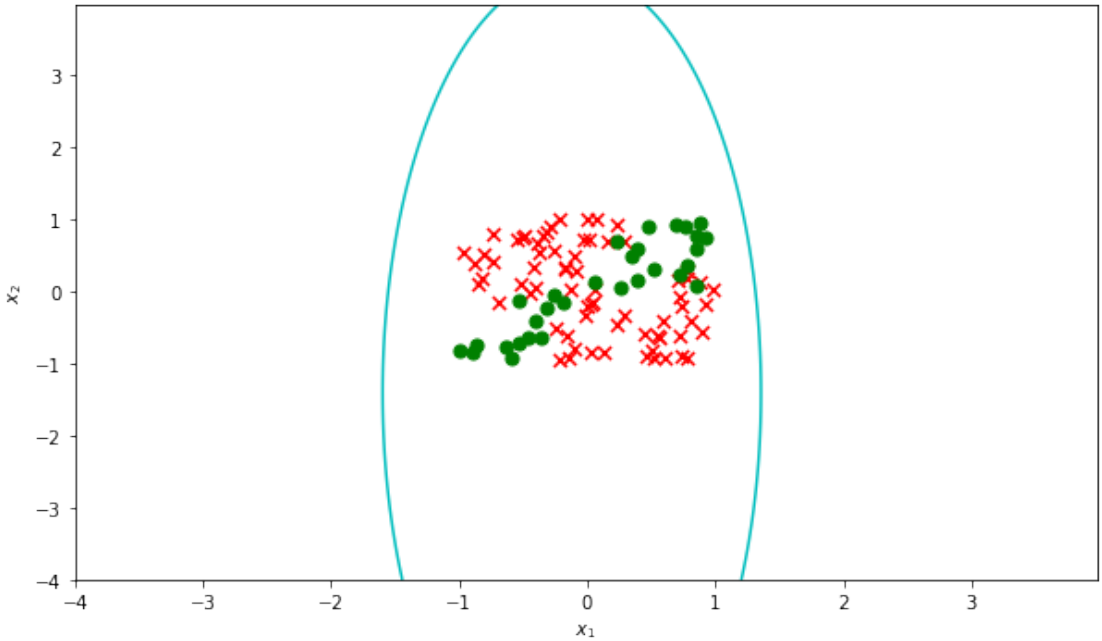
```
In [59]: XY = np.column_stack((Xbn, Ybn))
         XY_split = [XY[np.where(XY[:,3] == c)[0]] for c in classes]
         X_split = [XY_split[c][:,0:3] for c in classes]
         Y_split = [XY_split[c][:,3] for c in classes]

         X_mean = [np.mean(X_split[c], axis=0) for c in classes]
         X_std = [np.std(X_split[c], axis=0) for c in classes]
         print X_mean
         print X_std
```

```
[matrix([[ 1.        ,  0.03949835,  0.02825019]]), matrix([[ 1.        ,  0.09929617,  0.0620
[matrix([[ 0.        ,  0.52318432,  0.60106092]]), matrix([[ 0.        ,  0.61370281,  0.6081
```

```
In [60]: fig = plot_data_for_classification(Xbn, Ybn, xlabel=r'$x_1$', ylabel=r'$x_2$')
         draw_means(fig, X_mean, xmin=-1.0, xmax=1.0, ymin=-1.0, ymax=1.0)
         plot_prob(fig, X_mean, X_std, classes, xmin=-1.0, xmax=1.0, ymin=-1.0, ymax=1.0)
```



In [61]:
```
fig = plot_data_for_classification(Xbn, Ybn, xlabel=r'$x_1$', ylabel=r'$x_2$')
plot_decision_boundary_bayes(fig, X_mean, X_std, xmin=-4.0, xmax=4.0, ymin=-4.0, ymax=
```

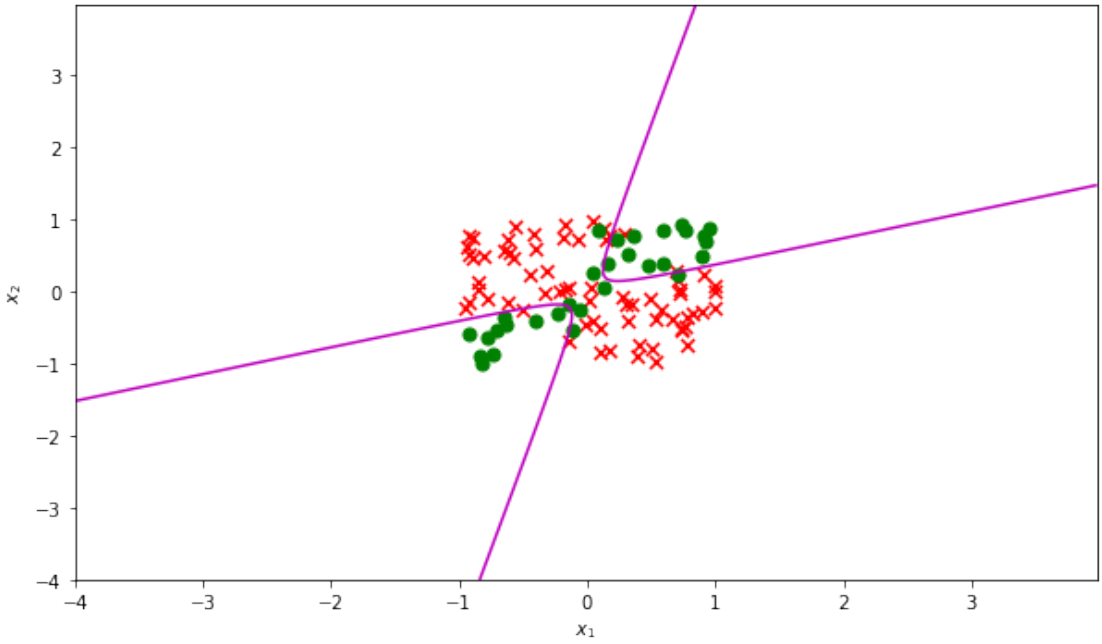


In [62]:
```
# Uruchomienie metody gradientu prostego dla regresji logistycznej
theta_start = np.matrix(np.zeros(Xbnp.shape[1])).reshape(Xbnp.shape[1], 1)
```

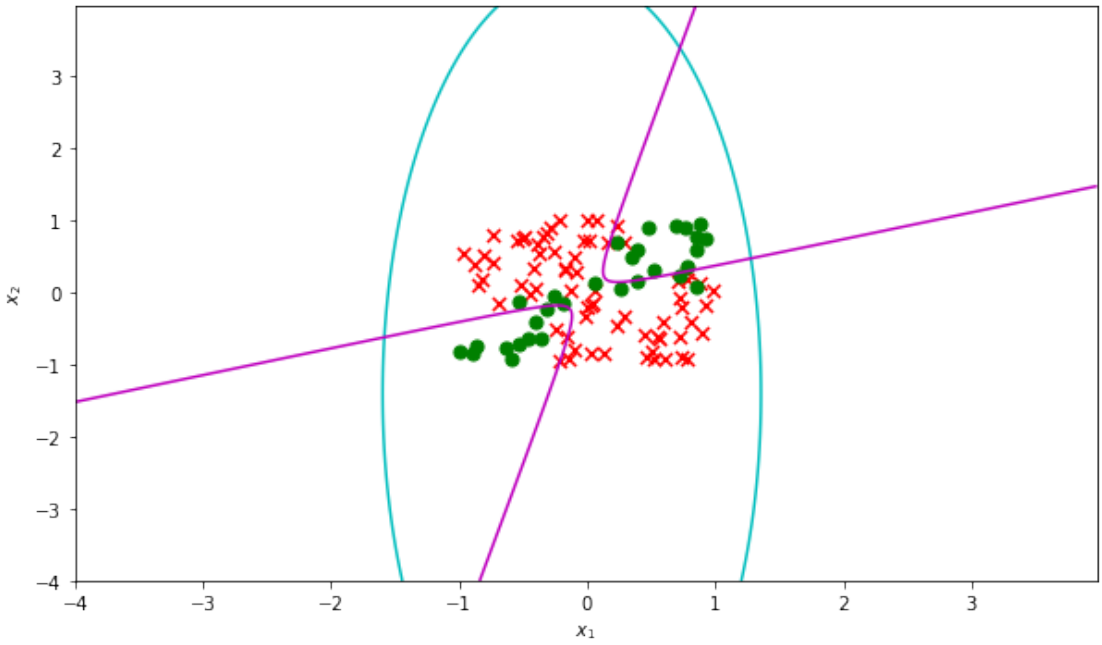
```
        theta, errors = GD(h, J, dJ, theta_start, Xbnp, Ybn,
                           alpha=0.05, eps=10**-7, maxSteps=100000)
        print(r'theta = {}'.format(theta))
```

```
theta = [[ -0.31582268]
 [  0.43496774]
 [ -0.21840373]
 [ -7.88802319]
 [ 22.73897346]
 [ -4.43682364]]
```

In [63]:
```
fig = plot_data_for_classification(Xbnp, Ybn, xlabel=r'$x_1$', ylabel=r'$x_2$')
plot_decision_boundary(fig, theta, Xbnp, xmin=-4.0, xmax=4.0)
```



In [64]:
```
fig = plot_data_for_classification(Xbn, Ybn, xlabel=r'$x_1$', ylabel=r'$x_2$')
plot_decision_boundary_bayes(fig, X_mean, X_std, xmin=-4.0, xmax=4.0, ymin=-4.0, ymax=
plot_decision_boundary(fig, theta, Xbnp, xmin=-4.0, xmax=4.0)
```



- Naiwny klasyfikator Bayesa nie dziaa, jeeli dane nie róni si redni i odchyleniem standardowym