Homework 1: Bayes optimal classifier

Students: Pratik Anand, Harsh Chaturvedi

Problem 1: Linear Classifier

Generating 10 centroids for inputs (x's), that will be used to generate a Gaussian Mixture Model.

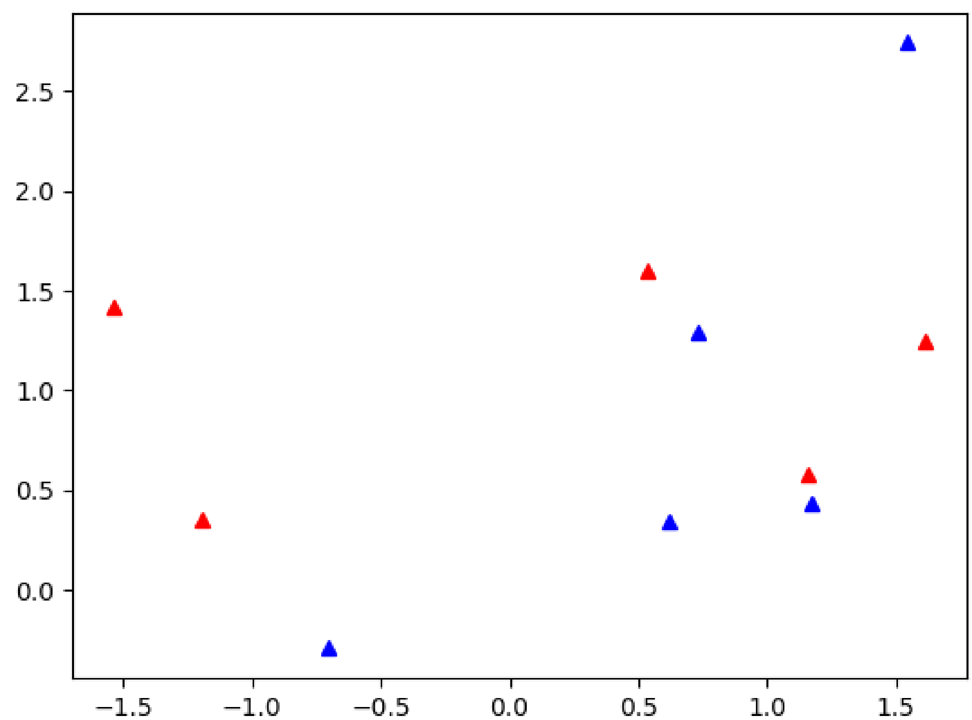
In [2]:

```
%matplotlib notebook

import matplotlib.pyplot as plt
import numpy as np, pickle
from sklearn import linear_model
from sklearn.metrics import confusion_matrix
from scipy.stats import multivariate_normal as mvn

# Generating 10 centroids for inputs (x's)
x_centroids_class1 = np.random.multivariate_normal(mean = [1, 0], cov = np.identity(2),
                                                   size = 5) #Class 1 centroids
x_centroids_class2 = np.random.multivariate_normal(mean = [0, 1], cov = np.identity(2),
                                                   size = 5) #Class 2 centroids
x_centroids = np.append(x_centroids_class1, x_centroids_class2, axis = 0) #All centroids

#plotting
ax = plt.subplots()[1]
ax.plot(x_centroids_class1[:, 0], x_centroids_class1[:, 1], marker = '^', linestyle = '', color = 'blue')
ax.plot(x_centroids_class2[:, 0], x_centroids_class2[:, 1], marker = '^', linestyle = '', color = 'red')
```



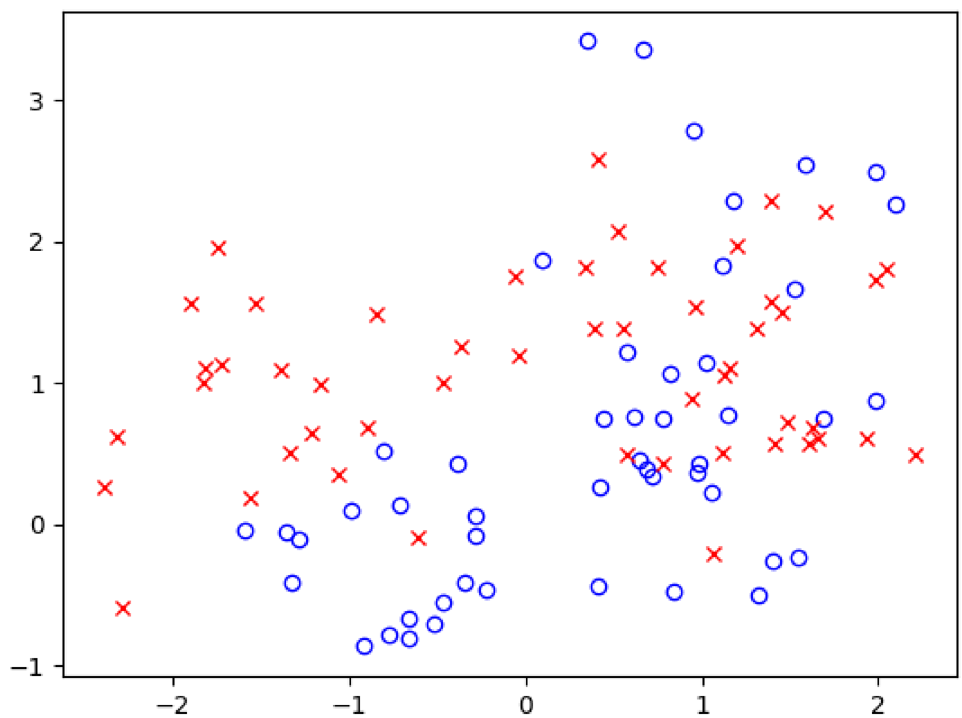
Out[2]: [<matplotlib.lines.Line2D at 0xa088cf8>]

Generating training data around centroids

In [3]:
```
# Generating data around centroids
x_train, y_train, x_train_subclass = [], [], []

for _ in range(100):
    centroid_idx = np.random.choice(10) #sample i for m_i from 1 to 10
    x_train_subclass.append(centroid_idx)
    x_train.append(np.random.multivariate_normal(mean = x_centroids[centroid_idx],
                                                 cov = np.identity(2)/5.)) #sample xj
    if centroid_idx < 5: y_train.append(0) #if i in {0, ..4}, y = 0
    else: y_train.append(1) #if i in {5, ..9}, y = 1

x_train, y_train, x_train_subclass = np.array(x_train), np.array(y_train), np.array(x_train_subclass) #list
s to arrays
#plotting
ax = plt.subplots()[1]
ax.plot(x_train[y_train == 0, 0], x_train[y_train == 0, 1], linestyle = '', marker = 'o', color = 'blue',
        markerfacecolor = 'none')
ax.plot(x_train[y_train == 1, 0], x_train[y_train == 1, 1], linestyle = '', marker = 'x', color = 'red')
```



Out[3]:
```
[<matplotlib.lines.Line2D at 0xa0ff6d8>]
```

Functions for plotting boundary and getting FP and FN rates

In [4]:
```python
def plot_boundary(ax, clf, x, y, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    params: dictionary of params to pass to contourf, optional
    """

    def make_meshgrid(x, y):
        grid_res = 500
        x_min, x_max = x.min(), x.max()
        y_min, y_max = y.min(), y.max()
        xx, yy = np.meshgrid(np.arange(x_min, x_max, (x_max - x_min)*1./grid_res)[:grid_res],
                             np.arange(y_min, y_max, (y_max - y_min)*1./grid_res)[:grid_res])
        return xx, yy

    xx, yy = make_meshgrid(x, y)
    Z = clf(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, **params)
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())

def false_rates(clf, x, y):
    tn, fp, fn, tp = confusion_matrix(y, clf(x)).ravel()
    fp_rate, fn_rate = (fp + 0.)/(fp + tn)*100., (fn + 0.)/(fn + tp)*100.
    return fp_rate, fn_rate
```

Testing a linear model

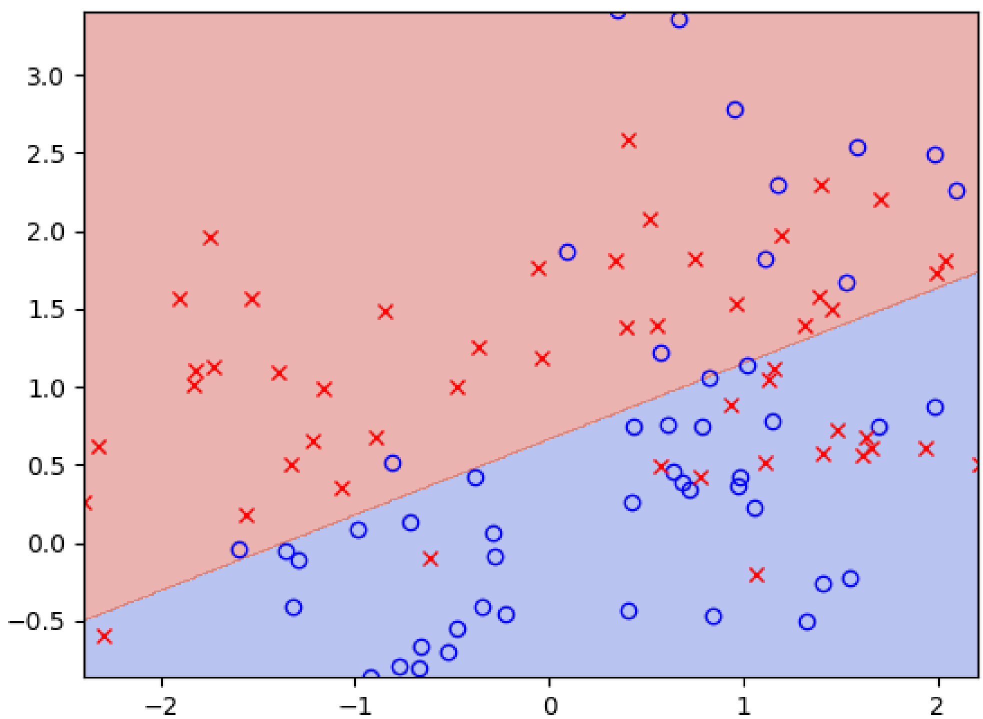
We fit a linear model to the training data and draw the decision boundary (subspace of x where $y \approx 0.5$). We highlight the false positive and false negatives in the model predictions and print the rates of both. The model performs quite poorly.

In [5]:

```
#train regression model
model = linear_model.LinearRegression()
model.fit(X = x_train, y = y_train)
lin_clf = lambda x: np.floor(model.predict(X = x) + 0.5)

#plotting boundary
ax_lin = pickle.loads(pickle.dumps(ax))
plot_boundary(ax_lin, lin_clf, x_train[:, 0],
              x_train[:, 1], cmap = plt.cm.coolwarm, alpha = 0.4)

#fp and fn rates
fp_rate, fn_rate = false_rates(lin_clf, x_train, y_train)
print 'false positive rate = ' + str(fp_rate) + '% , false negative rate = ' + str(fn_rate) + '%'
```




```
false positive rate = 28.5714285714% , false negative rate = 31.3725490196%
```

Problem 2: Bayes Optimal Classifier (BOC) derivation

Deriving optimal separating boundary, given all parameters of the data generating model

Expected prediction error EPE(G') = E (L(G,G')) where E : posterior expectation L : Loss function G : actual class G': predicted class

Since, G is discrete here :

$$L(G, G') = 0 \text{ if } G = G', 1 \text{ otherwise}$$

$$E_{X,G}(L(G, G')) = \int dGdXL(G, G')P(G, X)$$

Joint probability density : P(X, G) = P(G,X) = P (G|X)P(X) = P(X|G)P(G) Hence,

$$E = \int dX(dGL(G, G')P(G|X))P(X)$$

$(dGL(G, G')P(G|X))P(X)$ needs to be minimized since G is discrete

$$G' = ArgMIN_G(E)$$

Assuming class K* is correct :

$$G' = ArgMIN_G(P(G = 1|X) + P(G = 2|X) + \ldots . P(G = K * -1|X) + 0 + \ldots)$$

$$= ArgMIN_G(1 - P(G = K * |X))$$

$$G' = ArgMAX_{G_k}(P(G_K|X))$$

Problem 3: BOC implementation, given subclass labels, means & covs

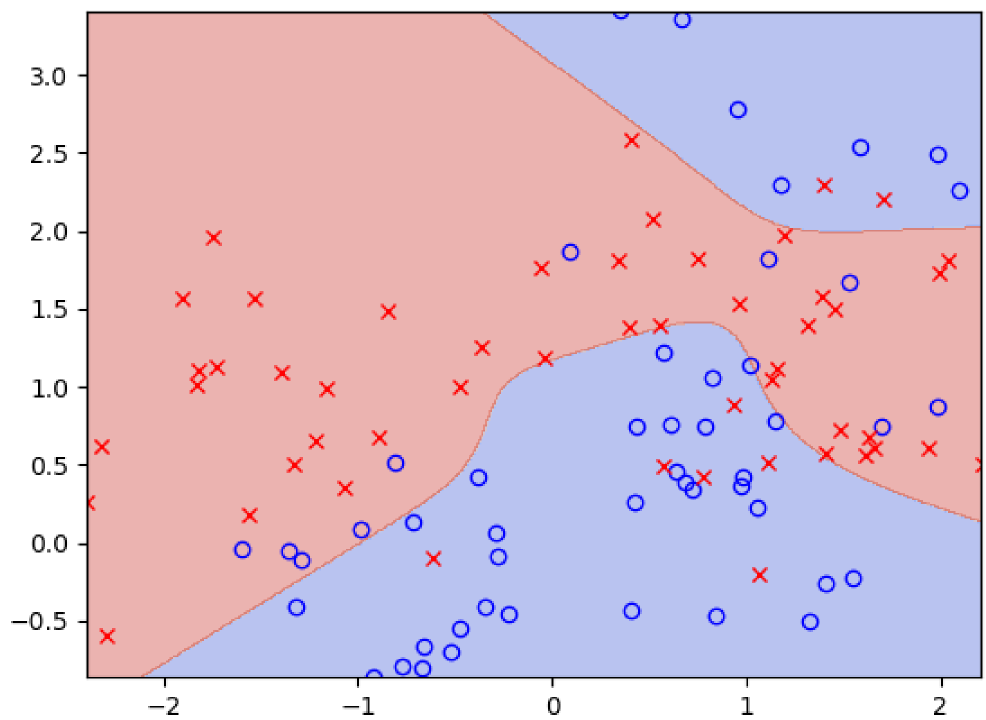
In [6]:

```
#function for bayes classifier
def bayes_clf(X, subc_means, subc_covs):
    prob_class2_numer = np.sum([mvn.pdf(x = X, mean = mean, cov = cov) for mean, cov
                          in zip(subc_means[5:], subc_covs[5:])], axis = 0) #P(class 2 | X)
    prob_class2_denom = np.sum([mvn.pdf(x = X, mean = mean, cov = cov) for mean, cov
                          in zip(subc_means, subc_covs)], axis = 0) #P(class 1 | X) + P(class 2 | X)
    prob_class2 = np.divide(prob_class2_numer, prob_class2_denom) #P_normalized(class 2 | X)
    return np.floor(prob_class2 + 0.5) #binary output

#bayes_clf for true means, covs
bayes_clf_more_info = lambda x: bayes_clf(X = x, subc_means = x_centroids, subc_covs = np.array([np.identit
y(2)/5.]*10))

#plotting boundary
ax_bay = pickle.loads(pickle.dumps(ax))
plot_boundary(ax_bay, bayes_clf_more_info, x_train[:, 0],
              x_train[:, 1], cmap = plt.cm.coolwarm, alpha = 0.4)

#fp and fn rates
fp_rate, fn_rate = false_rates(bayes_clf_more_info, x_train, y_train)
print 'false positive rate = ' + str(fp_rate) + '% , false negative rate = ' + str(fn_rate) + '%'
```




```
false positive rate = 20.4081632653% , false negative rate = 15.6862745098%
```

Problem 4: BOC implementation, given only subclass labels

We estimate the means and covariances of the subclasses numerically, then use the function we developed in Problem 4.

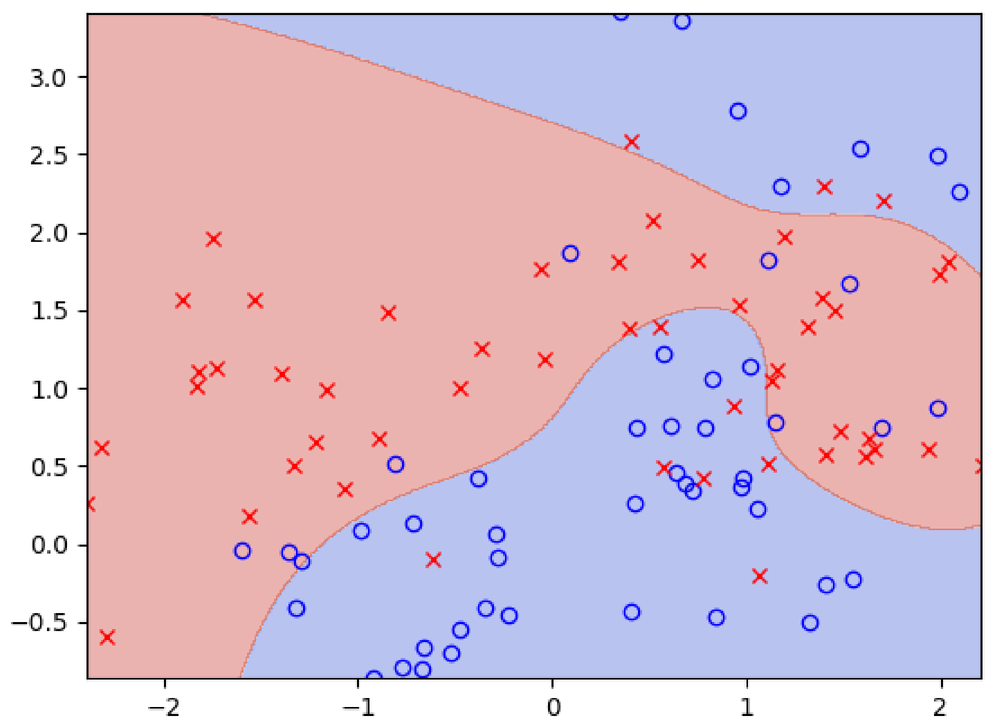
In [7]:

```
#estimated subclass means and covs, given only subclass labels
est_subc_means, est_subc_covs = [], []
for subc in range(10):
    x_subc = x_train[x_train_subclass == subc]
    est_subc_means.append(np.mean(x_subc, axis = 0))
    est_subc_covs.append(np.cov(x_subc, rowvar = False))
est_subc_means, est_subc_covs = np.array(est_subc_means), np.array(est_subc_covs)

#bayes_clf for estimated means, covs
bayes_clf_less_info = lambda x: bayes_clf(X = x, subc_means = est_subc_means, subc_covs = est_subc_covs)

#plotting boundary
ax_bay = pickle.loads(pickle.dumps(ax))
plot_boundary(ax_bay, bayes_clf_less_info, x_train[:, 0],
              x_train[:, 1], cmap = plt.cm.coolwarm, alpha = 0.4)

#fp and fn rates
fp_rate, fn_rate = false_rates(bayes_clf_less_info, x_train, y_train)
print 'false positive rate = ' + str(fp_rate) + '% , false negative rate = ' + str(fn_rate) + '%'
```




```
false positive rate = 20.4081632653% , false negative rate = 19.6078431373%
```

Problem 5 : Without the help of subclasses

Since only the super class labels are present and no subclass labels are present, we could have used a familiar algorithm like k-nearest neighbour to find k sized clusters. We can then compute the means and covariances of these subclasses and use the same in the manner of Problem 4 to build a Bayes optimal classifier.

In [ ]: