Student Names and IDs:

- Newton Kwan, nk150
- Joyce Choi, jc515
- Ashka Stephen, aas74

Homework 9

Part 1: Losses

Problem 1.1

Use the formulas in the class notes to write functions with headers

```
def regressionLoss(z, delta):
def hingeLoss(y, delta):
```

that take a label $z \in Z = \{0, 1\}$ or $y \in Y = \{-1, 1\}$ and a value δ for the signed distance from the separating hyperplane and compute the logistic-regression loss and the hinge loss *as functions of label and* δ. Keep in mind that the logistic-regression loss is a composition of logistic function and cross-entropy loss.

Also write a function with header

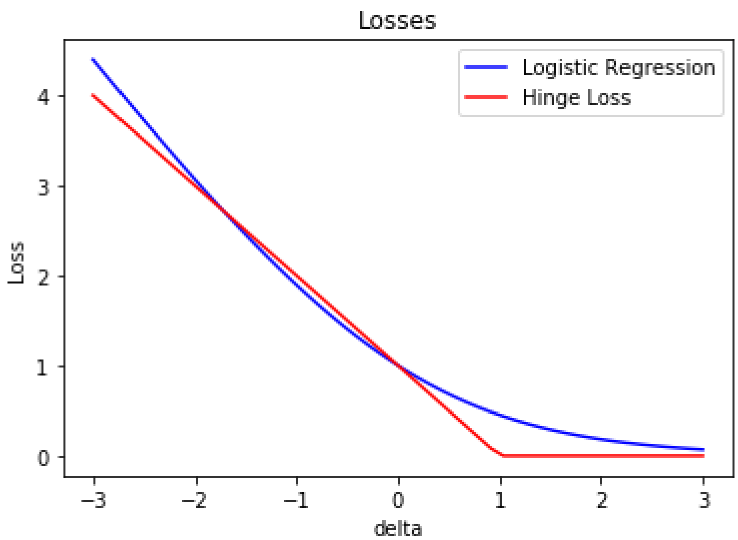
```
def plotLosses(y):
```

that takes a label $y \in Y = \{-1, 1\}$ and plots the two loss functions for $-3 \leq \delta \leq 3$. Show your code and the two plots that result from calling `plotLosses`, first with argument 1 and then with argument -1.

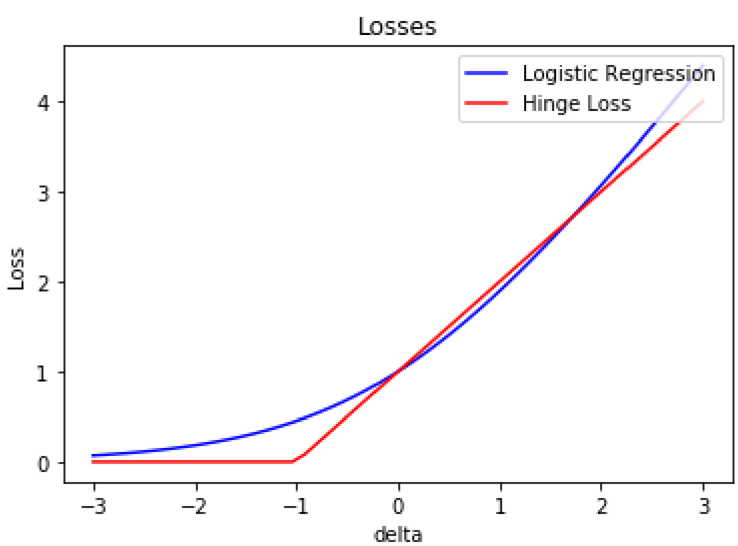
Answer

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def regressionLoss(z, delta):
    '''
    This takes label z in {0,1} and a value delta for the signed distanc
e from the separating
    hyperplane and computes the logistic-regression loss of label and de
lta
    '''

    p = 1 / (1 + np.exp(-delta))              # logistic function p
    x_loss = -z*np.log2(p) - (1-z)*np.log2(1-p) # cross entropy loss, z
 is the label {0,1}, p is the logistic function

    return x_loss

def hingeLoss(y, delta):
    '''
    this takes label y in {-1,1} and a value delta for the signed distan
ce from the separating
    hyperplane and compute the hinge loss as functions of label and delt
a.
    '''

    loss = max([0, 1 - y*delta])

    return loss

def plotLosses(y):
    '''
    This function takes a label y in {-1, 1} and plots the two loss func
tions
    for -3 <= delta <= 3
    '''

    # converting y's to z's for logistic regression function which takes
 z in {0, 1}
    if y == -1: # this takes y = -1 and makes z = 0
        z = 0
    if y == 1:  # this takes y = 1 and makes z = 1
        z = 1

    deltas = np.linspace(-3, 3, 50) # generates a list of 50 deltas (sig
ned distnaces) between -3 and 3
    reg_losses = []                 # initialize list of logistic regres
sion losses
    hinge_losses = []               # initialize list of hinge losses
    for delta in deltas:
        reg_loss = regressionLoss(z, delta)
        reg_losses.append(reg_loss)
        hinge_loss = hingeLoss(y, delta)
        hinge_losses.append(hinge_loss)

    plt.plot(deltas, reg_losses, label = "Logistic Regression", color =
```

```
"b")
    plt.plot(deltas, hinge_losses, label = "Hinge Loss", color = "r")
    plt.title("Losses")
    plt.xlabel("delta")
    plt.ylabel("Loss")
    plt.legend(loc = "upper right")

    return

# y = 1
test_y = 1
plotLosses(test_y)
```



In [2]:
```
# y = -1
test_y = -1
plotLosses(test_y)
```



Problem 1.2

Suppose that there is a single data outlier that is misclassified by a very large (negative) margin. Referring to the plots in Problem 1.1, which of the two losses is more sensitive to that outlier, and why?

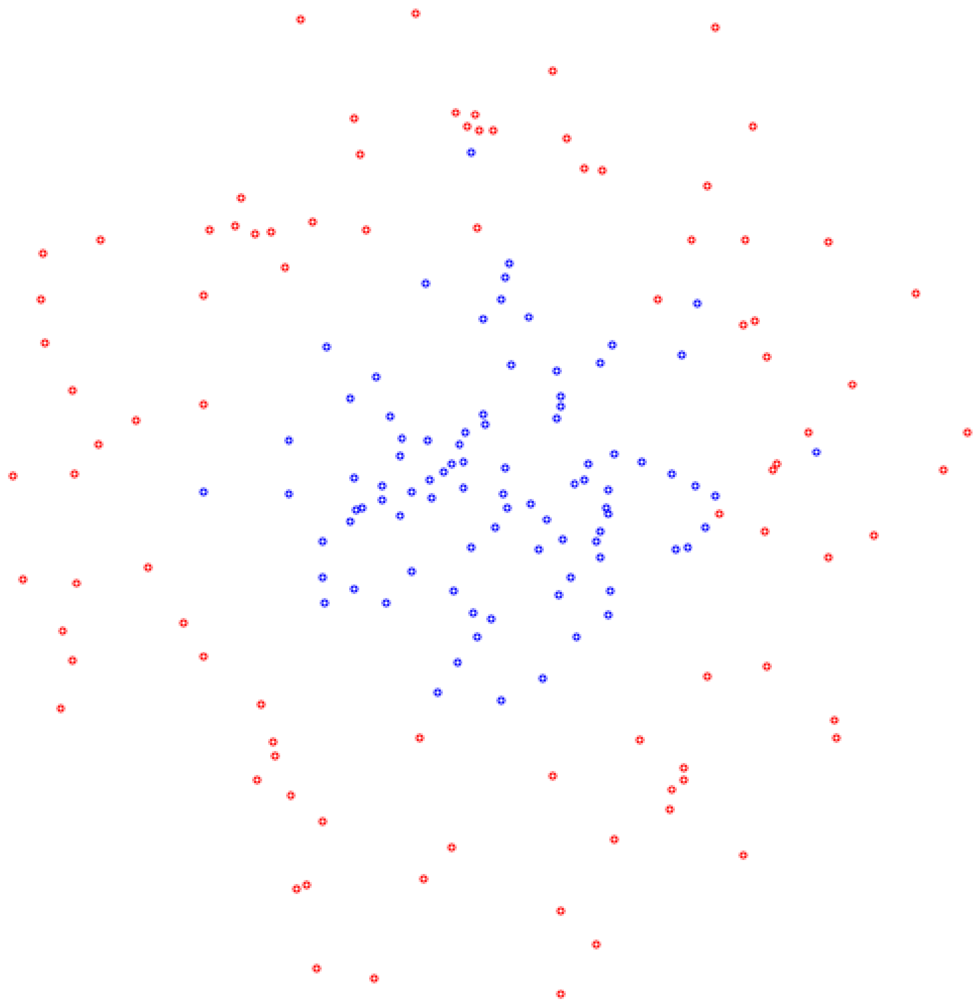
Answer

The logistic regression loss is more sensitive to a very large (negative) margin because the loss is exponential decreasing or increasing and the hinge loss is either 0 or linear. On both graphs, the logistic regression loss is greater than the hinge loss.

Part 2: SVMs

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as ds
from sklearn.model_selection import train_test_split
%matplotlib inline
ns = 300
data = {}
data['x'], data['y'] = ds.make_circles(n_samples=ns,
    noise=0.2, factor=0.3, random_state=1)
testFraction = 0.6
T, S = {}, {}
T['x'], S['x'], T['y'], S['y'] = train_test_split(data['x'], data['y'],
 test_size=testFraction, random_state=0)
p, n = S['x'][S['y']==1], S['x'][S['y']!=1]
plt.figure(figsize=(10,10))
plt.plot(p[:, 0], p[:, 1], '.b', fillstyle='none')
plt.plot(n[:, 0], n[:, 1], '.r', fillstyle='none')
plt.axis('equal')
plt.axis('off')
```

```
Out[3]: (-1.409118932486272,
         1.4707802536886163,
         -1.509294151919646,
         1.4493604864382867)
```



Problem 2.1

The data in T and S is obviously not linearly separable, so a linear classifier should not be expected to do well. To verify this, train `sklearn.svm.SVC` with arguments `kernel='linear', C=1` on T, show its zero-one training and test error rates (on S) as percentages with two decimal digits after the period, and plot the data in T and decision regions. Warning: Most points wil be support vectors for this first plot. This is OK.

Answer

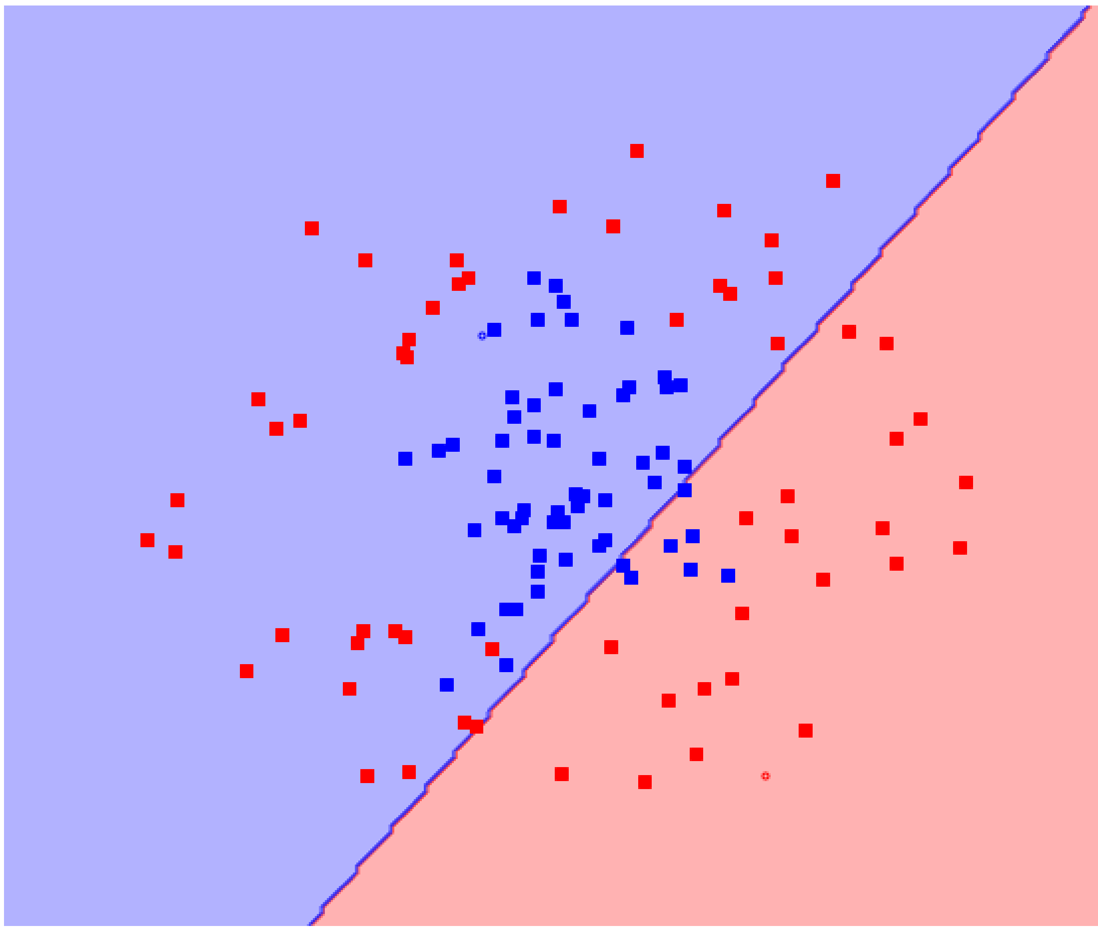
In [4]:
```
from sklearn.svm import SVC
from matplotlib import colors
import math

# train sklearn.svm.SVC with arguments kernel = 'linear', C = 1 on T
# show its zero-one training and test error rates (on S) as percentages
 with two decimal places

clf = SVC(C = 1, kernel = 'linear') # training a linear Support Vector C
lassification
clf.fit(T['x'], T['y'])            # fit the classifier to the training
 data
zero_one_training_rate = 1 - clf.score(T['x'], T['y'])
test_error_rate = 1 - clf.score(S['x'], S['y'])           # test the cla
ssifier on the test set and get a score

print("Zero-one training error rate: {:.2f}".format(zero_one_training_ra
te*100), "%")
print("Test error rate: {:.2f}".format(test_error_rate*100), "%")

# plotting the data in T and the decision regions


# find the maximum and minimum x and y value (4 numbers)

max_y = -math.inf
max_x = -math.inf
min_y = math.inf
min_x = math.inf

for i in range(len(p)):
    # min and max x
    if T['x'][i][0] < min_x: # check if x value in T is less than our cu
rrent lowest x value
        min_x = T['x'][i][0]
    if T['x'][i][0] > max_x: # check if x value in T is greater than our
 current highest x value
        max_x = T['x'][i][0]
    if T['x'][i][1] < min_y: # check if y value in T is less than our cu
rrent lowest y value
        min_y = T['x'][i][1]
    if T['x'][i][1] > max_y: # check if y value in T is greater than our
 current highest y value
        max_y = T['x'][i][1]

max_x = max_x + 0.5 # add 0.5 to the max x value
min_x = min_x - 0.5 # subtract 0.5 to the min x value
max_y = max_y + 0.5 # subtract 0.5 to the max y value
min_y = min_y - 0.5 # subtract 0.5 from the min y value


x_range = np.linspace(min_x, max_x, (max_x - min_x) / 0.02) # create a l
ist of the x range
y_range = np.linspace(min_y, max_y, (max_y - min_y) / 0.02) # create a l
ist of the y range
X, Y = np.meshgrid(x_range, y_range)  # create the meshgrid of X, Y
```

```
#flatten
X_flat = np.concatenate(X).ravel()
Y_flat = np.concatenate(Y).ravel()

#stack x and y
XY = np.stack((X_flat, Y_flat), axis=-1)
predicted = clf.predict(XY).reshape((160, 193))
predicted_labels = clf.predict(T['x'])

a, b = T['x'][T['y']==1], T['x'][T['y']!=1]

# initialize the lists

true_pos_x = []
true_pos_y = []

true_neg_x = []
true_neg_y = []

false_pos_x = []
false_pos_y = []

false_neg_x = []
false_neg_y = []

svm_pos_x = []
svm_pos_y = []
svm_neg_x = []
svm_neg_y = []

svm_indices = clf.support_ # indices for the support vectors


size_of_T = len(T['x'])
for sample_index in range(size_of_T):
    sample = T['x'][sample_index]     # sample coordinate
    true_label = T['y'][sample_index] # true label
    predicted_label = predicted_labels[sample_index] # predicted label

    # check svm_indices
    if sample_index in svm_indices:
        # current sample is an svm if this goes through
        if true_label == 1:
            svm_pos_x.append(sample[0])
            svm_pos_y.append(sample[1])
        if true_label != 1:
            svm_neg_x.append(sample[0])
            svm_neg_y.append(sample[1])

    # true positive
    if true_label == 1 and predicted_label == 1:
        true_pos_x.append(sample[0])
        true_pos_y.append(sample[1])
    # true negative
    if true_label != 1 and predicted_label != 1:
        true_neg_x.append(sample[0])
```

```
        true_neg_y.append(sample[1])
    # false positive
    if true_label != 1 and predicted_label == 1:
        false_pos_x.append(sample[0])
        false_pos_y.append(sample[1])
    # false negatives
    if true_label == 1 and predicted_label != 1:
        false_neg_x.append(sample[0])
        false_neg_y.append(sample[1])

plt.figure(figsize=(10,10))

plt.plot(true_pos_x, true_pos_y, '.b', fillstyle ='none')
plt.plot(true_neg_x, true_neg_y, '.r', fillstyle ='none')
plt.plot(false_pos_x, false_pos_y, 'xr', fillstyle ='none')
plt.plot(false_neg_x, false_neg_y, 'xb', fillstyle ='none')
plt.plot(svm_pos_x, svm_pos_y, 'sb', fillstyle = None)
plt.plot(svm_neg_x, svm_neg_y, 'sr', fillstyle = None)

plt.axis('equal')
plt.axis('off')

h = plt.contourf(X,Y,predicted, alpha=0.3, cmap = colors.ListedColormap
(['r','b'])) #check colors
```

```
Zero-one training error rate: 37.50 %
Test error rate: 41.11 %
```

```
/Users/joycechoi/anaconda3/lib/python3.6/site-packages/ipykernel_launch
er.py:43: DeprecationWarning: object of type <class 'numpy.float64'> ca
nnot be safely interpreted as an integer.
/Users/joycechoi/anaconda3/lib/python3.6/site-packages/ipykernel_launch
er.py:44: DeprecationWarning: object of type <class 'numpy.float64'> ca
nnot be safely interpreted as an integer.
```



Problem 2.2

Do the same with `kernel='rbf'` (same value for `C` as before).

Answer

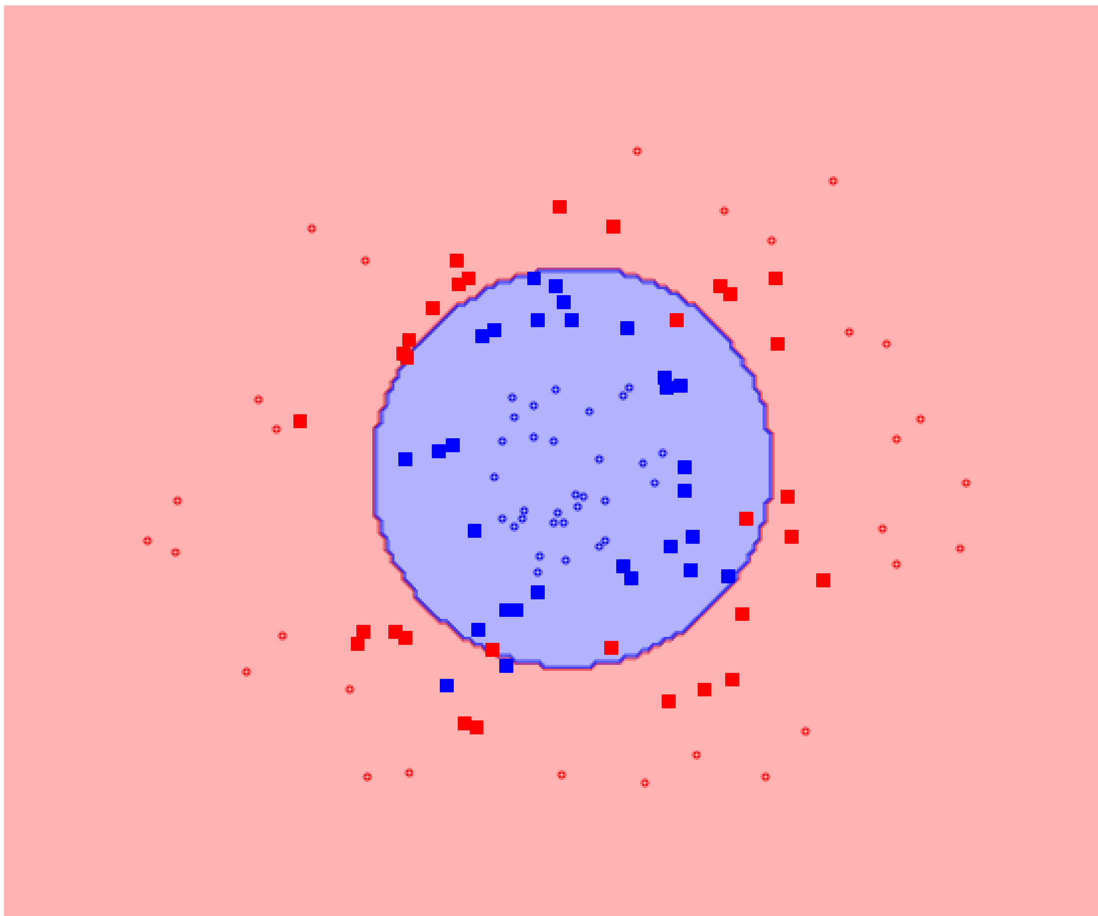
In [5]:
```
from sklearn.svm import SVC
from matplotlib import colors
import math

# train sklearn.svm.SVC with arguments kernel = 'rbf', C = 1 on T
# show its zero-one training and test error rates (on S) as percentages
 with two decimal places

clf = SVC(C = 1, kernel = 'rbf') # training a linear Support Vector Clas
sification
clf.fit(T['x'], T['y'])            # fit the classifier to the training
 data
zero_one_training_rate = 1 - clf.score(T['x'], T['y'])
test_error_rate = 1 - clf.score(S['x'], S['y'])          # test the cla
ssifier on the test set and get a score

print("Zero-one training error rate: {:.2f}".format(zero_one_training_ra
te*100), "%")
print("Test error rate: {:.2f}".format(test_error_rate*100), "%")

# plotting the data in T and the decision regions


# find the maximum and minimum x and y value (4 numbers)

max_y = -math.inf
max_x = -math.inf
min_y = math.inf
min_x = math.inf

for i in range(len(p)):
    # min and max x
    if T['x'][i][0] < min_x: # check if x value in T is less than our cu
rrent lowest x value
        min_x = T['x'][i][0]
    if T['x'][i][0] > max_x: # check if x value in T is greater than our
 current highest x value
        max_x = T['x'][i][0]
    if T['x'][i][1] < min_y: # check if y value in T is less than our cu
rrent lowest y value
        min_y = T['x'][i][1]
    if T['x'][i][1] > max_y: # check if y value in T is greater than our
 current highest y value
        max_y = T['x'][i][1]

max_x = max_x + 0.5 # add 0.5 to the max x value
min_x = min_x - 0.5 # subtract 0.5 to the min x value
max_y = max_y + 0.5 # subtract 0.5 to the max y value
min_y = min_y - 0.5 # subtract 0.5 from the min y value



x_range = np.linspace(min_x, max_x, (max_x - min_x) / 0.02) # create a l
ist of the x range
y_range = np.linspace(min_y, max_y, (max_y - min_y) / 0.02) # create a l
ist of the y range
X, Y = np.meshgrid(x_range, y_range)  # create the meshgrid of X, Y
```

```python
#flatten
X_flat = np.concatenate(X).ravel()
Y_flat = np.concatenate(Y).ravel()

#stack x and y
XY = np.stack((X_flat, Y_flat), axis=-1)
predicted = clf.predict(XY).reshape((160, 193))
predicted_labels = clf.predict(T['x'])

a, b = T['x'][T['y']==1], T['x'][T['y']!=1]

# initialize the lists

true_pos_x = []
true_pos_y = []

true_neg_x = []
true_neg_y = []

false_pos_x = []
false_pos_y = []

false_neg_x = []
false_neg_y = []

svm_pos_x = []
svm_pos_y = []
svm_neg_x = []
svm_neg_y = []

svm_indices = clf.support_ # indices for the support vectors


size_of_T = len(T['x'])
for sample_index in range(size_of_T):
    sample = T['x'][sample_index]     # sample coordinate
    true_label = T['y'][sample_index] # true label
    predicted_label = predicted_labels[sample_index] # predicted label

    # check svm_indices
    if sample_index in svm_indices:
        # current sample is an svm if this goes through
        if true_label == 1:
            svm_pos_x.append(sample[0])
            svm_pos_y.append(sample[1])
        if true_label != 1:
            svm_neg_x.append(sample[0])
            svm_neg_y.append(sample[1])

    # true positive
    if true_label == 1 and predicted_label == 1:
        true_pos_x.append(sample[0])
        true_pos_y.append(sample[1])
    # true negative
    if true_label != 1 and predicted_label != 1:
        true_neg_x.append(sample[0])
```

```
        true_neg_y.append(sample[1])
    # false positive
    if true_label != 1 and predicted_label == 1:
        false_pos_x.append(sample[0])
        false_pos_y.append(sample[1])
    # false negatives
    if true_label == 1 and predicted_label != 1:
        false_neg_x.append(sample[0])
        false_neg_y.append(sample[1])

plt.figure(figsize=(10,10))

plt.plot(true_pos_x, true_pos_y, '.b', fillstyle ='none')
plt.plot(true_neg_x, true_neg_y, '.r', fillstyle ='none')
plt.plot(false_pos_x, false_pos_y, 'xr', fillstyle ='none')
plt.plot(false_neg_x, false_neg_y, 'xb', fillstyle ='none')
plt.plot(svm_pos_x, svm_pos_y, 'sb', fillstyle = None)
plt.plot(svm_neg_x, svm_neg_y, 'sr', fillstyle = None)

plt.axis('equal')
plt.axis('off')

h = plt.contourf(X,Y,predicted, alpha=0.3, cmap = colors.ListedColormap
(['r','b'])) #check colors
```

```
Zero-one training error rate: 5.00 %
Test error rate: 3.33 %
```

```
/Users/joycechoi/anaconda3/lib/python3.6/site-packages/ipykernel_launch
er.py:43: DeprecationWarning: object of type <class 'numpy.float64'> ca
nnot be safely interpreted as an integer.
/Users/joycechoi/anaconda3/lib/python3.6/site-packages/ipykernel_launch
er.py:44: DeprecationWarning: object of type <class 'numpy.float64'> ca
nnot be safely interpreted as an integer.
```



Problem 2.3

Each data point in T and S has two features, x_1 and x_2. Augment these by adding the following redundant features

$$
\begin{aligned}
x_3 &= x_1^2 \\
x_4 &= x_2^2 \\
x_5 &= x_1 x_2
\end{aligned}
$$

Then repeat the experiment above with `SVC(kernel='linear', C=1)` on the augmented data. Of course, plot just x_1 and x_2, not the other features, unless you have a 5D printer handy.

Answer

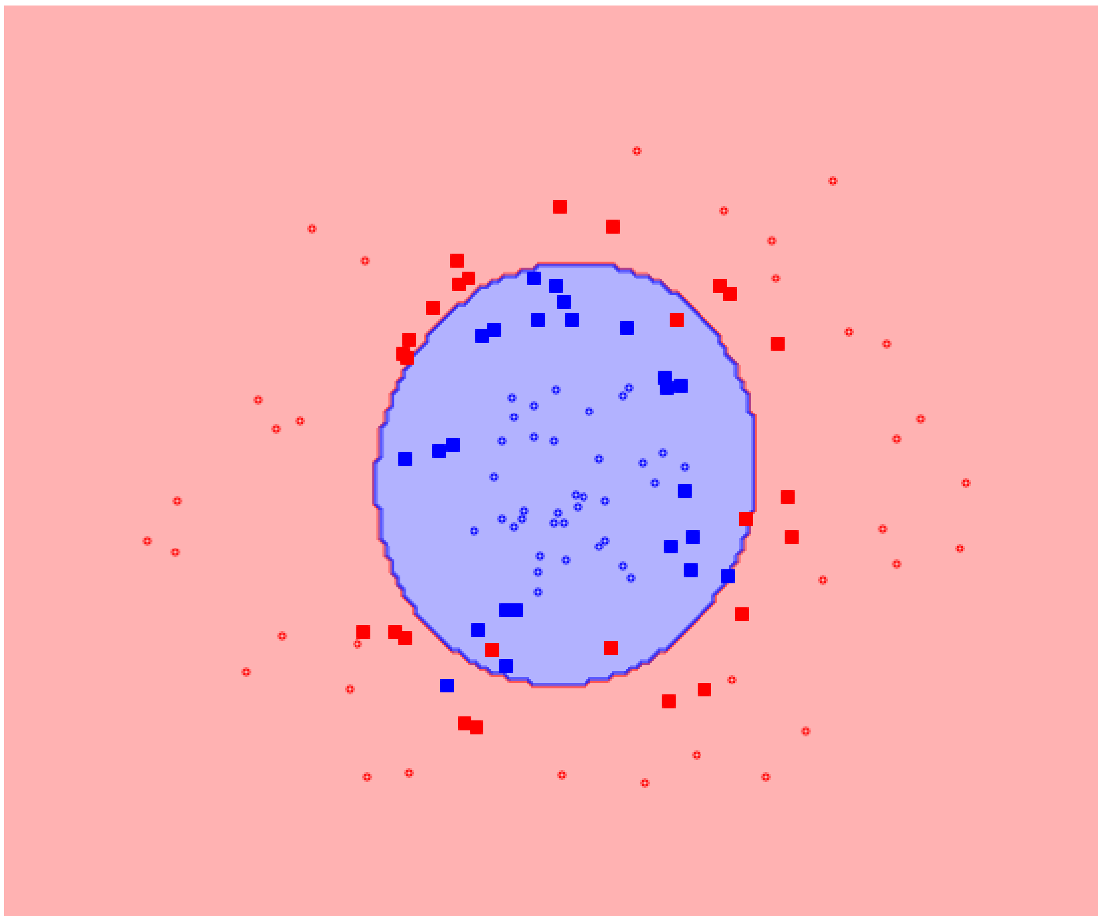
In [6]:
```
# train sklearn.svm.SVC with arguments kernel = 'linear', C = 1 on aug_T
_x (Augmented training set T )
# show its zero-one training and test error rates (on S) as percentages
 with two decimal places

# finding the max and min along each dimension for plotting the contour

x1_dimension = T['x'][:, 0] # get all the values of T['x'] along the x1
 dimension
x2_dimension = T['x'][:, 1] # get all the values of T['x'] along the x2
 dimension

max_x1 = max(x1_dimension) # max value of x1
min_x1 = min(x1_dimension) # min value of x1
max_x2 = max(x2_dimension) # max value of x2
min_x2 = min(x2_dimension) # min value of x2

# add or subtract from each max or min respectively
max_x1 = max_x1 + 0.5 # add 0.5 to the max x1 value
min_x1 = min_x1 - 0.5 # subtract 0.5 to the min x1 value
max_x2 = max_x2 + 0.5 # add 0.5 to the max x2 value
min_x2 = min_x2 - 0.5 # subtract 0.5 from the min x2 value

x1_range = np.linspace(min_x1, max_x1, (max_x1 - min_x1) / 0.02) # creat
e a list of the x1 range
x2_range = np.linspace(min_x2, max_x2, (max_x2 - min_x2) / 0.02) # creat
e a list of the x2 range
X1, X2 = np.meshgrid(x1_range, x2_range)                         # creat
e the meshgrid of X1, X2

#flatten X1 and X2 to make it useable in np.stack
X1_flat = np.concatenate(X1).ravel()
X2_flat = np.concatenate(X2).ravel()

#stack X1 and X2. Creates an array an (dim(X1) x dim(X2), 2) array
stack = np.stack((X1_flat, X2_flat), axis=-1)

# augment the stack with x3, x4, and x5
stack_length = len(stack)                               # stack length or nu
mber of entries in stack
augmented_training_set = np.zeros((len(T['x']), 5)) # the same length as
 T['x'] but 5-d as opposed to 2-d
augmented_test_set = np.zeros((len(S['x']), 5))     # the same length as
 S['x'] but 5-d as opposed to 2-d
augmented_T_length = len(augmented_training_set)    # length of augmente
d T
augmented_S_length = len(augmented_test_set)        # length of augmente
d S
augmented_stack = np.zeros((stack_length, 5))       # initialize the aug
mented stack

# loop through the stack to create x3, x4, and x5 and put those values i
nto the augmented stack
for sample_index in range(stack_length):
    x1 = stack[sample_index][0]
    x2 = stack[sample_index][1]
```

```
    x3 = x1**2
    x4 = x2**2
    x5 = x1*x2
    augmented_stack[sample_index][0] = x1
    augmented_stack[sample_index][1] = x2
    augmented_stack[sample_index][2] = x3
    augmented_stack[sample_index][3] = x4
    augmented_stack[sample_index][4] = x5

for sample_index in range(augmented_T_length):
    sample = T['x'][sample_index]
    x1 = sample[0] # looks at the first coordinate x1 of the sample at s
ample_index in T['x']
    x2 = sample[1] # looks at the second coordinate x2 of the sample at
 sample_index in T['x']
    x3 = x1**2
    x4 = x2**2
    x5 = x1*x2
    augmented_training_set[sample_index][0] = x1
    augmented_training_set[sample_index][1] = x2
    augmented_training_set[sample_index][2] = x3
    augmented_training_set[sample_index][3] = x4
    augmented_training_set[sample_index][4] = x5

for sample_index in range(augmented_S_length):
    sample = S['x'][sample_index]
    x1 = sample[0] # looks at the first coordinate x1 of the sample at s
ample_index in S['x']
    x2 = sample[1] # looks at the second coordinate x2 of the sample at
 sample_index in S['x']
    x3 = x1**2
    x4 = x2**2
    x5 = x1*x2
    augmented_test_set[sample_index][0] = x1
    augmented_test_set[sample_index][1] = x2
    augmented_test_set[sample_index][2] = x3
    augmented_test_set[sample_index][3] = x4
    augmented_test_set[sample_index][4] = x5

clf = SVC(C = 1, kernel = 'linear')
  # training a linear Support Vector Classification
clf.fit(augmented_training_set, T['y'])
  # fit the classifier to the training data
zero_one_training_rate = 1 - clf.score(augmented_training_set, T['y'])
  # train the classifier on augmented training set aug_T_x and get error
 rate
test_error_rate = 1 - clf.score(augmented_test_set, S['y'])
  # test the classifier on the augmented test set aug_S_x and get error
 rate

print("Zero-one training error rate: {:.2f}".format(zero_one_training_ra
te*100), "%")
print("Test error rate: {:.2f}".format(test_error_rate*100), "%")

predicted = clf.predict(augmented_stack).reshape(X1.shape) # train the c
lassifier on the augmented stack
predicted_labels = clf.predict(augmented_training_set)     # get the pre
```

```
dictions of the classifier on the augmented training set

# initialize the different categories of data points

true_pos_x = []
true_pos_y = []

true_neg_x = []
true_neg_y = []

false_pos_x = []
false_pos_y = []

false_neg_x = []
false_neg_y = []

svm_pos_x = []
svm_pos_y = []
svm_neg_x = []
svm_neg_y = []

svm_indices = clf.support_ # indices for the support vectors

# correctly place each data point into one of the 6 categories:
# true positive, true negative, false positive, false negatives, svm pos
itive, svm negative

size_of_T = len(augmented_training_set)
for sample_index in range(size_of_T):
    sample = augmented_training_set[sample_index]                  # sa
mple coordinate
    true_label = T['y'][sample_index]                # true label
    predicted_label = predicted_labels[sample_index] # predicted label

    # check svm_indices
    if sample_index in svm_indices:
        # current sample is an svm if this goes through
        if true_label == 1:              # put the sample into the posit
ive svm list
            svm_pos_x.append(sample[0])
            svm_pos_y.append(sample[1])
        if true_label != 1:              # put the sample into the negat
ive svm list
            svm_neg_x.append(sample[0])
            svm_neg_y.append(sample[1])

    # true positive
    if true_label == 1 and predicted_label == 1:
        true_pos_x.append(sample[0])
        true_pos_y.append(sample[1])
    # true negative
    if true_label != 1 and predicted_label != 1:
        true_neg_x.append(sample[0])
        true_neg_y.append(sample[1])
    # false positive
    if true_label != 1 and predicted_label == 1:
        false_pos_x.append(sample[0])
```

```
        false_pos_y.append(sample[1])
    # false negatives
    if true_label == 1 and predicted_label != 1:
        false_neg_x.append(sample[0])
        false_neg_y.append(sample[1])



plt.figure(figsize=(10,10))

# plot the data points from the 6 categories
plt.plot(true_pos_x, true_pos_y, '.b', fillstyle ='none')
plt.plot(true_neg_x, true_neg_y, '.r', fillstyle ='none')
plt.plot(false_pos_x, false_pos_y, 'xr', fillstyle ='none')
plt.plot(false_neg_x, false_neg_y, 'xb', fillstyle ='none')
plt.plot(svm_pos_x, svm_pos_y, 'sb', fillstyle = None)
plt.plot(svm_neg_x, svm_neg_y, 'sr', fillstyle = None)

# specifiy some plotting things
plt.axis('equal')
plt.axis('off')

# plot the decision boundary
h = plt.contourf(X1,X2,predicted, alpha=0.3, cmap = colors.ListedColorma
p(['r','b'])) #check colors
```

```
/Users/joycechoi/anaconda3/lib/python3.6/site-packages/ipykernel_launch
er.py:20: DeprecationWarning: object of type <class 'numpy.float64'> ca
nnot be safely interpreted as an integer.
/Users/joycechoi/anaconda3/lib/python3.6/site-packages/ipykernel_launch
er.py:21: DeprecationWarning: object of type <class 'numpy.float64'> ca
nnot be safely interpreted as an integer.

Zero-one training error rate: 4.17 %
Test error rate: 3.89 %
```



Problem 2.4

Explain carefully why this type of data augmentation works well for this specific data set.

Answer

When we change the dimensions of the data to 5 dimensions, the points are still plotted in the same locations but now the linear classifier has 3 more dimensions worth of information which is why it's able to more closely follow the data and make a circular curve.

In this case, we augment with values ${x_1}^2, {x_2}^2, (x_1) * (x_2)$, which are the transformations that give an ellipse (as ${x_1}^2 + {x_2}^2 + (x_1) * (x_2) = r$is the equation for an ellipse). Therefore, augmenting the original points allows us to get a distribution of points that are separable in a higher dimension, which is why augmenting the data works to generate the SVM in this particular data set.

Problem 2.5

Try the experiment in Problem 2.3 with `sklearn.linear_model.LogisticRegression`. Use parameters `C=1e5, solver='lbfgs', multi_class='multinomial', random_state=0`.

Answer

In [7]:
```
from sklearn.linear_model import LogisticRegression

# train sklearn.svm.SVC with arguments kernel = 'linear', C = 1 on aug_T
_x (Augmented training set T )
# show its zero-one training and test error rates (on S) as percentages
 with two decimal places

# finding the max and min along each dimension for plotting the contour

x1_dimension = T['x'][:, 0] # get all the values of T['x'] along the x1
 dimension
x2_dimension = T['x'][:, 1] # get all the values of T['x'] along the x2
 dimension

max_x1 = max(x1_dimension) # max value of x1
min_x1 = min(x1_dimension) # min value of x1
max_x2 = max(x2_dimension) # max value of x2
min_x2 = min(x2_dimension) # min value of x2

# add or subtract from each max or min respectively
max_x1 = max_x1 + 0.5 # add 0.5 to the max x1 value
min_x1 = min_x1 - 0.5 # subtract 0.5 to the min x1 value
max_x2 = max_x2 + 0.5 # add 0.5 to the max x2 value
min_x2 = min_x2 - 0.5 # subtract 0.5 from the min x2 value

x1_range = np.linspace(min_x1, max_x1, (max_x1 - min_x1) / 0.02) # creat
e a list of the x1 range
x2_range = np.linspace(min_x2, max_x2, (max_x2 - min_x2) / 0.02) # creat
e a list of the x2 range
X1, X2 = np.meshgrid(x1_range, x2_range)                         # creat
e the meshgrid of X1, X2

#flatten X1 and X2 to make it useable in np.stack
X1_flat = np.concatenate(X1).ravel()
X2_flat = np.concatenate(X2).ravel()

#stack X1 and X2. Creates an array an (dim(X1) x dim(X2), 2) array
stack = np.stack((X1_flat, X2_flat), axis=-1)

# augment the stack with x3, x4, and x5
stack_length = len(stack)                               # stack length or nu
mber of entries in stack
augmented_training_set = np.zeros((len(T['x']), 5)) # the same length as
 T['x'] but 5-d as opposed to 2-d
augmented_test_set = np.zeros((len(S['x']), 5))     # the same length as
 S['x'] but 5-d as opposed to 2-d
augmented_T_length = len(augmented_training_set)    # length of augmente
d T
augmented_S_length = len(augmented_test_set)        # length of augmente
d S
augmented_stack = np.zeros((stack_length, 5))       # initialize the aug
mented stack

# loop through the stack to create x3, x4, and x5 and put those values i
nto the augmented stack
for sample_index in range(stack_length):
```

```
    x1 = stack[sample_index][0]
    x2 = stack[sample_index][1]
    x3 = x1**2
    x4 = x2**2
    x5 = x1*x2
    augmented_stack[sample_index][0] = x1
    augmented_stack[sample_index][1] = x2
    augmented_stack[sample_index][2] = x3
    augmented_stack[sample_index][3] = x4
    augmented_stack[sample_index][4] = x5

for sample_index in range(augmented_T_length):
    sample = T['x'][sample_index]
    x1 = sample[0] # looks at the first coordinate x1 of the sample at s
ample_index in T['x']
    x2 = sample[1] # looks at the second coordinate x2 of the sample at
 sample_index in T['x']
    x3 = x1**2
    x4 = x2**2
    x5 = x1*x2
    augmented_training_set[sample_index][0] = x1
    augmented_training_set[sample_index][1] = x2
    augmented_training_set[sample_index][2] = x3
    augmented_training_set[sample_index][3] = x4
    augmented_training_set[sample_index][4] = x5

for sample_index in range(augmented_S_length):
    sample = S['x'][sample_index]
    x1 = sample[0] # looks at the first coordinate x1 of the sample at s
ample_index in S['x']
    x2 = sample[1] # looks at the second coordinate x2 of the sample at
 sample_index in S['x']
    x3 = x1**2
    x4 = x2**2
    x5 = x1*x2
    augmented_test_set[sample_index][0] = x1
    augmented_test_set[sample_index][1] = x2
    augmented_test_set[sample_index][2] = x3
    augmented_test_set[sample_index][3] = x4
    augmented_test_set[sample_index][4] = x5

clf = LogisticRegression(C=1e5, solver='lbfgs', multi_class='multinomia
l', random_state=0)
clf.fit(augmented_training_set, T['y'])
  # fit the classifier to the training data
zero_one_training_rate = 1 - clf.score(augmented_training_set, T['y'])
  # train the classifier on augmented training set aug_T_x and get error
 rate
test_error_rate = 1 - clf.score(augmented_test_set, S['y'])
  # test the classifier on the augmented test set aug_S_x and get error
 rate

print("Zero-one training error rate: {:.2f}".format(zero_one_training_ra
te*100), "%")
print("Test error rate: {:.2f}".format(test_error_rate*100), "%")

predicted = clf.predict(augmented_stack).reshape(X1.shape) # train the c
```

```
lassifier on the augmented stack
predicted_labels = clf.predict(augmented_training_set)     # get the pre
dictions of the classifier on the augmented training set

# initialize the different categories of data points

true_pos_x = []
true_pos_y = []

true_neg_x = []
true_neg_y = []

false_pos_x = []
false_pos_y = []

false_neg_x = []
false_neg_y = []

# correctly place each data point into one of the 4 categories:
# true positive, true negative, false positive, false negatives

size_of_T = len(augmented_training_set)
for sample_index in range(size_of_T):
    sample = augmented_training_set[sample_index]                      # sa
mple coordinate
    true_label = T['y'][sample_index]               # true label
    predicted_label = predicted_labels[sample_index] # predicted label

    # true positive
    if true_label == 1 and predicted_label == 1:
        true_pos_x.append(sample[0])
        true_pos_y.append(sample[1])
    # true negative
    if true_label != 1 and predicted_label != 1:
        true_neg_x.append(sample[0])
        true_neg_y.append(sample[1])
    # false positive
    if true_label != 1 and predicted_label == 1:
        false_pos_x.append(sample[0])
        false_pos_y.append(sample[1])
    # false negatives
    if true_label == 1 and predicted_label != 1:
        false_neg_x.append(sample[0])
        false_neg_y.append(sample[1])

plt.figure(figsize=(10,10))

# plot the data points from the 6 categories
plt.plot(true_pos_x, true_pos_y, '.b', fillstyle ='none')
plt.plot(true_neg_x, true_neg_y, '.r', fillstyle ='none')
plt.plot(false_pos_x, false_pos_y, 'xr', fillstyle ='none')
plt.plot(false_neg_x, false_neg_y, 'xb', fillstyle ='none')

# specifiy some plotting things
plt.axis('equal')
plt.axis('off')
```

```
# plot the decision boundary
h = plt.contourf(X1,X2,predicted, alpha=0.3, cmap = colors.ListedColorma
p(['r','b'])) #check colors
```

```
Zero-one training error rate: 4.17 %
Test error rate: 5.56 %
```

```
/Users/joycechoi/anaconda3/lib/python3.6/site-packages/ipykernel_launch
er.py:22: DeprecationWarning: object of type <class 'numpy.float64'> ca
nnot be safely interpreted as an integer.
/Users/joycechoi/anaconda3/lib/python3.6/site-packages/ipykernel_launch
er.py:23: DeprecationWarning: object of type <class 'numpy.float64'> ca
nnot be safely interpreted as an integer.
```