CMPE 257:03 -Homework 1 Learning from data Problem 1.4 and 1.5

(a) Generate a linearly separated data set of size 20. Plot the examples (xn,yn) as well as target function f on a plane. Be sure to mark the examples from different classes differently and add lables to the plot.

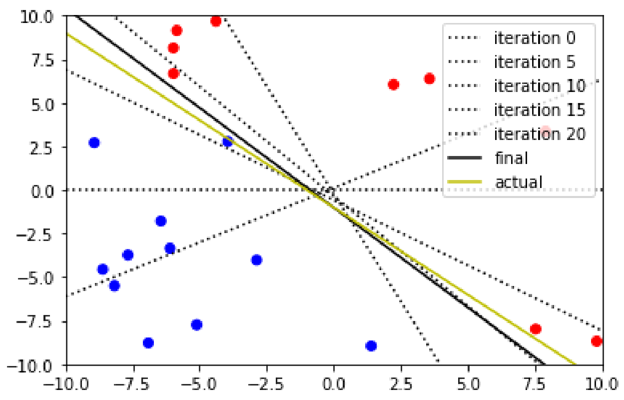
(b) Run the perceptron learning algorithm on the data set above. Report the number of updates that the algorithm takes before converging. plot the examples {(xn,yn)}, the target function f, and the final hypothesis g in the same figure. Comment on wheteher f is close to g.

In [26]:

```
import matplotlib.pyplot as plt
import numpy as np

# generate a linearly separated data of size 20-

w_act = np.array([1, 1, 1])
w_init = np.array([3, -50, 0])
d = 2
num_sample = 20
data_range = 10

X_train = np.vstack([
    np.ones(num_sample),
    np.random.uniform(
        -data_range,
        data_range,
        (d, num_sample),
    ),
])

y_train = evaluate_h(w_act, X_train)
x_axis = X_train[1, :]
y_axis = X_train[2, :]
lables = ['r' if y > 0 else 'b' for y in y_train]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])

# check if the hypothesis h from the hypothesis set H is following the weight and dimensionality c
ondition
def evaluate_h(w, X):

    '''
    Evaluate the hypothesis, h \in H, which is described by its set of weights,
    w, against 1 or more points in the input space (stored in a matrix).
    '''
    assert len(w.shape) == 1
    assert len(X.shape) == 2
    assert w.shape[0] == X.shape[0]
    return np.sign(w @ X)

# Run perceptron learning algorithm for the training data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
        if np.all(correct):
# Print the total number of iterations that let to the generation of final hypothesis g
            print('Total number of iterations: ',n)
            return w
        else:
            i = np.argmax(~correct)  # indice of first misclassified point
            w = w + y_train[i]*X_train[:, i]
        n = n + 1


def plot_hypothesis(x_axis, w, *plot_args, **plot_kwargs):
    m = -w[1]/w[2] if w[2] != 0 else 0
```

```
    b = -w[0]/w[2] if w[2] != 0 else 0
    y_axis = m*x_axis + b
    plt.plot(x_axis, y_axis, *plot_args, **plot_kwargs)

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)

w_final = run_perceptron(w_init, X_train, y_train, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
plt.show()
```

```
Total number of iterations:  24
```



In the above case, we can see that it took 24 iterations for the generation of final hypothesis g. As we can see that the final hyperplane represented by the black line is separating the points perfectly and there are no misclassified points. Hence, f is close to g.

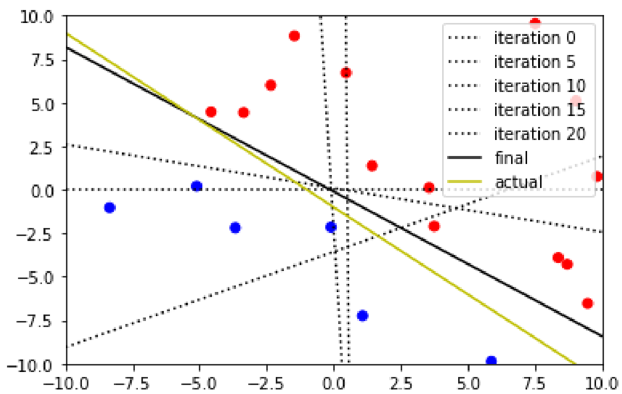
(c) Repeat the step with a different set of randonly generated data of size 20 and compare the results with (b)

In [27]:

```
import matplotlib.pyplot as plt
import numpy as np

# generate a linearly separated data of size 20-

w_act = np.array([1, 1, 1])
w_init = np.array([3, -50, 0])
d = 2
num_sample = 20
data_range = 10

X_train = np.vstack([
    np.ones(num_sample),
    np.random.uniform(
        -data_range,
        data_range,
        (d, num_sample),
    ),
])

y_train = evaluate_h(w_act, X_train)
x_axis = X_train[1, :]
y_axis = X_train[2, :]
lables = ['r' if y > 0 else 'b' for y in y_train]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])

# check if the hypothesis h from the hypothesis set H is following the weight and dimensionality c
ondition
```

```
def evaluate_h(w, X):
    '''
    Evaluate the hypothesis, h \in H, which is described by its set of weights,
    w, against 1 or more points in the input space (stored in a matrix).
    '''
    assert len(w.shape) == 1
    assert len(X.shape) == 2
    assert w.shape[0] == X.shape[0]
    return np.sign(w @ X)

# Run perceptron learning algorithm for the training data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
        if np.all(correct):
# Print the total number of iterations that let to the generation of final hypothesis g
            print('Total number of iterations: ',n)
            return w
        else:
            i = np.argmax(~correct)  # indice of first misclassified point
            w = w + y_train[i]*X_train[:, i]
        n = n + 1


def plot_hypothesis(x_axis, w, *plot_args, **plot_kwargs):
    m = -w[1]/w[2] if w[2] != 0 else 0
    b = -w[0]/w[2] if w[2] != 0 else 0
    y_axis = m*x_axis + b
    plt.plot(x_axis, y_axis, *plot_args, **plot_kwargs)

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)

w_final = run_perceptron(w_init, X_train, y_train, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
plt.show()
```

```
Total number of iterations:  22
```



In the above plot, we can see that with a new set of data, we are able to generate a good approximation of final hypothesis and it takes 22 iterations to classify the points correctly in this case.

(d) Repeat the step with a different set of randonly generated data of size 100 and compare the results with (b)

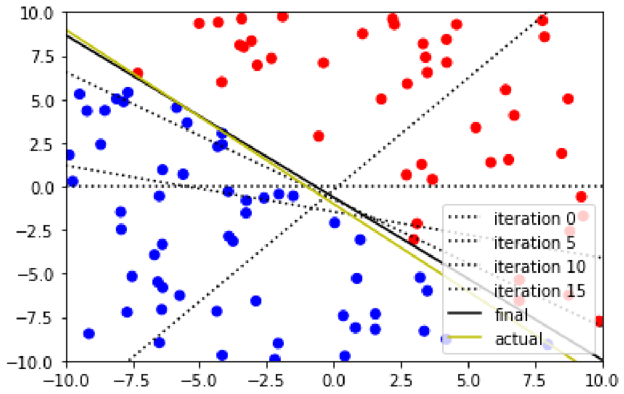
In [31]:

In [31]:

```
import matplotlib.pyplot as plt
import numpy as np

# generate a linearly separated data of size 100-

w_act = np.array([1, 1, 1])
w_init = np.array([3, -50, 0])
d = 2
num_sample = 100
data_range = 10

X_train = np.vstack([
    np.ones(num_sample),
    np.random.uniform(
        -data_range,
        data_range,
        (d, num_sample),
    ),
])

y_train = evaluate_h(w_act, X_train)
x_axis = X_train[1, :]
y_axis = X_train[2, :]
lables = ['r' if y > 0 else 'b' for y in y_train]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])

# check if the hypothesis h from the hypothesis set H is following the weight and dimensionality c
ondition
def evaluate_h(w, X):

    '''
    Evaluate the hypothesis, h \in H, which is described by its set of weights,
    w, against 1 or more points in the input space (stored in a matrix).
    '''
    assert len(w.shape) == 1
    assert len(X.shape) == 2
    assert w.shape[0] == X.shape[0]
    return np.sign(w @ X)

# Run perceptron learning algorithm for the training data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
        if np.all(correct):
# Print the total number of iterations that let to the generation of final hypothesis g
            print('Total number of iterations: ',n)
            return w
        else:
            i = np.argmax(~correct)  # indice of first misclassified point
            w = w + y_train[i]*X_train[:, i]
        n = n + 1


def plot_hypothesis(x_axis, w, *plot_args, **plot_kwargs):
    m = -w[1]/w[2] if w[2] != 0 else 0
    b = -w[0]/w[2] if w[2] != 0 else 0
    y_axis = m*x_axis + b
    plt.plot(x_axis, y_axis, *plot_args, **plot_kwargs)

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)

w_final = run_perceptron(w_init, X_train, y_train, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual')
plt.legend()
plt.xlim(-data_range, data_range)
```

```
plt.ylim(-data_range, data_range)
plt.show()
```

```
Total number of iterations:  19
```



In the above plot, we can see that with a new set of 100 data, we are able to generate a good approximation of final hypothesis and it takes 19 iterations to classify the points correctly in this case.

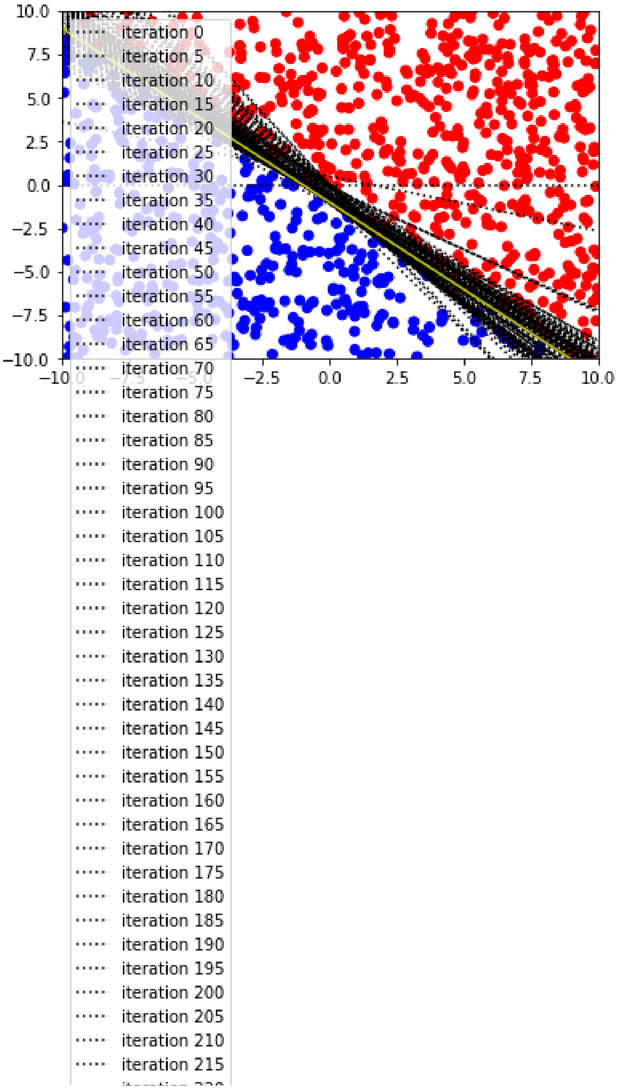
(e)Repeat the step with a different set of randonly generated data of size 1000 and compare the results with (b)

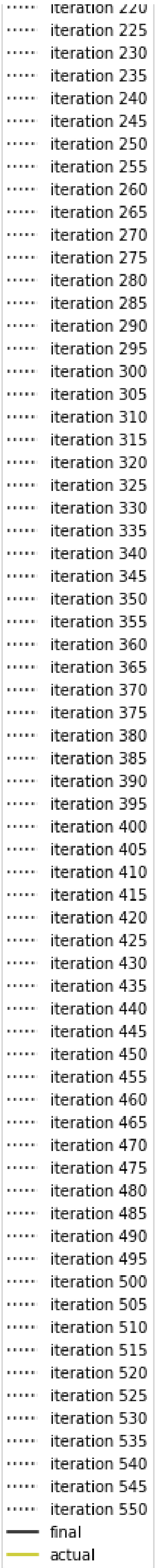
In [32]:

```
import matplotlib.pyplot as plt
import numpy as np

# generate a linearly separated data of size 1000-

w_act = np.array([1, 1, 1])
w_init = np.array([3, -50, 0])
d = 2
num_sample = 1000
data_range = 10

X_train = np.vstack([
    np.ones(num_sample),
    np.random.uniform(
        -data_range,
        data_range,
        (d, num_sample),
    ),
])

y_train = evaluate_h(w_act, X_train)
x_axis = X_train[1, :]
y_axis = X_train[2, :]
lables = ['r' if y > 0 else 'b' for y in y_train]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])

# check if the hypothesis h from the hypothesis set H is following the weight and dimensionality c
ondition
def evaluate_h(w, X):

    '''
    Evaluate the hypothesis, h \in H, which is described by its set of weights,
    w, against 1 or more points in the input space (stored in a matrix).
    '''
    assert len(w.shape) == 1
    assert len(X.shape) == 2
    assert w.shape[0] == X.shape[0]
    return np.sign(w @ X)

# Run perceptron learning algorithm for the training data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
```

```python
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
        if np.all(correct):
# Print the total number of iterations that let to the generation of final hypothesis g
            print('Total number of iterations: ',n)
            return w
        else:
            i = np.argmax(~correct)   # indice of first misclassified point
            w = w + y_train[i]*X_train[:, i]
        n = n + 1


def plot_hypothesis(x_axis, w, *plot_args, **plot_kwargs):
    m = -w[1]/w[2] if w[2] != 0 else 0
    b = -w[0]/w[2] if w[2] != 0 else 0
    y_axis = m*x_axis + b
    plt.plot(x_axis, y_axis, *plot_args, **plot_kwargs)

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)

w_final = run_perceptron(w_init, X_train, y_train, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
plt.show()
```

```
Total number of iterations:  554
```





In the above plot, we can see that with a new set of 100 data, we are able to generate a good approximation of final hypothesis and it takes 554 iterations to classify the points correctly in this case.

Problem 1.5 - The perceptron learning algorithm works like this: in each iteration t, pick a randon (x(t), y(t)) and compute the 'signal' s(t) = wT(t) x(t). if y(t).s(t) <=0, update w by w(t+1) <- w(t) + y(t). x(t);

One may say that the algorithm does not take the closeness between s(t) and y(t) into account. so if y(t). s(t) <= 1, update w by : w(t+1) <= w(t) + n (y(t) - s(t)).x(t)

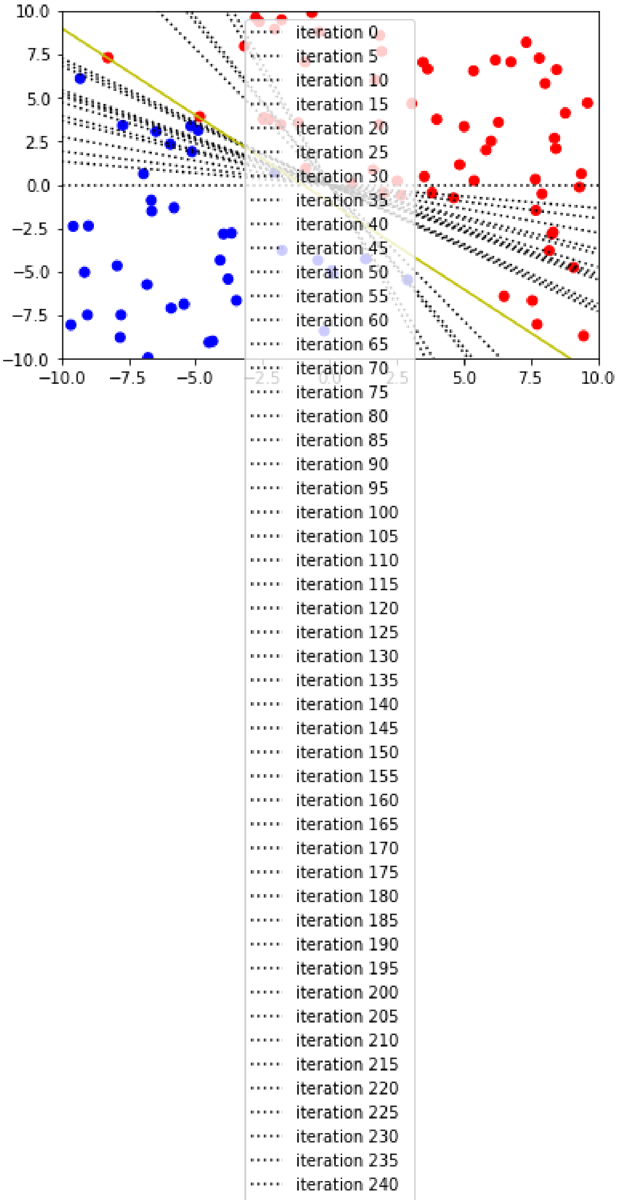
(a) Generate training set of 100 and test set of 1000. run the algorithm by taking n =100 on training set until 1000 updates. plot training set and target function. Report the error on test set

In [104]:

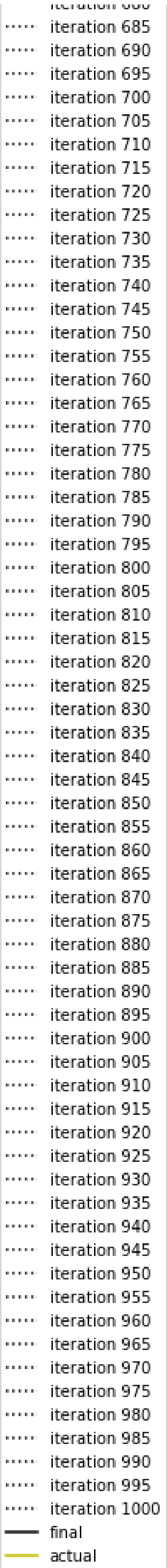
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# generate a linearly separated training data of size 100 and test data of size 1000 -
w_act = np.array([1, 1, 1])
w_init = np.array([3, -50, 0])
d = 2
num_sample = 100
data_range = 10
num_sample_test =1000
error = 0
# check if the hypothesis h from the hypothesis set H is following the weight and dimensionality c
ondition
def evaluate_h(w, X):

    '''
    Evaluate the hypothesis, h \in H, which is described by its set of weights,
    w, against 1 or more points in the input space (stored in a matrix).
    '''
    assert len(w.shape) == 1
    assert len(X.shape) == 2
    assert w.shape[0] == X.shape[0]
    return np.sign(w @ X)
X_train = np.vstack([
    np.ones(num_sample),
    np.random.uniform(
        -data_range,
        data_range,
        (d, num_sample),
    ),
])

X_test = np.vstack([
    np.ones(num_sample_test),
    np.random.uniform(
        -data_range,
        data_range,
        (d, num_sample_test),
    ),
])

y_train = evaluate_h(w_act, X_train)
y_test = evaluate_h(w_act, X_test)
x_axis = X_train[1, :]
y_axis = X_train[2, :]
lables = ['r' if y > 0 else 'b' for y in y_train]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])


# Run perceptron learning algorithm for the training data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
# Calculate error after 1000 updates
        if n == 1000:
            return w

        else:
# calculate weight considering the closeness between s and y_train
            i = np.argmax(~correct)  # indice of first misclassified point
            s = w * X_train[:,i]
            d = y_train[i] * s
            d = d[0] + d[1] + d[2]
            if d <=1:
                w = w + 100 * (y_train[i] - s) * X_train[:, i]
                error = (y_train[i] - s)

        n = n + 1
```

```python
def plot_hypothesis(x_axis, w, *plot_args, **plot_kwargs):
    m = -w[1]/w[2] if w[2] != 0 else 0
    b = -w[0]/w[2] if w[2] != 0 else 0
    y_axis = m*x_axis + b
    plt.plot(x_axis, y_axis, *plot_args, **plot_kwargs)

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)

w_final = run_perceptron(w_init, X_train, y_train, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
plt.show()
```

```
C:\Users\kashi\Anaconda3\lib\site-packages\ipykernel_launcher.py:71: RuntimeWarning: overflow
encountered in multiply
C:\Users\kashi\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: RuntimeWarning: invalid value
encountered in sign
C:\Users\kashi\Anaconda3\lib\site-packages\ipykernel_launcher.py:78: RuntimeWarning: invalid value
encountered in double_scalars
C:\Users\kashi\Anaconda3\lib\site-packages\ipykernel_launcher.py:71: RuntimeWarning: invalid value
encountered in add
```



····· iteration 245
····· iteration 250
····· iteration 255
····· iteration 260
····· iteration 265
····· iteration 270
····· iteration 275
····· iteration 280
····· iteration 285
····· iteration 290
····· iteration 295
····· iteration 300
····· iteration 305
····· iteration 310
····· iteration 315
····· iteration 320
····· iteration 325
····· iteration 330
····· iteration 335
····· iteration 340
····· iteration 345
····· iteration 350
····· iteration 355
····· iteration 360
····· iteration 365
····· iteration 370
····· iteration 375
····· iteration 380
····· iteration 385
····· iteration 390
····· iteration 395
····· iteration 400
····· iteration 405
····· iteration 410
····· iteration 415
····· iteration 420
····· iteration 425
····· iteration 430
····· iteration 435
····· iteration 440
····· iteration 445
····· iteration 450
····· iteration 455
····· iteration 460
····· iteration 465
····· iteration 470
····· iteration 475
····· iteration 480
····· iteration 485
····· iteration 490
····· iteration 495
····· iteration 500
····· iteration 505
····· iteration 510
····· iteration 515
····· iteration 520
····· iteration 525
····· iteration 530
····· iteration 535
····· iteration 540
····· iteration 545
····· iteration 550
····· iteration 555
····· iteration 560
····· iteration 565
····· iteration 570
····· iteration 575
····· iteration 580
····· iteration 585
····· iteration 590
····· iteration 595
····· iteration 600
····· iteration 605
····· iteration 610
····· iteration 615
····· iteration 620
····· iteration 625
····· iteration 630
····· iteration 635
····· iteration 640
····· iteration 645
····· iteration 650
····· iteration 655
····· iteration 660
····· iteration 665
····· iteration 670
····· iteration 675
····· iteration 680



The plot above is the plot for training data set for n = 100.

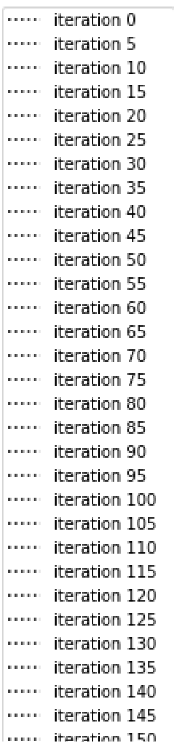
In [111]:

```
# Run perceptron learning algorithm for the test data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
#        if np.all(correct):
# Print the total number of iterations that let to the generation of final hypothesis g
```

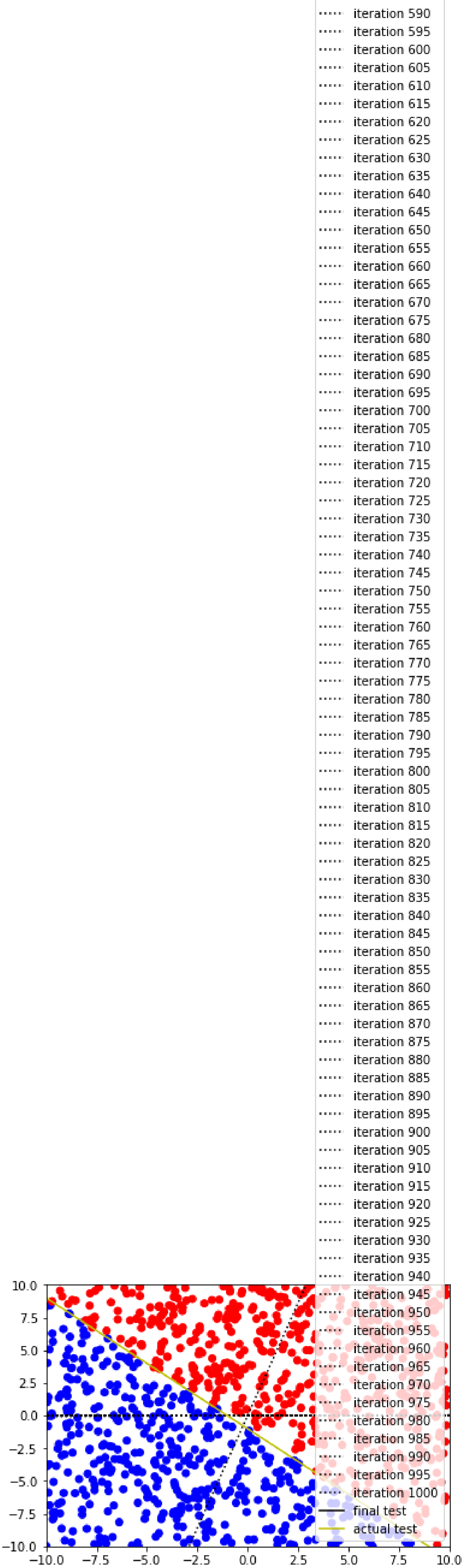
```
#             print('Total number of iterations: ',n)
#             return w
        if n == 1000:
            error = np.absolute(error)
            print("Error on test data for n = 100: ",error[0]/1000)
            return w

        else:

            i = np.argmax(~correct)  # indice of first misclassified point
            s = w * X_train[:,i]
            d = y_train[i] * s
            d = d[0] + d[1] + d[2]
            if d <=1:
                w = w + 100 * (y_train[i] - s) * X_train[:, i]
                error = (y_train[i] - s)

        n = n + 1
x_axis = X_test[1, :]
y_axis = X_test[2, :]
lables = ['r' if y > 0 else 'b' for y in y_test]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)
# Run Perceptron learning algorith for test data set

w_final = run_perceptron(w_init, X_test, y_test, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final test')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual test')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
plt.show()
```

```
C:\Users\kashi\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: RuntimeWarning: overflow encountered in multiply
C:\Users\kashi\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: RuntimeWarning: invalid value encountered in add
C:\Users\kashi\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: RuntimeWarning: invalid value encountered in sign
```

```
Error on test data for n = 100:  8.424947340550924e+166
```



····· iteration 155
····· iteration 160
····· iteration 165
····· iteration 170
····· iteration 175
····· iteration 180
····· iteration 185
····· iteration 190
····· iteration 195
····· iteration 200
····· iteration 205
····· iteration 210
····· iteration 215
····· iteration 220
····· iteration 225
····· iteration 230
····· iteration 235
····· iteration 240
····· iteration 245
····· iteration 250
····· iteration 255
····· iteration 260
····· iteration 265
····· iteration 270
····· iteration 275
····· iteration 280
····· iteration 285
····· iteration 290
····· iteration 295
····· iteration 300
····· iteration 305
····· iteration 310
····· iteration 315
····· iteration 320
····· iteration 325
····· iteration 330
····· iteration 335
····· iteration 340
····· iteration 345
····· iteration 350
····· iteration 355
····· iteration 360
····· iteration 365
····· iteration 370
····· iteration 375
····· iteration 380
····· iteration 385
····· iteration 390
····· iteration 395
····· iteration 400
····· iteration 405
····· iteration 410
····· iteration 415
····· iteration 420
····· iteration 425
····· iteration 430
····· iteration 435
····· iteration 440
····· iteration 445
····· iteration 450
····· iteration 455
····· iteration 460
····· iteration 465
····· iteration 470
····· iteration 475
····· iteration 480
····· iteration 485
····· iteration 490
····· iteration 495
····· iteration 500
····· iteration 505
····· iteration 510
····· iteration 515
····· iteration 520
····· iteration 525
····· iteration 530
····· iteration 535
····· iteration 540
····· iteration 545
····· iteration 550
····· iteration 555
····· iteration 560
····· iteration 565
····· iteration 570
····· iteration 575
····· iteration 580
····· iteration 585


iteration 590
iteration 595
iteration 600
iteration 605
iteration 610
iteration 615
iteration 620
iteration 625
iteration 630
iteration 635
iteration 640
iteration 645
iteration 650
iteration 655
iteration 660
iteration 665
iteration 670
iteration 675
iteration 680
iteration 685
iteration 690
iteration 695
iteration 700
iteration 705
iteration 710
iteration 715
iteration 720
iteration 725
iteration 730
iteration 735
iteration 740
iteration 745
iteration 750
iteration 755
iteration 760
iteration 765
iteration 770
iteration 775
iteration 780
iteration 785
iteration 790
iteration 795
iteration 800
iteration 805
iteration 810
iteration 815
iteration 820
iteration 825
iteration 830
iteration 835
iteration 840
iteration 845
iteration 850
iteration 855
iteration 860
iteration 865
iteration 870
iteration 875
iteration 880
iteration 885
iteration 890
iteration 895
iteration 900
iteration 905
iteration 910
iteration 915
iteration 920
iteration 925
iteration 930
iteration 935
iteration 940
iteration 945
iteration 950
iteration 955
iteration 960
iteration 965
iteration 970
iteration 975
iteration 980
iteration 985
iteration 990
iteration 995
iteration 1000
final test
actual test
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

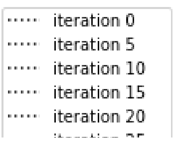
The above plot is the output of test data for the previously trained data set with n = 100. The error rate for this case is 8.42e+166 which is equivalent to 10.3

(b) The previous problem is now solved for n=1

In [121]:

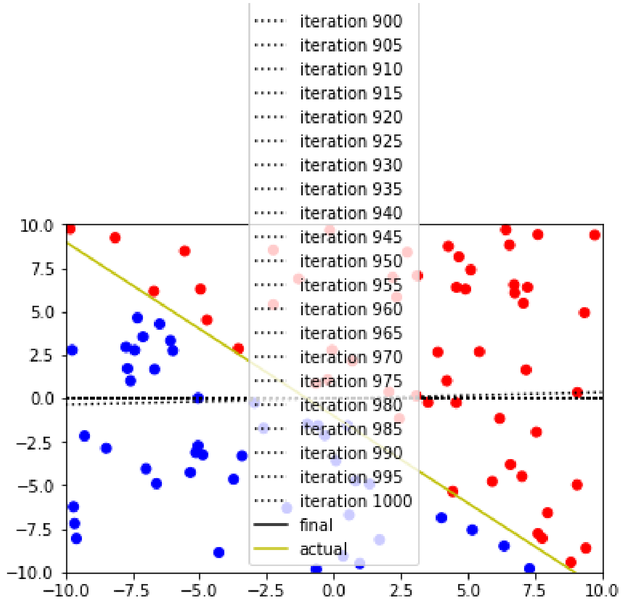
```
y_train = evaluate_h(w_act, X_train)
y_test = evaluate_h(w_act, X_test)
x_axis = X_train[1, :]
y_axis = X_train[2, :]
lables = ['r' if y > 0 else 'b' for y in y_train]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])
# Run perceptron learning algorithm for the training data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
# Calculate error after 1000 updates
        if n == 1000:
            return w

        else:
# calculate weight considering the closeness between s and y_train
            i = np.argmax(~correct)  # indice of first misclassified point
            s = w * X_train[:,i]
            d = y_train[i] * s
            d = d[0] + d[1] + d[2]
            if d <=1:
                w = w + 1 * (y_train[i] - s) * X_train[:, i]
                error = (y_train[i] - s)

        n = n + 1


def plot_hypothesis(x_axis, w, *plot_args, **plot_kwargs):
    m = -w[1]/w[2] if w[2] != 0 else 0
    b = -w[0]/w[2] if w[2] != 0 else 0
    y_axis = m*x_axis + b
    plt.plot(x_axis, y_axis, *plot_args, **plot_kwargs)

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)

w_final = run_perceptron(w_init, X_train, y_train, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
plt.show()
```

```
C:\Users\kashi\Anaconda3\lib\site-packages\ipykernel_launcher.py:28: RuntimeWarning: overflow encountered in multiply
C:\Users\kashi\Anaconda3\lib\site-packages\ipykernel_launcher.py:28: RuntimeWarning: invalid value encountered in add
C:\Users\kashi\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: RuntimeWarning: invalid value encountered in sign
```



```
·····  iteration 25
·····  iteration 30
·····  iteration 35
·····  iteration 40
·····  iteration 45
·····  iteration 50
·····  iteration 55
·····  iteration 60
·····  iteration 65
·····  iteration 70
·····  iteration 75
·····  iteration 80
·····  iteration 85
·····  iteration 90
·····  iteration 95
·····  iteration 100
·····  iteration 105
·····  iteration 110
·····  iteration 115
·····  iteration 120
·····  iteration 125
·····  iteration 130
·····  iteration 135
·····  iteration 140
·····  iteration 145
·····  iteration 150
·····  iteration 155
·····  iteration 160
·····  iteration 165
·····  iteration 170
·····  iteration 175
·····  iteration 180
·····  iteration 185
·····  iteration 190
·····  iteration 195
·····  iteration 200
·····  iteration 205
·····  iteration 210
·····  iteration 215
·····  iteration 220
·····  iteration 225
·····  iteration 230
·····  iteration 235
·····  iteration 240
·····  iteration 245
·····  iteration 250
·····  iteration 255
·····  iteration 260
·····  iteration 265
·····  iteration 270
·····  iteration 275
·····  iteration 280
·····  iteration 285
·····  iteration 290
·····  iteration 295
·····  iteration 300
·····  iteration 305
·····  iteration 310
·····  iteration 315
·····  iteration 320
·····  iteration 325
·····  iteration 330
·····  iteration 335
·····  iteration 340
·····  iteration 345
·····  iteration 350
·····  iteration 355
·····  iteration 360
·····  iteration 365
·····  iteration 370
·····  iteration 375
·····  iteration 380
·····  iteration 385
·····  iteration 390
·····  iteration 395
·····  iteration 400
·····  iteration 405
·····  iteration 410
·····  iteration 415
·····  iteration 420
·····  iteration 425
·····  iteration 430
·····  iteration 435
·····  iteration 440
·····  iteration 445
·····  iteration 450
·····  iteration 455
·····  iteration 460
```

```
····· iteration 460
····· iteration 465
····· iteration 470
····· iteration 475
····· iteration 480
····· iteration 485
····· iteration 490
····· iteration 495
····· iteration 500
····· iteration 505
····· iteration 510
····· iteration 515
····· iteration 520
····· iteration 525
····· iteration 530
····· iteration 535
····· iteration 540
····· iteration 545
····· iteration 550
····· iteration 555
····· iteration 560
····· iteration 565
····· iteration 570
····· iteration 575
····· iteration 580
····· iteration 585
····· iteration 590
····· iteration 595
····· iteration 600
····· iteration 605
····· iteration 610
····· iteration 615
····· iteration 620
····· iteration 625
····· iteration 630
····· iteration 635
····· iteration 640
····· iteration 645
····· iteration 650
····· iteration 655
····· iteration 660
····· iteration 665
····· iteration 670
····· iteration 675
····· iteration 680
····· iteration 685
····· iteration 690
····· iteration 695
····· iteration 700
····· iteration 705
····· iteration 710
····· iteration 715
····· iteration 720
····· iteration 725
····· iteration 730
····· iteration 735
····· iteration 740
····· iteration 745
····· iteration 750
····· iteration 755
····· iteration 760
····· iteration 765
····· iteration 770
····· iteration 775
····· iteration 780
····· iteration 785
····· iteration 790
····· iteration 795
····· iteration 800
····· iteration 805
····· iteration 810
····· iteration 815
····· iteration 820
····· iteration 825
····· iteration 830
····· iteration 835
····· iteration 840
····· iteration 845
····· iteration 850
····· iteration 855
····· iteration 860
····· iteration 865
····· iteration 870
····· iteration 875
····· iteration 880
····· iteration 885
····· iteration 890
····· iteration 895
```



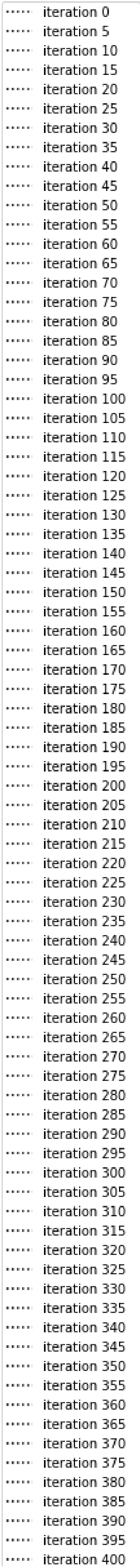
The plot above takes into account n =1 for training data

In [141]:

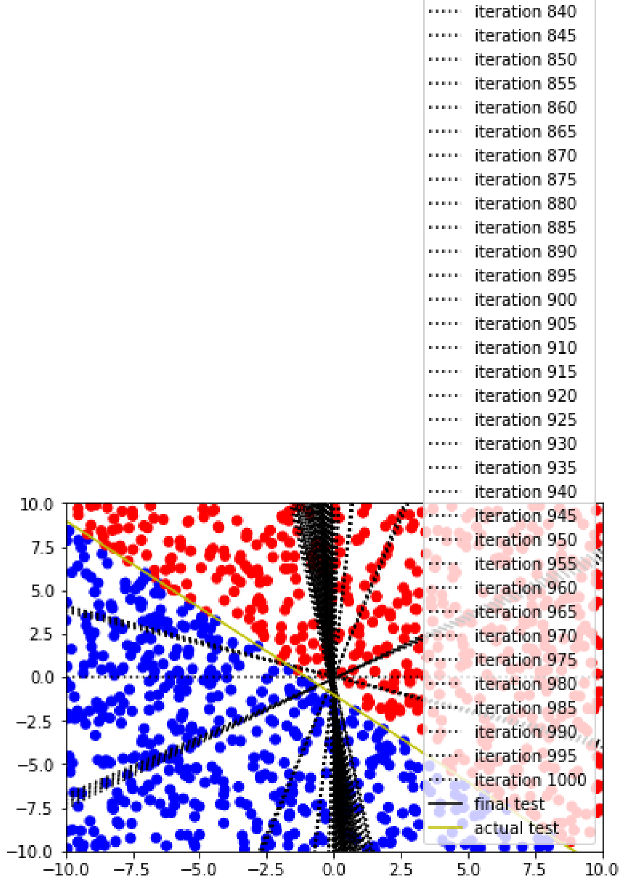
```
# Run perceptron learning algorithm for the test data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
#         if np.all(correct):
# Print the total number of iterations that let to the generation of final hypothesis g
#             print('Total number of iterations: ',n)
#             return w
        if n == 1000:
            error = np.absolute(error)
            print("Error on test data for n = 1: ",error[0]/1000)
            return w

        else:

            i = np.argmax(~correct)  # indice of first misclassified point
            s = w * X_train[:,i]
            d = y_train[i] * s
            d = d[0] + d[1] + d[2]
            if d <=1:
                w = w + 1 * (y_train[i] - s) * X_train[:, i]
                error = (y_train[i] - s)

        n = n + 1
x_axis = X_test[1, :]
y_axis = X_test[2, :]
lables = ['r' if y > 0 else 'b' for y in y_test]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)
# Run Perceptron learning algorith for test data set

w_final = run_perceptron(w_init, X_test, y_test, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final test')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual test')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
```

```
plt.show()
```

```
Error on test data for n = 1:  0.0
```



····· iteration 405
····· iteration 410
····· iteration 415
····· iteration 420
····· iteration 425
····· iteration 430
····· iteration 435
····· iteration 440
····· iteration 445
····· iteration 450
····· iteration 455
····· iteration 460
····· iteration 465
····· iteration 470
····· iteration 475
····· iteration 480
····· iteration 485
····· iteration 490
····· iteration 495
····· iteration 500
····· iteration 505
····· iteration 510
····· iteration 515
····· iteration 520
····· iteration 525
····· iteration 530
····· iteration 535
····· iteration 540
····· iteration 545
····· iteration 550
····· iteration 555
····· iteration 560
····· iteration 565
····· iteration 570
····· iteration 575
····· iteration 580
····· iteration 585
····· iteration 590
····· iteration 595
····· iteration 600
····· iteration 605
····· iteration 610
····· iteration 615
····· iteration 620
····· iteration 625
····· iteration 630
····· iteration 635
····· iteration 640
····· iteration 645
····· iteration 650
····· iteration 655
····· iteration 660
····· iteration 665
····· iteration 670
····· iteration 675
····· iteration 680
····· iteration 685
····· iteration 690
····· iteration 695
····· iteration 700
····· iteration 705
····· iteration 710
····· iteration 715
····· iteration 720
····· iteration 725
····· iteration 730
····· iteration 735
····· iteration 740
····· iteration 745
····· iteration 750
····· iteration 755
····· iteration 760
····· iteration 765
····· iteration 770
····· iteration 775
····· iteration 780
····· iteration 785
····· iteration 790
····· iteration 795
····· iteration 800
····· iteration 805
····· iteration 810
····· iteration 815
····· iteration 820
····· iteration 825
····· iteration 830
····· iteration 835



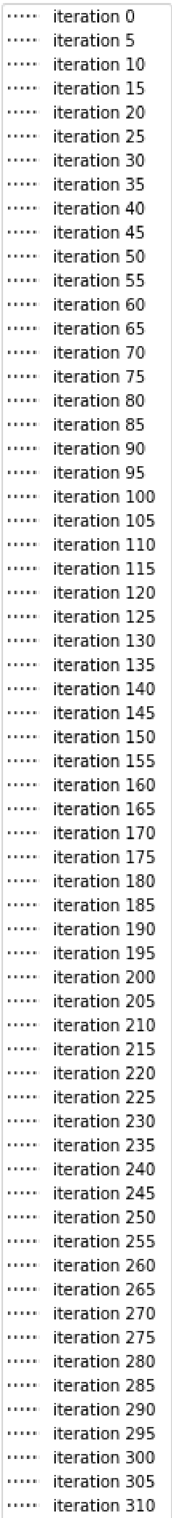
In the above plot the test data is plotted for n =1 and error is measured. The error rate in this case is 0

(c) The problem is now run for n = 0.01

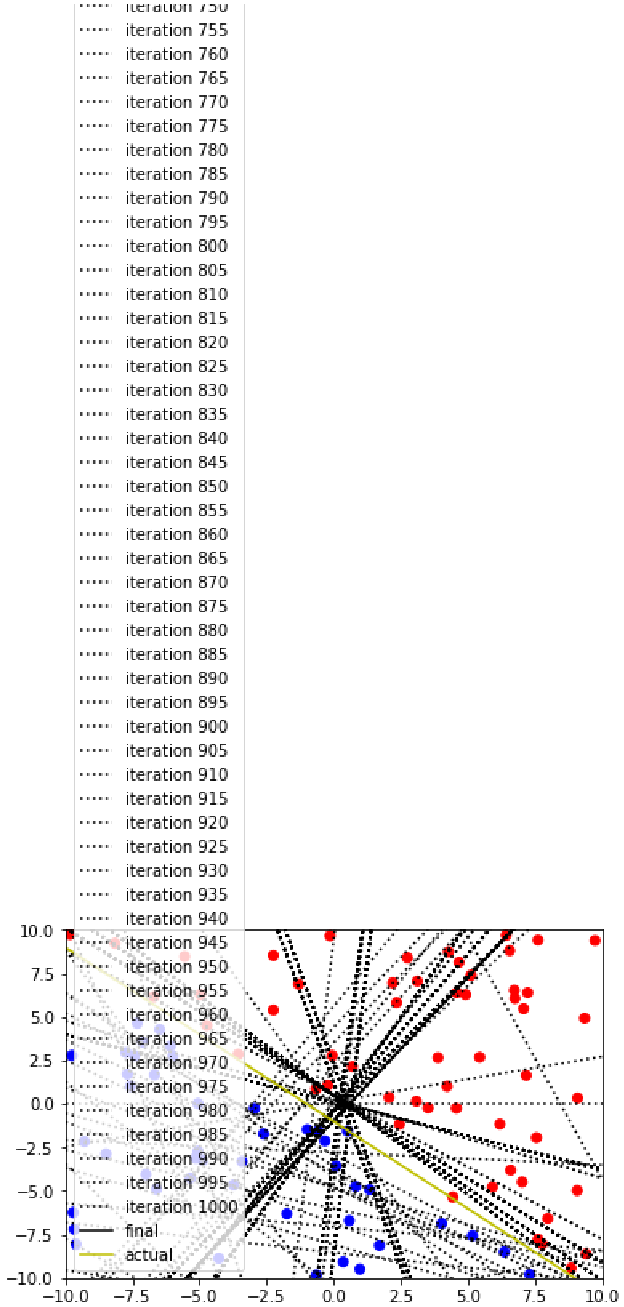
In [122]:

```
y_train = evaluate_h(w_act, X_train)
y_test = evaluate_h(w_act, X_test)
x_axis = X_train[1, :]
y_axis = X_train[2, :]
lables = ['r' if y > 0 else 'b' for y in y_train]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])


# Run perceptron learning algorithm for the training data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
# Calculate error after 1000 updates
        if n == 1000:
            return w

        else:
# calculate weight considering the closeness between s and y_train
            i = np.argmax(~correct)   # indice of first misclassified point
            s = w * X_train[:,i]
            d = y_train[i] * s
            d = d[0] + d[1] + d[2]
            if d <=1:
                w = w + 0.01 * (y_train[i] - s) * X_train[:, i]
                error = (y_train[i] - s)

        n = n + 1
```

```python
def plot_hypothesis(x_axis, w, *plot_args, **plot_kwargs):
    m = -w[1]/w[2] if w[2] != 0 else 0
    b = -w[0]/w[2] if w[2] != 0 else 0
    y_axis = m*x_axis + b
    plt.plot(x_axis, y_axis, *plot_args, **plot_kwargs)

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)

w_final = run_perceptron(w_init, X_train, y_train, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
plt.show()
```



····· iteration 315
····· iteration 320
····· iteration 325
····· iteration 330
····· iteration 335
····· iteration 340
····· iteration 345
····· iteration 350
····· iteration 355
····· iteration 360
····· iteration 365
····· iteration 370
····· iteration 375
····· iteration 380
····· iteration 385
····· iteration 390
····· iteration 395
····· iteration 400
····· iteration 405
····· iteration 410
····· iteration 415
····· iteration 420
····· iteration 425
····· iteration 430
····· iteration 435
····· iteration 440
····· iteration 445
····· iteration 450
····· iteration 455
····· iteration 460
····· iteration 465
····· iteration 470
····· iteration 475
····· iteration 480
····· iteration 485
····· iteration 490
····· iteration 495
····· iteration 500
····· iteration 505
····· iteration 510
····· iteration 515
····· iteration 520
····· iteration 525
····· iteration 530
····· iteration 535
····· iteration 540
····· iteration 545
····· iteration 550
····· iteration 555
····· iteration 560
····· iteration 565
····· iteration 570
····· iteration 575
····· iteration 580
····· iteration 585
····· iteration 590
····· iteration 595
····· iteration 600
····· iteration 605
····· iteration 610
····· iteration 615
····· iteration 620
····· iteration 625
····· iteration 630
····· iteration 635
····· iteration 640
····· iteration 645
····· iteration 650
····· iteration 655
····· iteration 660
····· iteration 665
····· iteration 670
····· iteration 675
····· iteration 680
····· iteration 685
····· iteration 690
····· iteration 695
····· iteration 700
····· iteration 705
····· iteration 710
····· iteration 715
····· iteration 720
····· iteration 725
····· iteration 730
····· iteration 735
····· iteration 740
····· iteration 745
iteration 750



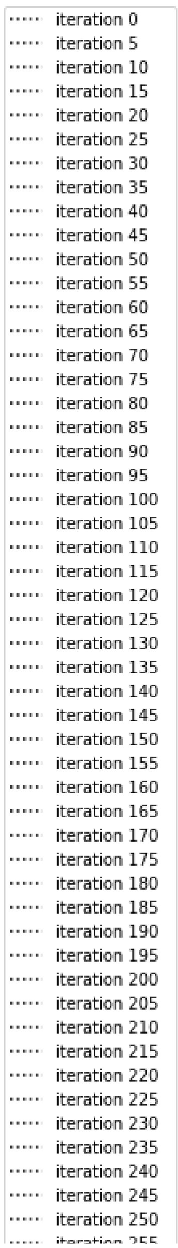
The plot above takes into account n =1 for training data

In [132]:

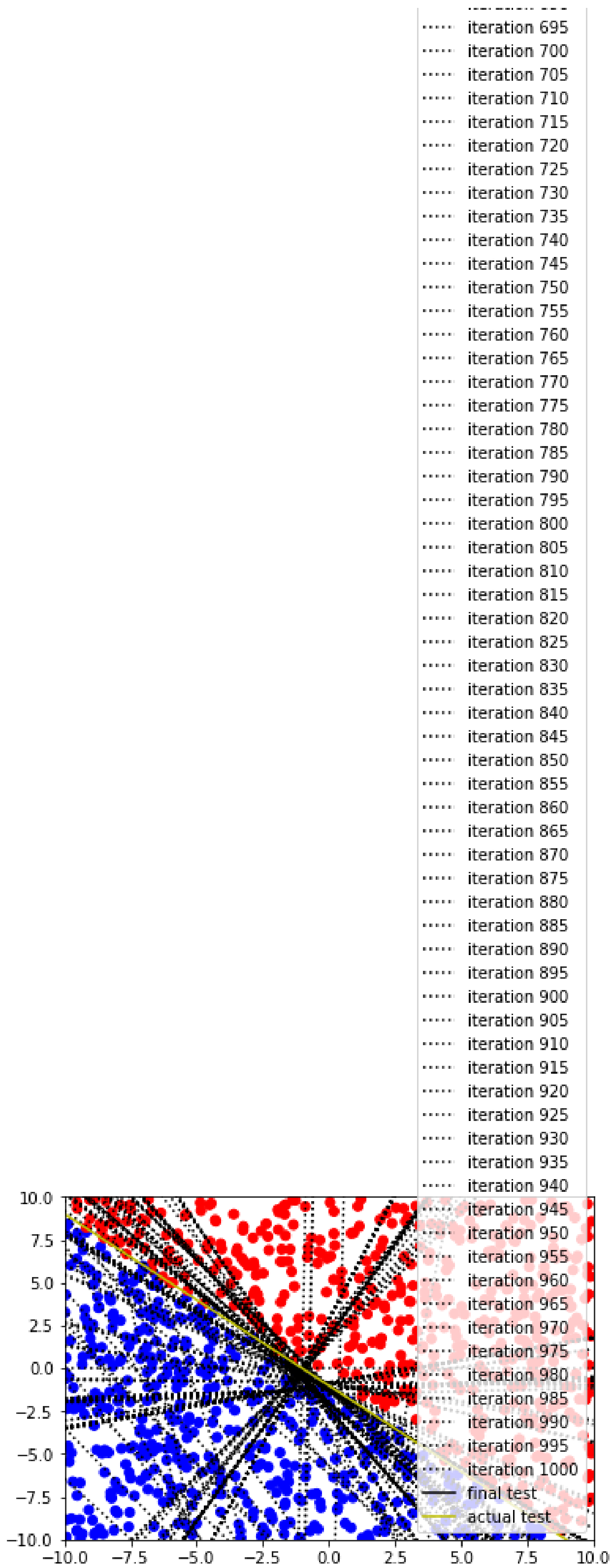
```
# Run perceptron learning algorithm for the test data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
#         if np.all(correct):
# Print the total number of iterations that let to the generation of final hypothesis g
#             print('Total number of iterations: ',n)
#             return w
        if n == 1000:
            error = np.absolute(error)
            print("Error on test data for n = 0.01: ",error[0]/1000)
            return w

        else:

            i = np.argmax(~correct)   # indice of first misclassified point
            s = w * X_train[:,i]
```

```
            d = y_train[i] * s
            d = d[0] + d[1] + d[2]
            if d <=1:
                w = w + 0.01 * (y_train[i] - s) * X_train[:, i]
                error = (y_train[i] - s)

        n = n + 1
x_axis = X_test[1, :]
y_axis = X_test[2, :]
lables = ['r' if y > 0 else 'b' for y in y_test]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)
# Run Perceptron learning algorith for test data set

w_final = run_perceptron(w_init, X_test, y_test, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final test')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual test')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
plt.show()
```

```
Error on test data for n = 0.01:  0.0009270516925850184
```



····· iteration 255
····· iteration 260
····· iteration 265
····· iteration 270
····· iteration 275
····· iteration 280
····· iteration 285
····· iteration 290
····· iteration 295
····· iteration 300
····· iteration 305
····· iteration 310
····· iteration 315
····· iteration 320
····· iteration 325
····· iteration 330
····· iteration 335
····· iteration 340
····· iteration 345
····· iteration 350
····· iteration 355
····· iteration 360
····· iteration 365
····· iteration 370
····· iteration 375
····· iteration 380
····· iteration 385
····· iteration 390
····· iteration 395
····· iteration 400
····· iteration 405
····· iteration 410
····· iteration 415
····· iteration 420
····· iteration 425
····· iteration 430
····· iteration 435
····· iteration 440
····· iteration 445
····· iteration 450
····· iteration 455
····· iteration 460
····· iteration 465
····· iteration 470
····· iteration 475
····· iteration 480
····· iteration 485
····· iteration 490
····· iteration 495
····· iteration 500
····· iteration 505
····· iteration 510
····· iteration 515
····· iteration 520
····· iteration 525
····· iteration 530
····· iteration 535
····· iteration 540
····· iteration 545
····· iteration 550
····· iteration 555
····· iteration 560
····· iteration 565
····· iteration 570
····· iteration 575
····· iteration 580
····· iteration 585
····· iteration 590
····· iteration 595
····· iteration 600
····· iteration 605
····· iteration 610
····· iteration 615
····· iteration 620
····· iteration 625
····· iteration 630
····· iteration 635
····· iteration 640
····· iteration 645
····· iteration 650
····· iteration 655
····· iteration 660
····· iteration 665
····· iteration 670
····· iteration 675
····· iteration 680
····· iteration 685
····· iteration 690



In the above plot the test data is plotted for n =0.01 and error is measured. The error rate in this case is 0.00092

(d)The problem is now run for n = 0.0001

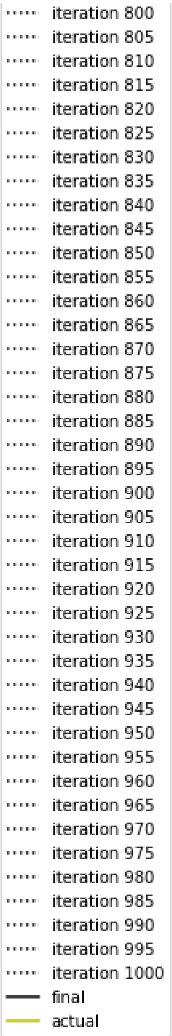
In [117]:

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# generate a linearly separated training data of size 100 and test data of size 1000 -
w_act = np.array([1, 1, 1])
w_init = np.array([3, -50, 0])
d = 2
num_sample = 100
```

```
data_range = 10
num_sample_test =1000
error = 0
# check if the hypothesis h from the hypothesis set H is following the weight and dimensionality c
ondition
def evaluate_h(w, X):

    '''
    Evaluate the hypothesis, h \in H, which is described by its set of weights,
    w, against 1 or more points in the input space (stored in a matrix).
    '''
    assert len(w.shape) == 1
    assert len(X.shape) == 2
    assert w.shape[0] == X.shape[0]
    return np.sign(w @ X)
X_train = np.vstack([
    np.ones(num_sample),
    np.random.uniform(
        -data_range,
        data_range,
        (d, num_sample),
    ),
])

X_test = np.vstack([
    np.ones(num_sample_test),
    np.random.uniform(
        -data_range,
        data_range,
        (d, num_sample_test),
    ),
])

y_train = evaluate_h(w_act, X_train)
y_test = evaluate_h(w_act, X_test)
x_axis = X_train[1, :]
y_axis = X_train[2, :]
lables = ['r' if y > 0 else 'b' for y in y_train]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])


# Run perceptron learning algorithm for the training data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
# Calculate error after 1000 updates
        if n == 1000:
            return w

        else:
# calculate weight considering the closeness between s and y_train
            i = np.argmax(~correct)  # indice of first misclassified point
            s = w * X_train[:,i]
            d = y_train[i] * s
            d = d[0] + d[1] + d[2]
            if d <=1:
                w = w + 0.0001 * (y_train[i] - s) * X_train[:, i]
                error = (y_train[i] - s)

        n = n + 1


def plot_hypothesis(x_axis, w, *plot_args, **plot_kwargs):
    m = -w[1]/w[2] if w[2] != 0 else 0
    b = -w[0]/w[2] if w[2] != 0 else 0
    y_axis = m*x_axis + b
    plt.plot(x_axis, y_axis, *plot_args, **plot_kwargs)

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
```

```
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)

w_final = run_perceptron(w_init, X_train, y_train, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
plt.show()
```

····· iteration 365
····· iteration 370
····· iteration 375
····· iteration 380
····· iteration 385
····· iteration 390
····· iteration 395
····· iteration 400
····· iteration 405
····· iteration 410
····· iteration 415
····· iteration 420
····· iteration 425
····· iteration 430
····· iteration 435
····· iteration 440
····· iteration 445
····· iteration 450
····· iteration 455
····· iteration 460
····· iteration 465
····· iteration 470
····· iteration 475
····· iteration 480
····· iteration 485
····· iteration 490
····· iteration 495
····· iteration 500
····· iteration 505
····· iteration 510
····· iteration 515
····· iteration 520
····· iteration 525
····· iteration 530
····· iteration 535
····· iteration 540
····· iteration 545
····· iteration 550
····· iteration 555
····· iteration 560
····· iteration 565
····· iteration 570
····· iteration 575
····· iteration 580
····· iteration 585
····· iteration 590
····· iteration 595
····· iteration 600
····· iteration 605
····· iteration 610
····· iteration 615
····· iteration 620
····· iteration 625
····· iteration 630
····· iteration 635
····· iteration 640
····· iteration 645
····· iteration 650
····· iteration 655
····· iteration 660
····· iteration 665
····· iteration 670
····· iteration 675
····· iteration 680
····· iteration 685
····· iteration 690
····· iteration 695
····· iteration 700
····· iteration 705
····· iteration 710
····· iteration 715
····· iteration 720
····· iteration 725
····· iteration 730
····· iteration 735
····· iteration 740
····· iteration 745
····· iteration 750
····· iteration 755
····· iteration 760
····· iteration 765
····· iteration 770
····· iteration 775
····· iteration 780
····· iteration 785
····· iteration 790
····· iteration 795



In [ ]:

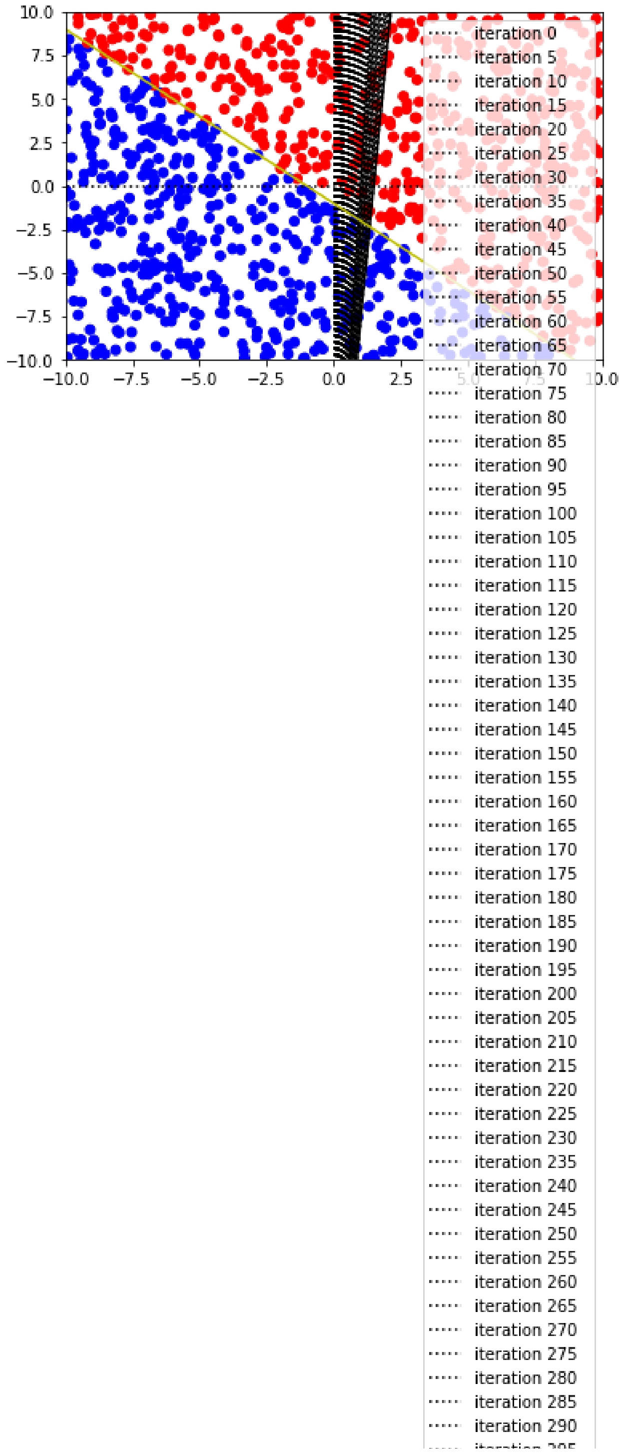
```
The plot above takes into account n =1 for training data
```

In [133]:

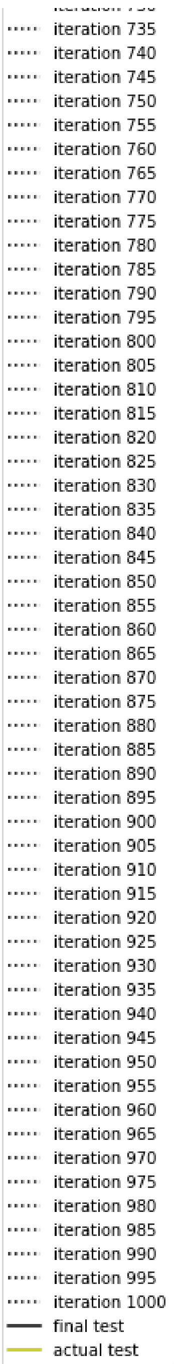
```
# Run perceptron learning algorithm for the test data set
def run_perceptron(w_init, X_train, y_train, iteration_callback=None):
    w = w_init.copy()
    n = 0
    while True:
        y = evaluate_h(w, X_train)
        if iteration_callback:
            iteration_callback(n, w)
        correct = y == y_train
#         if np.all(correct):
# Print the total number of iterations that let to the generation of final hypothesis g
#             print('Total number of iterations: ',n)
#             return w
        if n == 1000:
            error = np.absolute(error)
            print("Error on test data for n = 0.0001: ",error[0]/1000)
            return w

        else:

            i = np.argmax(~correct)  # indice of first misclassified point
            s = w * X_train[:,i]
            d = y_train[i] * s
            d = d[0] + d[1] + d[2]
            if d <=1:
                w = w + 0.0001 * (y_train[i] - s) * X_train[:, i]
                error = (y_train[i] - s)

        n = n + 1
x_axis = X_test[1, :]
```

```
x_axis = X_test[1, :]
y_axis = X_test[2, :]
lables = ['r' if y > 0 else 'b' for y in y_test]
plt.scatter(x_axis, y_axis, c=lables)
x_hypothesis = np.array([-data_range, data_range])

# Plot the hypothesis for all the iterations
def plot_iteration(n, w):
    if n % 5 == 0:
        label1 = 'iteration {}'.format(n)
        plot_hypothesis(x_hypothesis, w, 'k:', label=label1)
# Run Perceptron learning algorith for test data set

w_final = run_perceptron(w_init, X_test, y_test, plot_iteration)
plot_hypothesis(x_hypothesis, w_final, 'k', label='final test')
plot_hypothesis(x_hypothesis, w_act, 'y', label='actual test')
plt.legend()
plt.xlim(-data_range, data_range)
plt.ylim(-data_range, data_range)
plt.show()
```

```
Error on test data for n = 0.0001:  0.0036196935435885438
```



iteration 295
iteration 300
iteration 305
iteration 310
iteration 315
iteration 320
iteration 325
iteration 330
iteration 335
iteration 340
iteration 345
iteration 350
iteration 355
iteration 360
iteration 365
iteration 370
iteration 375
iteration 380
iteration 385
iteration 390
iteration 395
iteration 400
iteration 405
iteration 410
iteration 415
iteration 420
iteration 425
iteration 430
iteration 435
iteration 440
iteration 445
iteration 450
iteration 455
iteration 460
iteration 465
iteration 470
iteration 475
iteration 480
iteration 485
iteration 490
iteration 495
iteration 500
iteration 505
iteration 510
iteration 515
iteration 520
iteration 525
iteration 530
iteration 535
iteration 540
iteration 545
iteration 550
iteration 555
iteration 560
iteration 565
iteration 570
iteration 575
iteration 580
iteration 585
iteration 590
iteration 595
iteration 600
iteration 605
iteration 610
iteration 615
iteration 620
iteration 625
iteration 630
iteration 635
iteration 640
iteration 645
iteration 650
iteration 655
iteration 660
iteration 665
iteration 670
iteration 675
iteration 680
iteration 685
iteration 690
iteration 695
iteration 700
iteration 705
iteration 710
iteration 715
iteration 720
iteration 725
iteration 730



In the above plot the test data is plotted for n =0.01 and error is measured. The error rate in this case is 0.0036

Comparing all the 4 scenarios, we see that the error rate for n = 100 is 10.3 n = 1 is 0 n = 0.01 is 0.00092 n = 0.0001 is 0.0036