Activity 8 △ Item 1

a) $x^T w = 0$

$x^T = [x_1 \ x_2], w = \begin{bmatrix} 5 \\ -2 \end{bmatrix}$

$\rightarrow 5x_1 - 2x_2 = 0 \rightarrow \underbrace{x_2 = \frac{5}{2}x_1}$ (S)

(0,0) (2,5)

ii) a subspace in $\mathbb{R}^2$, why ??

~~dimension~~ ① origin $\in S$ ✓

② ~~if $\overset{(a)}{(0,0)}, \overset{(b)}{(2,5)} \in S$~~

~~$\rightarrow (a)+(b) = (2,5) \in S$~~

② if $(x_1, y_1), (x_2, y_2) \in S$

then $(x_1+x_2, y_1+y_2) \in S$ ✓

③ if $(x_1, y_1) \in S$

then $\alpha(x_1, y_1) \in S$

$\Rightarrow \underset{\in S}{\alpha(x_1, y_1)} = \underset{\in S}{\alpha(x_1, kx_1)}$

$k = \frac{5}{2}$

$\Rightarrow (x_1, y_1) = \cancel{(x_1, \ldots)} = \cancel{(x_1, \frac{5}{2} \ldots)}$

$= (x_1, kx_1) \in S$

$(x_2, y_2) = (x_2, kx_2) \in S$

then $(x_1+x_2, k(x_1+x_2)) \in S$

b) $x^T = [x_1 \ x_2 \ 1], w = \begin{bmatrix} 5 \\ -2 \\ 1 \end{bmatrix}$

$\rightarrow 5x_1 - 2x_2 + 1 = 0$

$\rightarrow \underbrace{x_2 = \frac{5}{2}x_1 + \frac{1}{2}}$ (S) $(0, \frac{1}{2})$ $(1, 3)$

ii) ① (S) does not go through the origin

c) $x^T = [x_1^2 \ x_2 \ 1], w = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$

$\rightarrow x_1^2 - 2x_2 + 1 = 0 \rightarrow \underbrace{\frac{x_1^2+1}{2} = x_2}$ (S)

$(0, \frac{1}{2})$ (1,1) (-1,1)

ii) ① (S) does not go through the origin

orthonormal basis : $\frac{1}{\sqrt{29}}\begin{bmatrix} 2 \\ 5 \end{bmatrix}$

△ Item 4 : a plane

△ Item 5 :

$x^T w = 0$

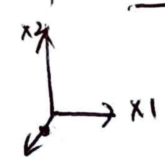
$\rightarrow w_1x_1 + w_2x_2 + w_3x_3 + w_4 = 0$

includes point$_a$ (0,0,1)

~~$\rightarrow w_3 = w_4$~~

$x_1, x_2 \in \mathbb{R}, \ x_3 = 1 \quad 1 = \frac{-w_4}{w_3} \rightarrow w = \alpha \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix} \quad \alpha[0 \ 0 \ 1 \ -1]$

$x_a^T \cdot w = 0 \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \alpha [0 \ 0 \ 1 \ -1] = 0$



if decision boundary parallel to x_1-x_2 plane

$\rightarrow$ orthogonal to $\begin{cases} x_1\text{-}x_3 \text{ plane} \\ x_2\text{-}x_3 \text{ plane} \end{cases}$

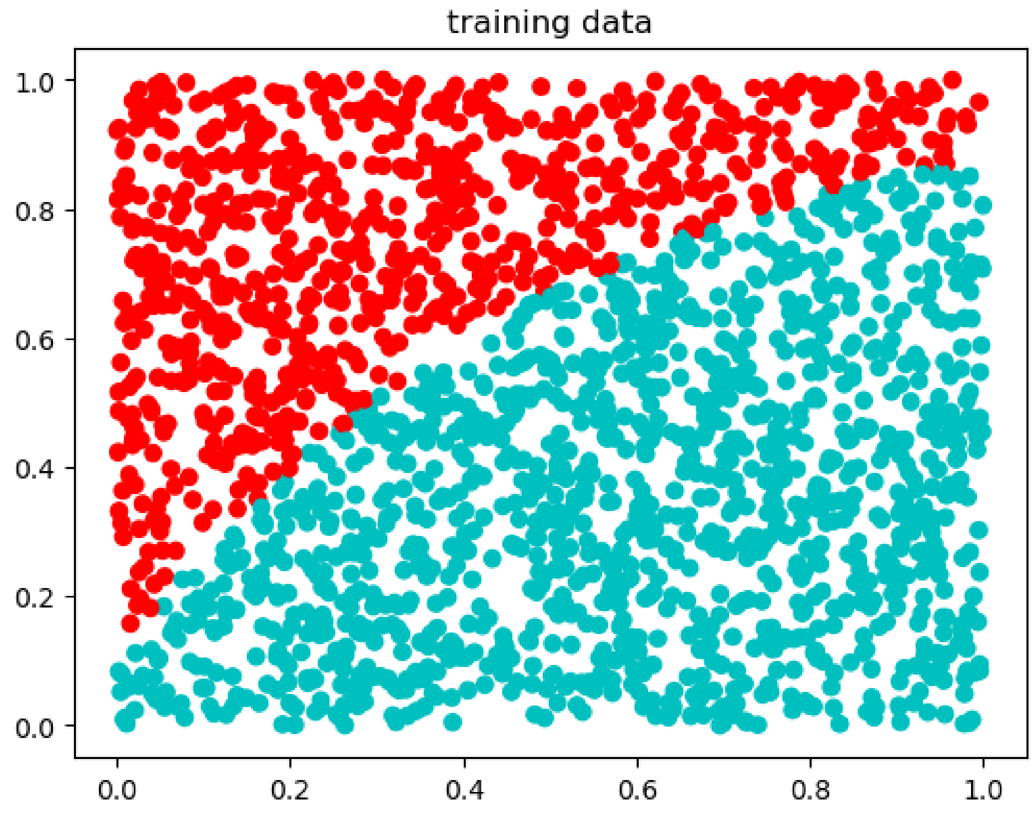
linear-classifier-item2

February 16, 2024

1 2a)

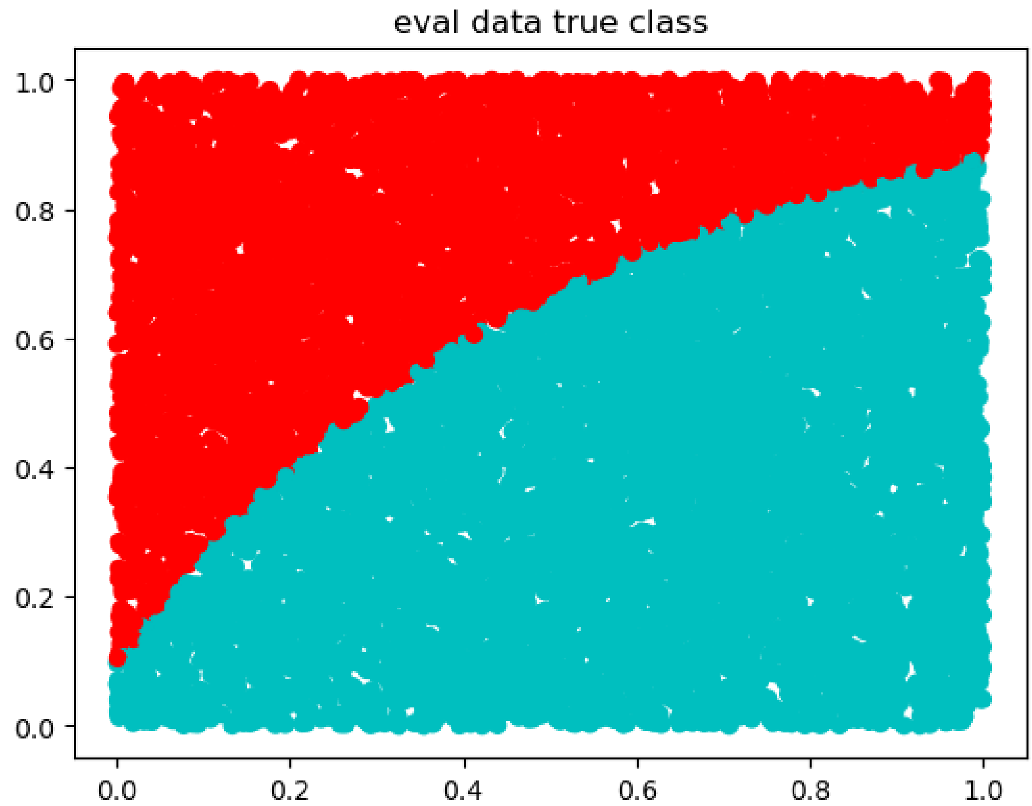
[10]:
```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

in_data = loadmat('classifier_data.mat')
print([key for key in in_data]) # -- use this line to see the keys in the␣
↪dictionary data structure

x_train = in_data['x_train']
x_eval = in_data['x_eval']
y_train = in_data['y_train']
y_eval = in_data['y_eval']

n_eval = np.size(y_eval)
n_train = np.size(y_train)
# print(n_eval, n_train)

plt.scatter(x_train[:,0],x_train[:,1], color=['c' if i==-1 else 'r' for i in␣
↪y_train[:,0]])
plt.title('training data')
plt.show()
```

```
['__header__', '__version__', '__globals__', 'x_eval', 'x_train', 'y_eval',
'y_train']
```

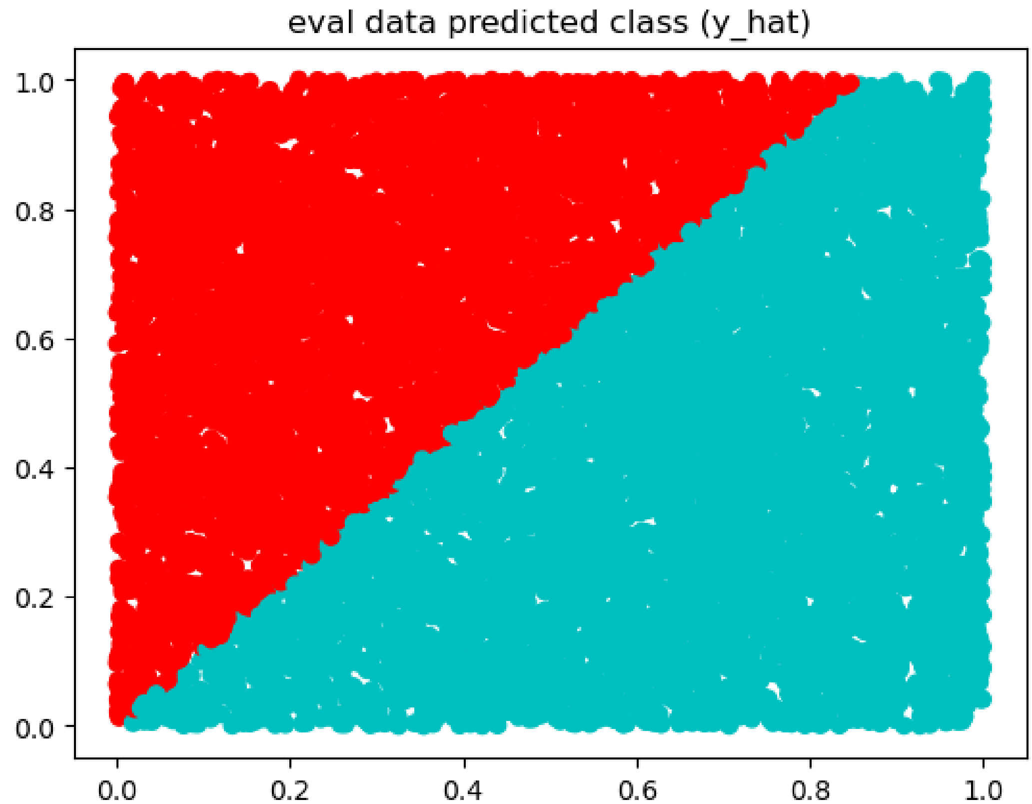


[11]:

```
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==-1 else 'r' for i in␣
  ↪y_eval[:,0]])
plt.title('eval data true class')
plt.show()
```

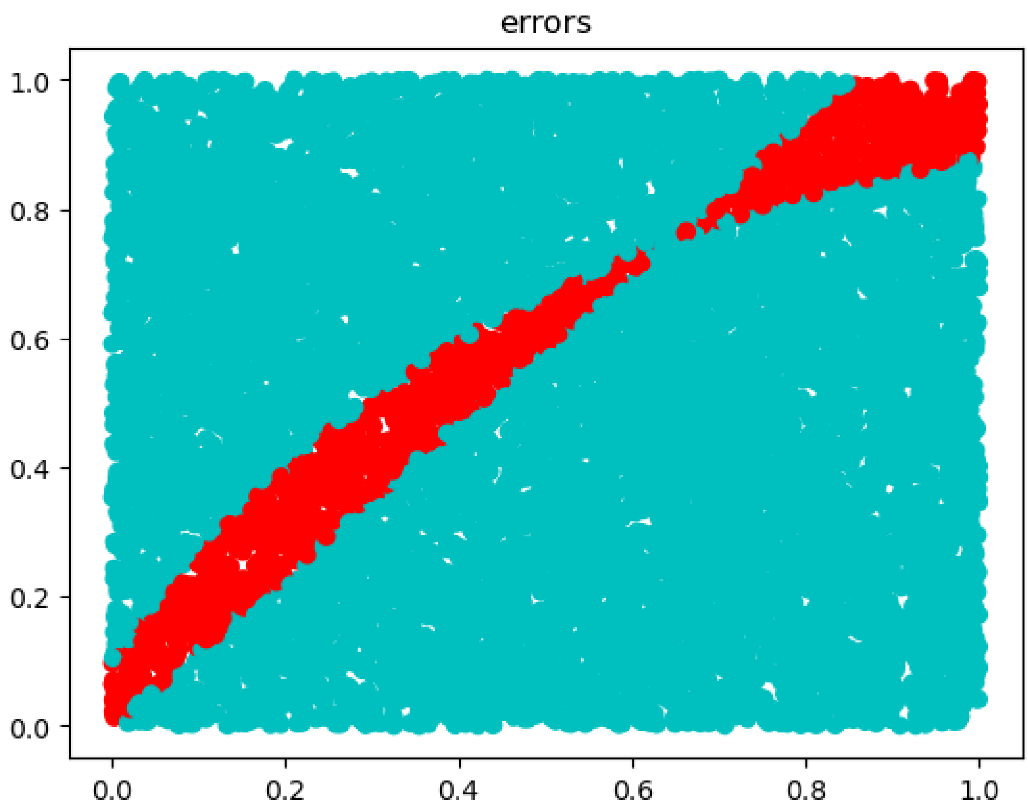


[12]:
```
## Classifier 1

# least squares as a loss function
# w = (X^T X)^(-1)X^T y
w_opt = np.linalg.inv(x_train.transpose()@x_train)@x_train.transpose()@y_train
y_hat = np.sign(x_eval@w_opt)

plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==-1 else 'r' for i in␣
 ↪y_hat[:,0]])
plt.title('eval data predicted class (y_hat)')
plt.show()
# Briefly comment on the foot of the classifier to the decision boundary␣
 ↪apparent in the evaluation data.
# --> The decision boundary appears as a linear line passing through the origin.
```



[20]:

```
error_vec = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat, y_eval))]
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==0 else 'r' for i in␣
 ↪error_vec])
plt.title('errors')
plt.show()

print('Errors: '+ str(sum(error_vec))) # sum(): because when the prediction is␣
 ↪an error, we get 1. Thus, sum all 1 can get the total error.
print('Error Rate:' + str(100*sum(error_vec)/ n_eval) + '%')
# Also identify the percent error based on the ratio of misclassified␣
 ↪evaluation data points to the total number of evaluation data points.
# --> 11.02%
```

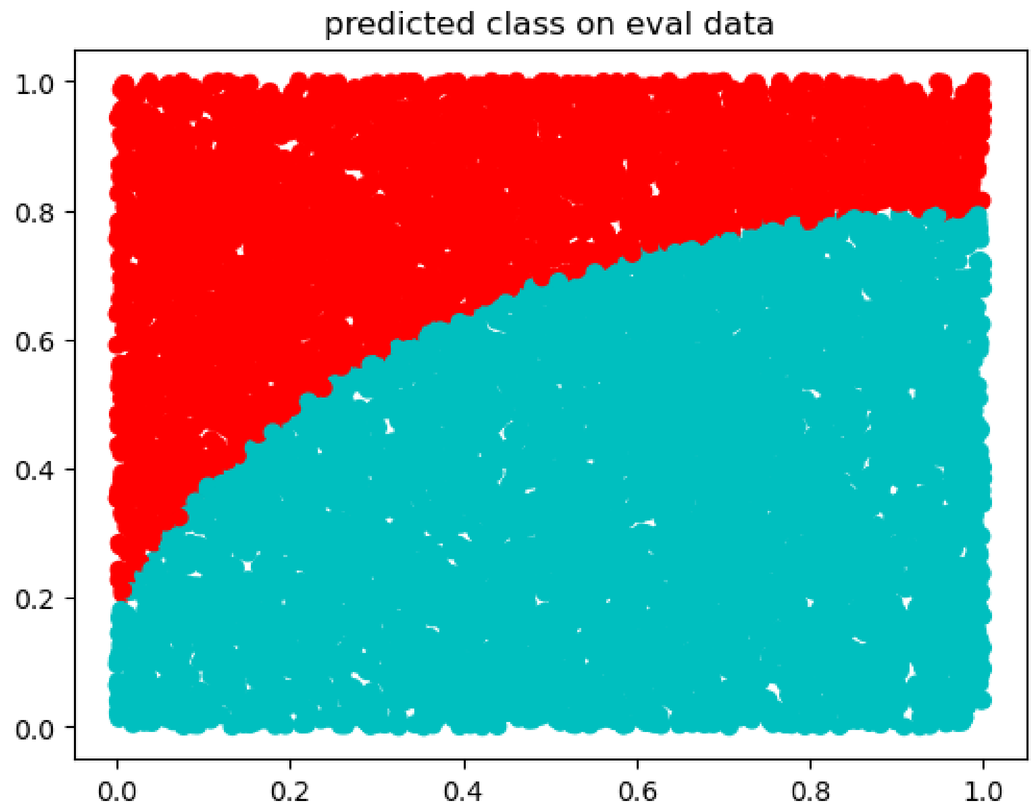



```
Errors: 1102
Percent Error:11.02%
```

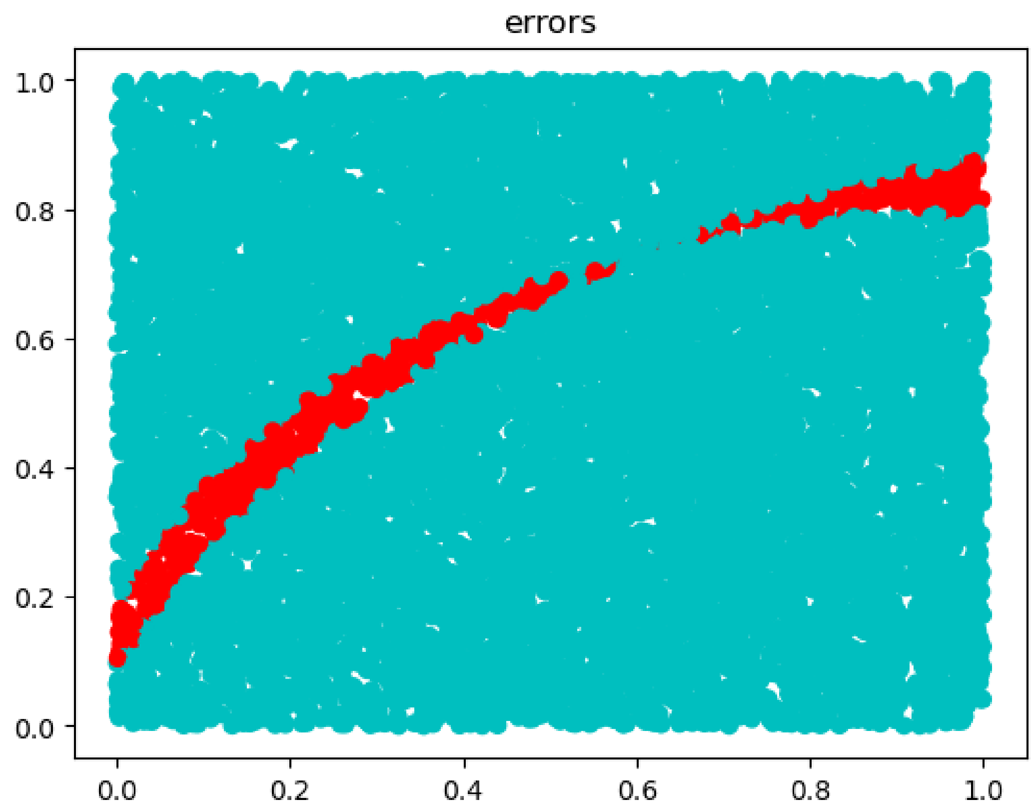
2 2b)

[14]:
```
## Classifier 2
# Stack arrays in sequence horizontally (column wise).
# Return a new array of given shape and type, filled with ones.
x_train_2 = np.hstack((x_train**2, x_train, np.ones((n_train,1)) ))
x_eval_2 = np.hstack((x_eval**2, x_eval, np.ones((n_eval,1)) ))

# w = (X^T X)^(-1)X^T y
w_opt_2 = np.linalg.inv(x_train_2.transpose()@x_train_2)@x_train_2.
 ↪transpose()@y_train
y_hat_2 = np.sign(x_eval_2@w_opt_2)

plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==-1 else 'r' for i in␣
 ↪y_hat_2[:,0]])
plt.title('predicted class on eval data')
plt.show()
```

```
# Briefly comment on the fit of the classifier to the decision boundary
↪apparent in the evaluation data.
# --> The decision boundary resembles a parabola, not intersecting the origin.
```



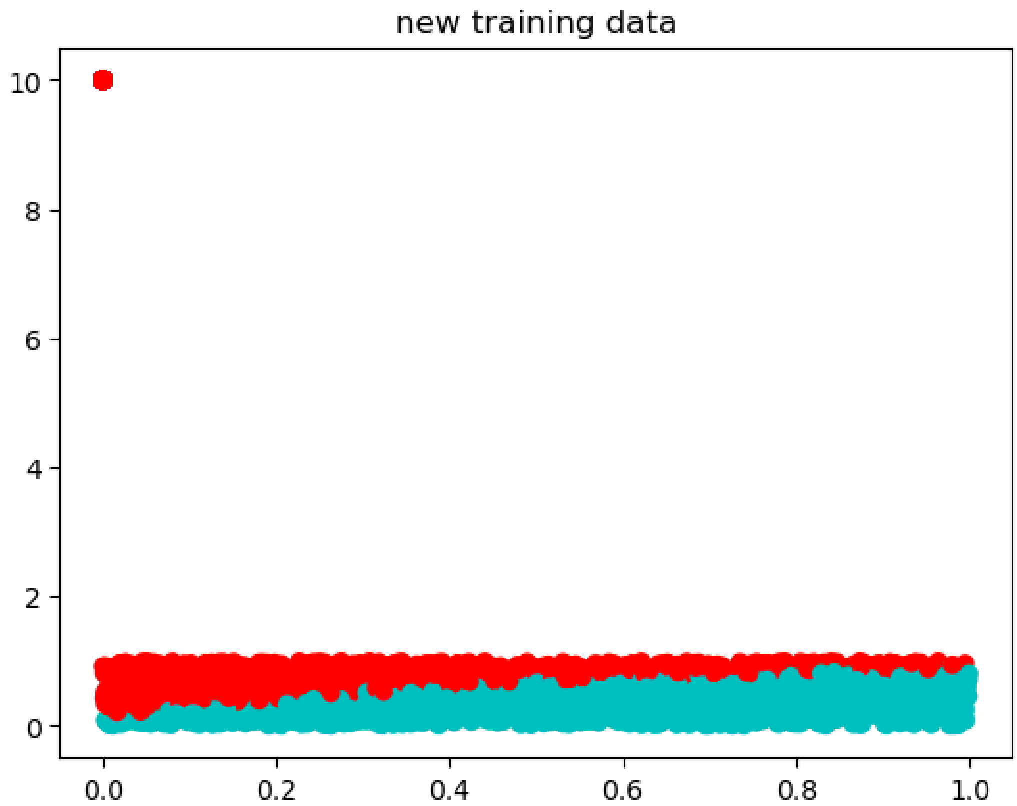
[22]:
```
error_vec_2 = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat_2, y_eval))]
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==0 else 'r' for i in
↪error_vec_2])
plt.title('errors')
plt.show()

print('Error: '+ str(sum(error_vec_2)))
print('Error Rate:' + str(100*sum(error_vec_2)/ n_eval) + '%')
# Also identify the percent error based on the ratio of misclassified
↪evaluation data points to the total number of evaluation data points.
# --> 5.42%
```



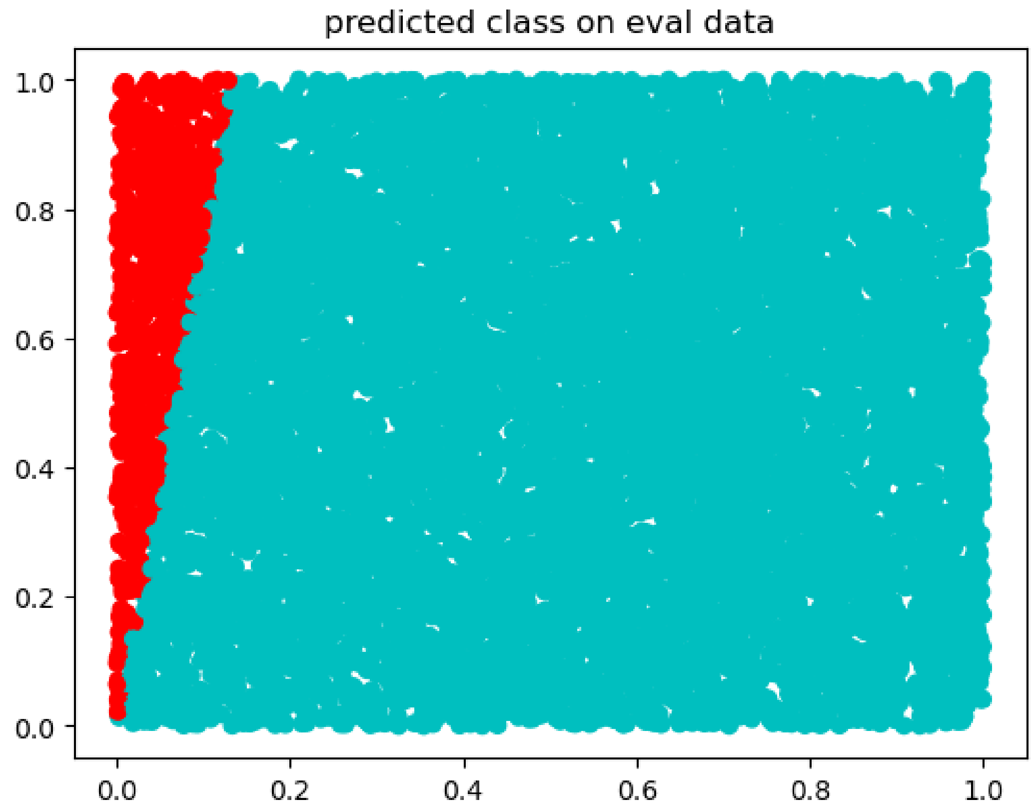

```
Error: 542
Percent Error:5.42%
```

3 2c)

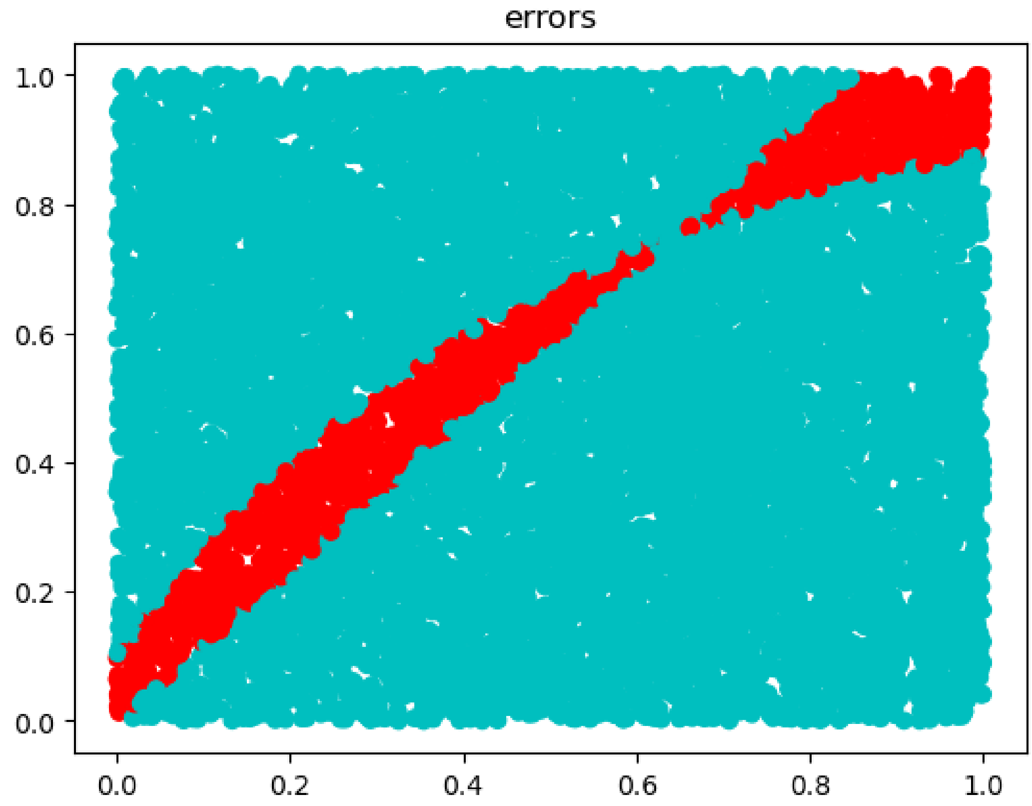
[24]:
```
## create new, correctly labeled points
n_new = 1000 #number of new datapoints
# x_train_new = np.hstack((np.zeros((n_new,1)), 3*np.ones((n_new,1))))
x_train_new = np.hstack((np.zeros((n_new,1)), 10*np.ones((n_new,1))))
y_train_new = np.ones((n_new,1))

## add these to the training data
# stack arrays vertically
x_train_outlier = np.vstack((x_train,x_train_new))
y_train_outlier = np.vstack((y_train,y_train_new))
plt.scatter(x_train_outlier[:,0],x_train_outlier[:,1], color=['c' if i==-1 else␣
 ↪'r' for i in y_train_outlier[:,0]])
plt.title('new training data')
plt.show()
```




```
[25]: #train with new data
# w = (X^T X)^(-1)X^T y
w_opt_outlier = np.linalg.inv(x_train_outlier.
 ↪transpose()@x_train_outlier)@x_train_outlier.transpose()@y_train_outlier
y_hat_outlier = np.sign(x_eval@w_opt_outlier)

plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==-1 else 'r' for i in␣
 ↪y_hat_outlier[:,0]])
plt.title('predicted class on eval data')
plt.show()
# What happens to the decision boundary when these new data points are included␣
 ↪in training? (vs (a))
# --> The decision boundary maintains a linear appearance, passing through the␣
 ↪origin,
# yet the slope increases, indicating a greater obliqueness in the line.
```



[26]:
```
error_vec_3 = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat_outlier,
    y_eval))]
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==0 else 'r' for i in
    error_vec])
plt.title('errors')
plt.show()

print('Errors: '+ str(sum(error_vec)))
print('Error Rate:' + str(100*sum(error_vec_3)/ n_eval) + '%')
# --> (0, 3): 21.34%
# --> (0, 10): 32.77%
```




```
Errors: 1102
Error Rate:32.77%
```

[ ]:
```
# What happens to the error rate if you move the 1000 data points to x1 = 0, x2␣
↪= 10? Why does this happen?
# --> The error rate rises as the outliers contribute to an increase in the␣
↪slope of the decision boundary.
```

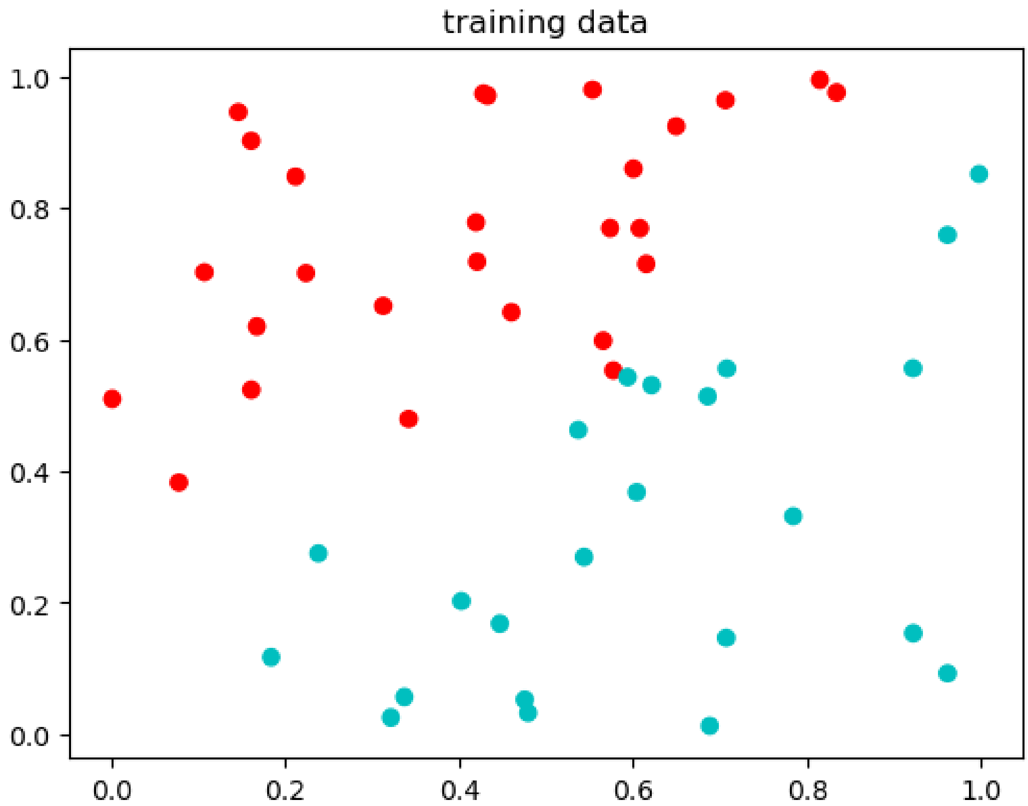
linear-classifier-item3

February 16, 2024

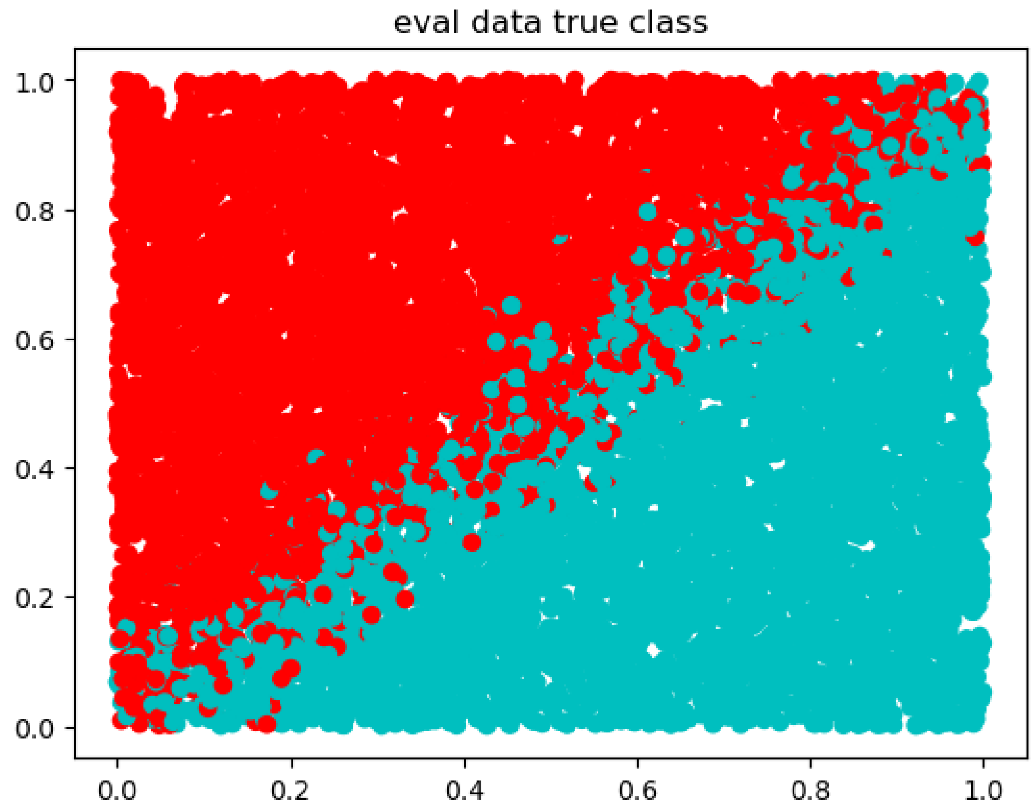
1 3c)

[3]:
```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

in_data = loadmat('overfitting_data.mat')
print([key for key in in_data]) # -- use this line to see the keys in the␣
    ↪dictionary data structure

x_train = in_data['x_train']
x_eval = in_data['x_eval']
y_train = in_data['y_train']
y_eval = in_data['y_eval']
# print(y_eval)

n_eval = np.size(y_eval)
n_train = np.size(y_train)
# print(n_eval, n_train)

plt.scatter(x_train[:,0],x_train[:,1], color=['c' if i==-1 else 'r' for i in␣
    ↪y_train[:,0]])
plt.title('training data')
plt.show()
```

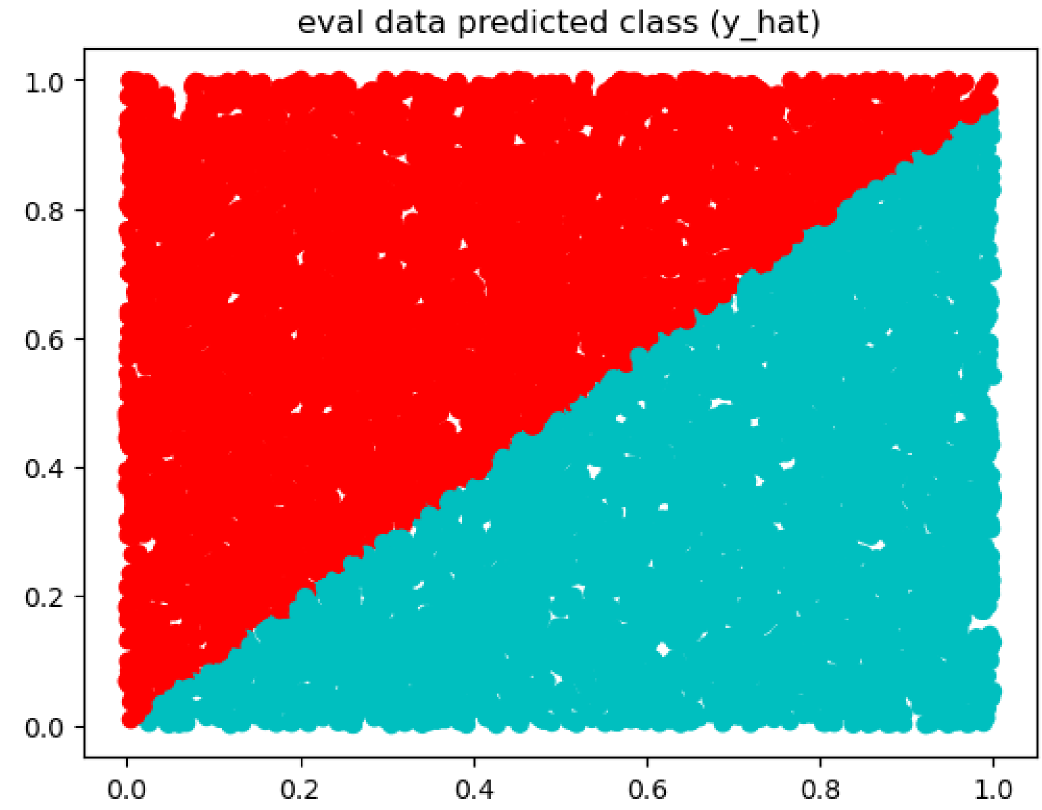
```
['__header__', '__version__', '__globals__', 'x_eval', 'x_train', 'y_eval',
'y_train']
```



[4]:
```
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==-1 else 'r' for i in␣
    ↪y_eval[:,0]])
plt.title('eval data true class')
plt.show()
```

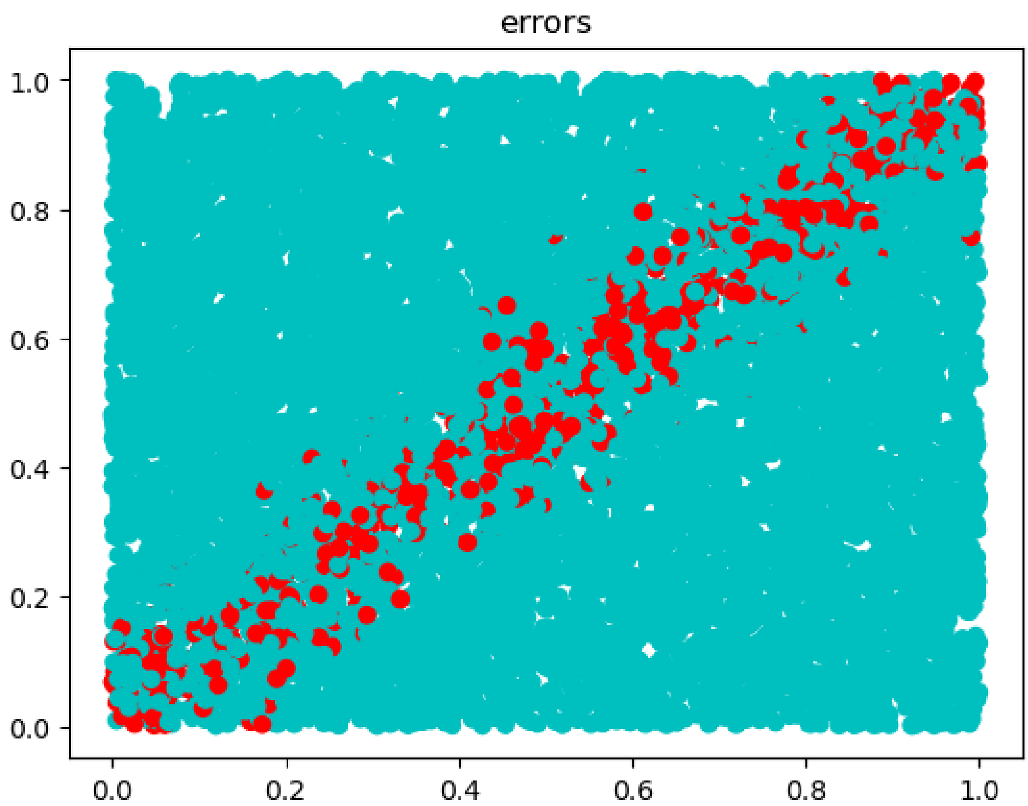


[5]:
```
## Classifier 1

# least squares as a loss function
# w = (X^T X)^(-1)X^T y
w_opt = np.linalg.inv(x_train.transpose()@x_train)@x_train.transpose()@y_train
y_hat = np.sign(x_eval@w_opt)

plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==-1 else 'r' for i in␣
 ↪y_hat[:,0]])
plt.title('eval data predicted class (y_hat)')
plt.show()
```



[6]:

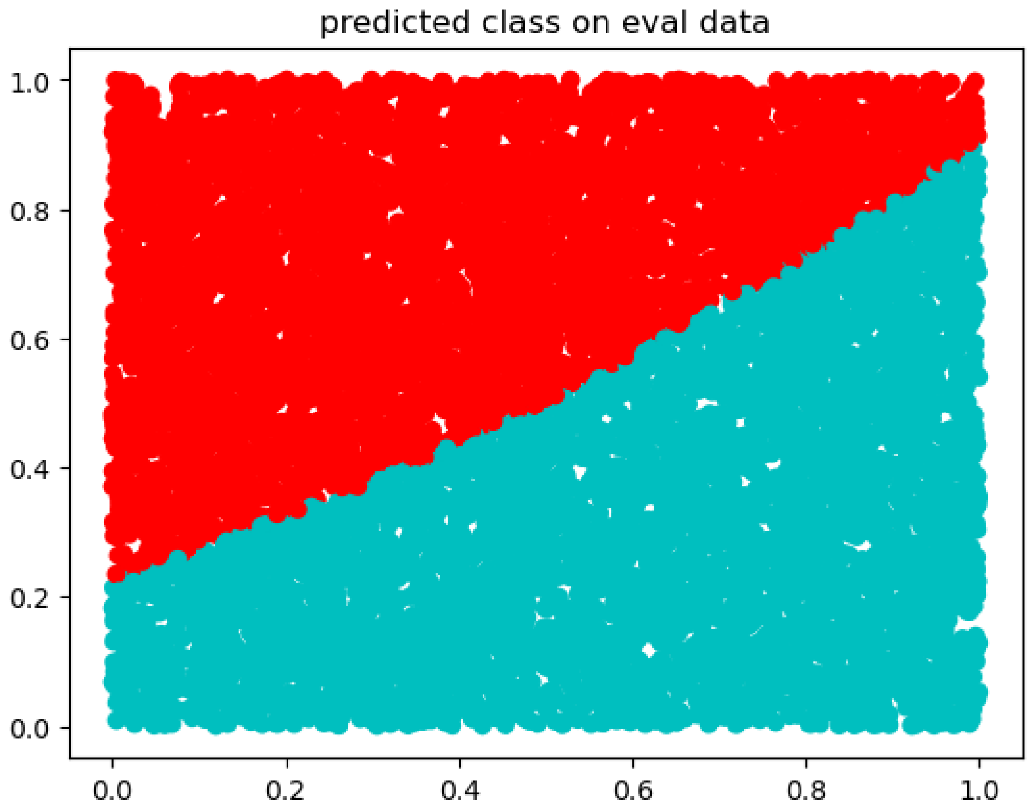
```
error_vec = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat, y_eval))]
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==0 else 'r' for i in␣
 ↪error_vec])
plt.title('errors')
plt.show()

print('Errors: '+ str(sum(error_vec))) # sum(): because when the prediction is␣
 ↪an error, we get 1. Thus, sum all 1 can get the total error.
# print('Error Rate:' + str(100*sum(error_vec)/ n_eval) + '%')
# How many errors are there?
# --> 759.
```



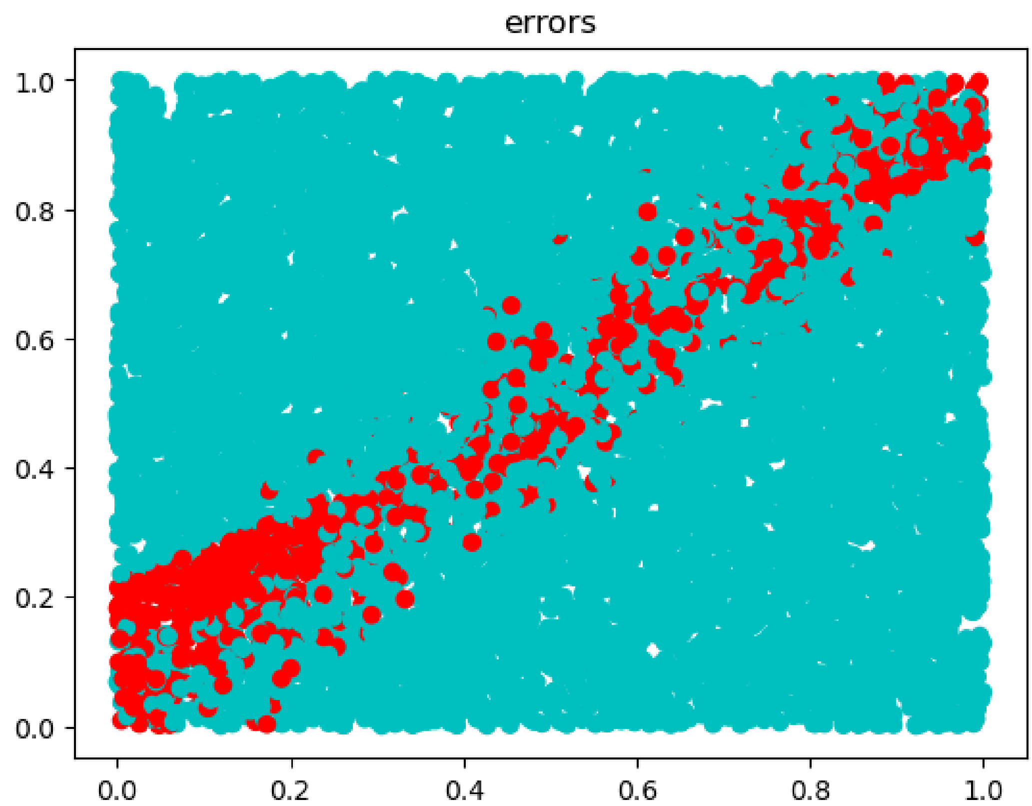

```
Errors: 759
```

2 3d)

[7]:
```
## Classifier 2
# Stack arrays in sequence horizontally (column wise).
# Return a new array of given shape and type, filled with ones.
x_train_2 = np.hstack((x_train**2, x_train, np.ones((n_train,1)) ))
x_eval_2 = np.hstack((x_eval**2, x_eval, np.ones((n_eval,1)) ))

# w = (X^T X)^(-1)X^T y
w_opt_2 = np.linalg.inv(x_train_2.transpose()@x_train_2)@x_train_2.
 ↪transpose()@y_train
y_hat_2 = np.sign(x_eval_2@w_opt_2)

plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==-1 else 'r' for i in␣
 ↪y_hat_2[:,0]])
plt.title('predicted class on eval data')
plt.show()
```



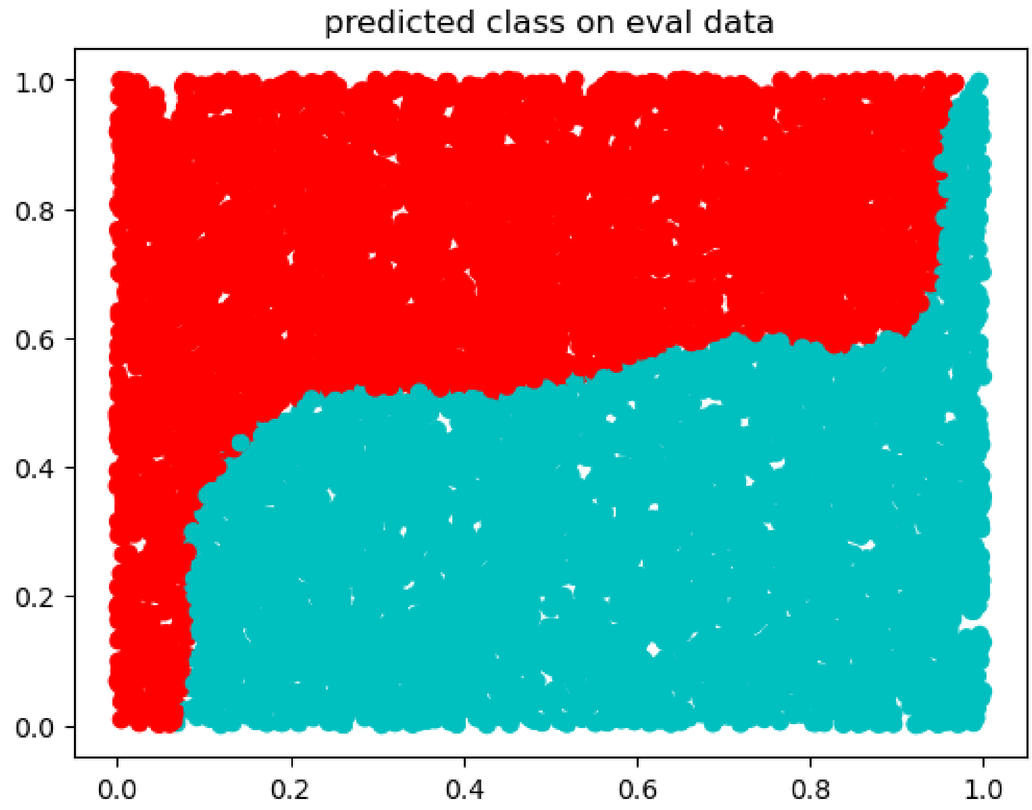
[9]:
```
error_vec_2 = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat_2, y_eval))]
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==0 else 'r' for i in␣
 ↪error_vec_2])
plt.title('errors')
plt.show()

print('Error: '+ str(sum(error_vec_2)))
# print('Error Rate:' + str(100*sum(error_vec_2)/ n_eval) + '%')
# How many errors are there?
# --> 1066
```



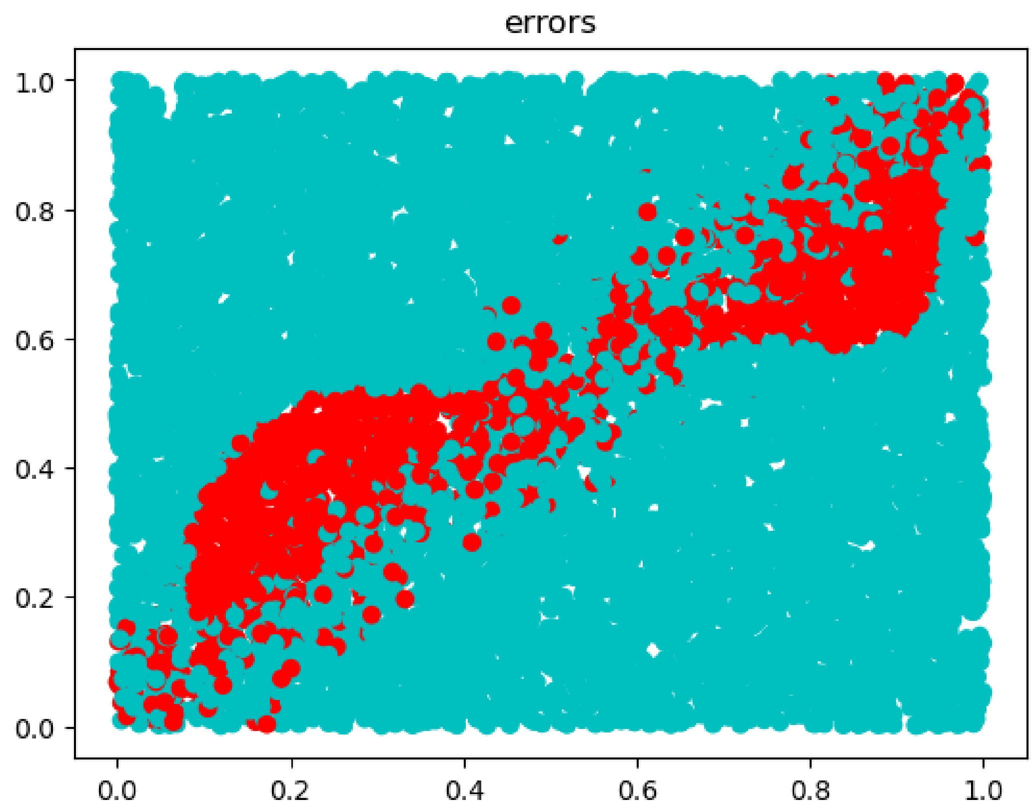

```
Error: 1066
```

3 3e)

[10]:
```
## Classifier 3
x_train_3 = np.hstack((x_train**6, x_train**5, x_train**4, x_train**3,␣
 ↪x_train**2, x_train, np.ones((n_train,1)) ))
x_eval_3 = np.hstack((x_eval**6, x_eval**5, x_eval**4, x_eval**3, x_eval**2,␣
 ↪x_eval, np.ones((n_eval,1)) ))

# w = (X^T X)^(-1)X^T y
w_opt_3 = np.linalg.inv(x_train_3.transpose()@x_train_3)@x_train_3.
 ↪transpose()@y_train
y_hat_3 = np.sign(x_eval_3@w_opt_3)

plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==-1 else 'r' for i in␣
 ↪y_hat_3[:,0]])
plt.title('predicted class on eval data')
plt.show()
```



[11]:
```
error_vec_3 = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat_3, y_eval))]
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==0 else 'r' for i in␣
 ↪error_vec_3])
plt.title('errors')
plt.show()

print('Error: '+ str(sum(error_vec_3)))
# print('Error Rate:' + str(100*sum(error_vec_2)/ n_eval) + '%')
# How many errors are there?
# --> 1677
```




```
Error: 1677
```

4 f) Of the three classifiers, which one performs worse? Why?

Classifiers3, the last one, because of overfitting.

[ ]:

[ ]:

[ ]:

[ ]: