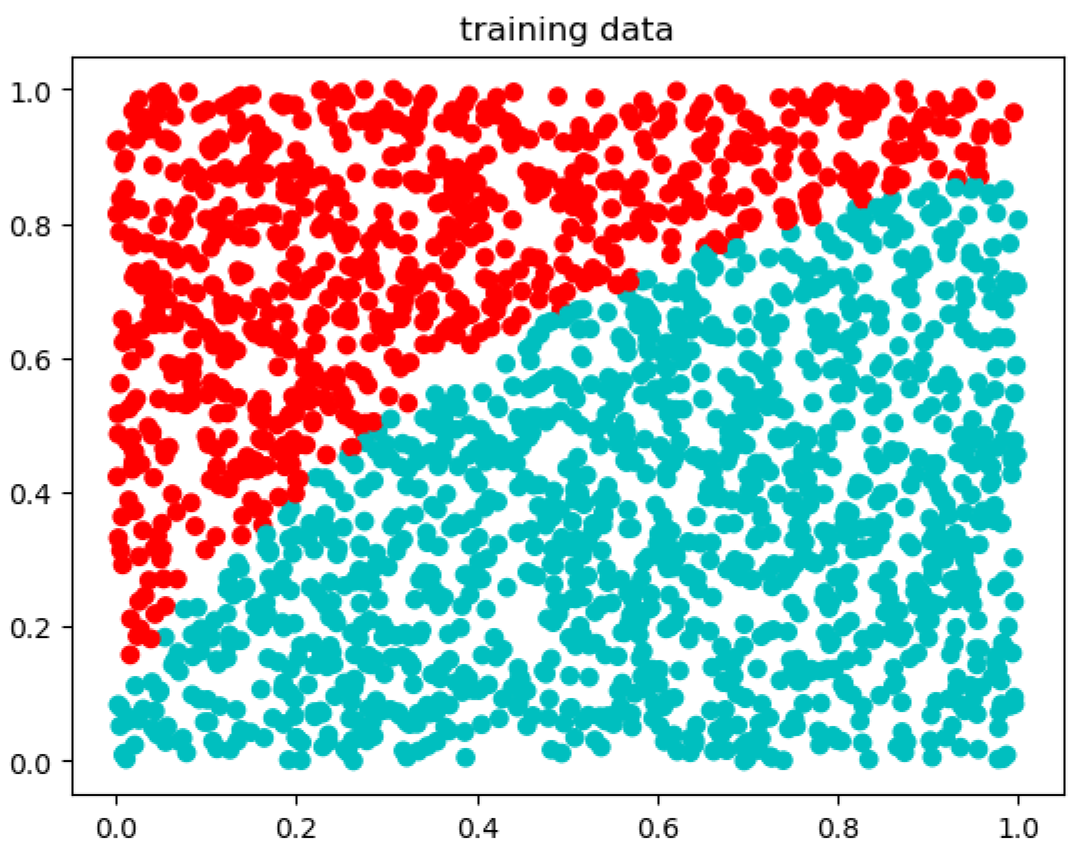
2a)

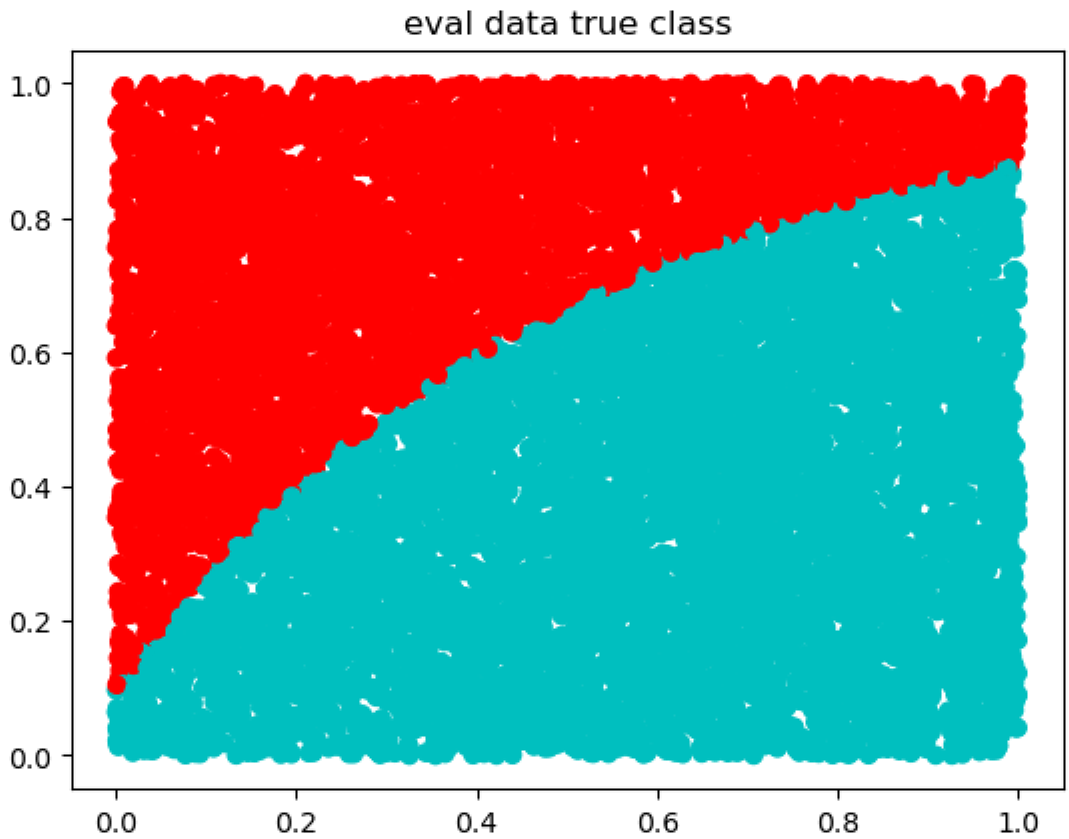
In [1]:

```
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 da

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)

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()
```

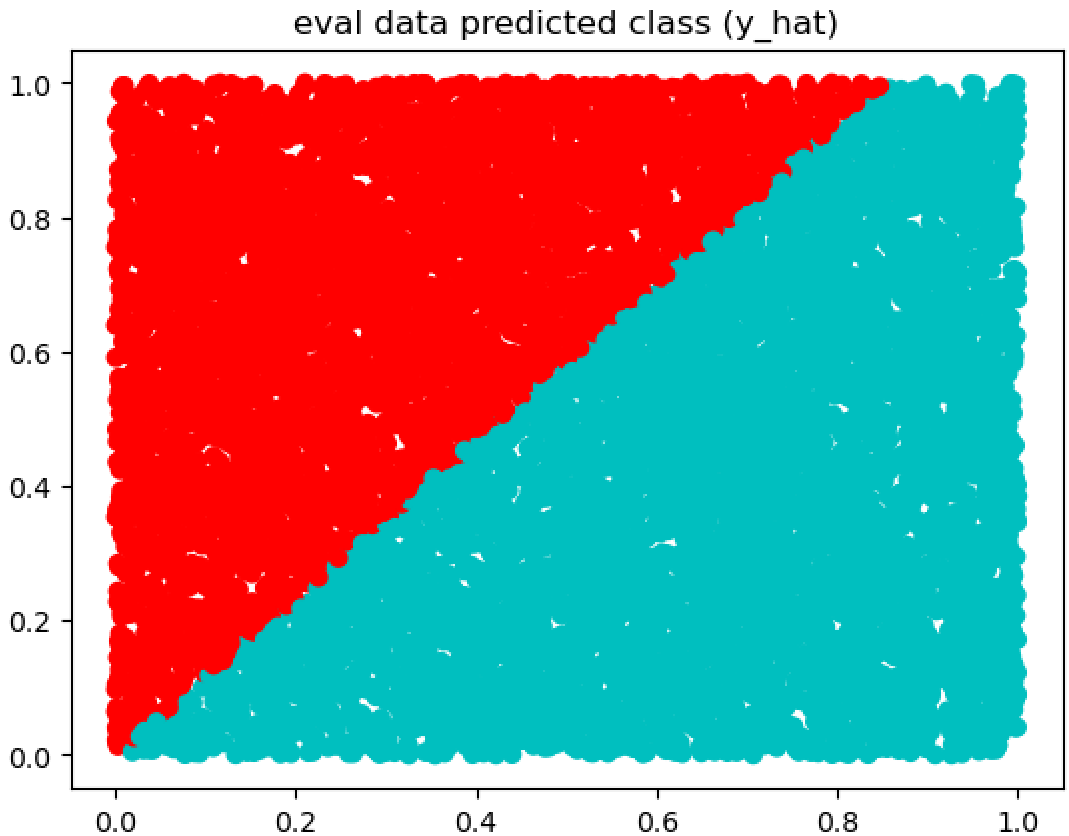


In [2]:

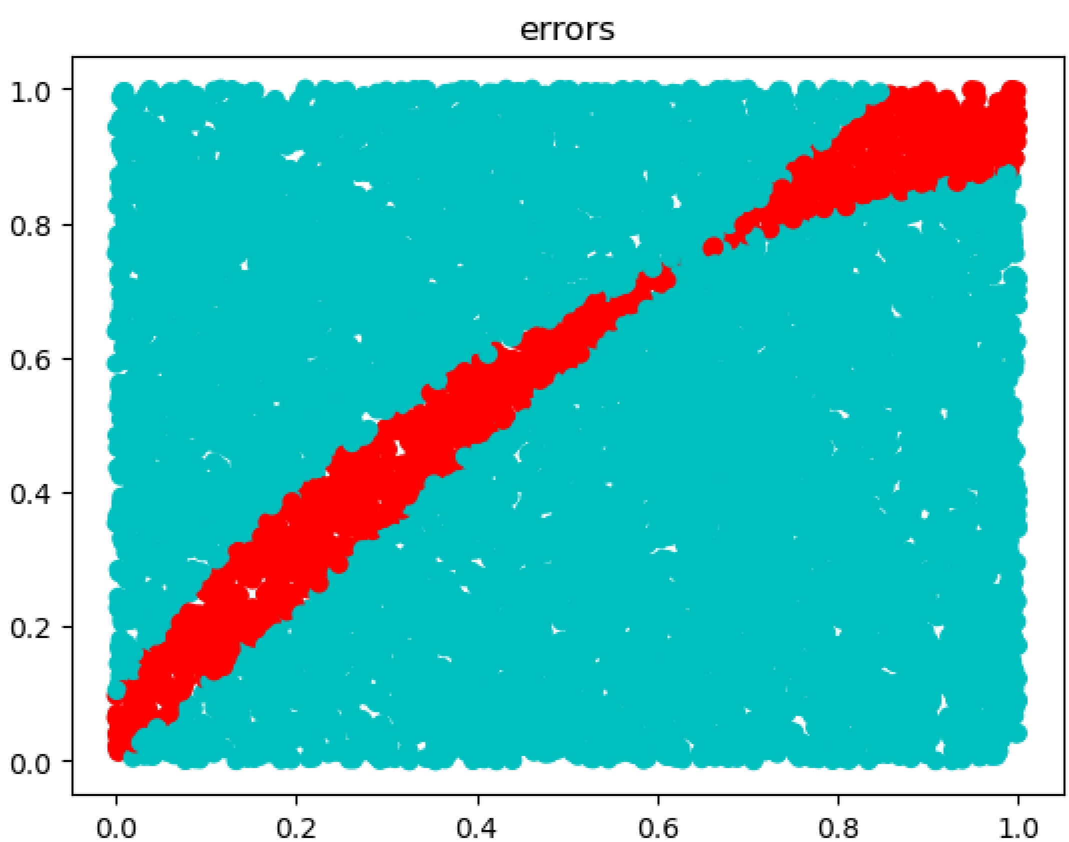
```
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==-1 else 'r' for i in y_eval[:,0]])
plt.title('eval data true class')
plt.show()
```



In [3]:
```
## Classifier 1

# 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()

# Problem 2a comment:
# The training data appears to be split along a curved decision boundary
# so the case where x^T is [x1 x2], which makes a linear decision boundary,
# has significant errors.

# % error = 1102/10000 = 0.1102 = 11.02%
```



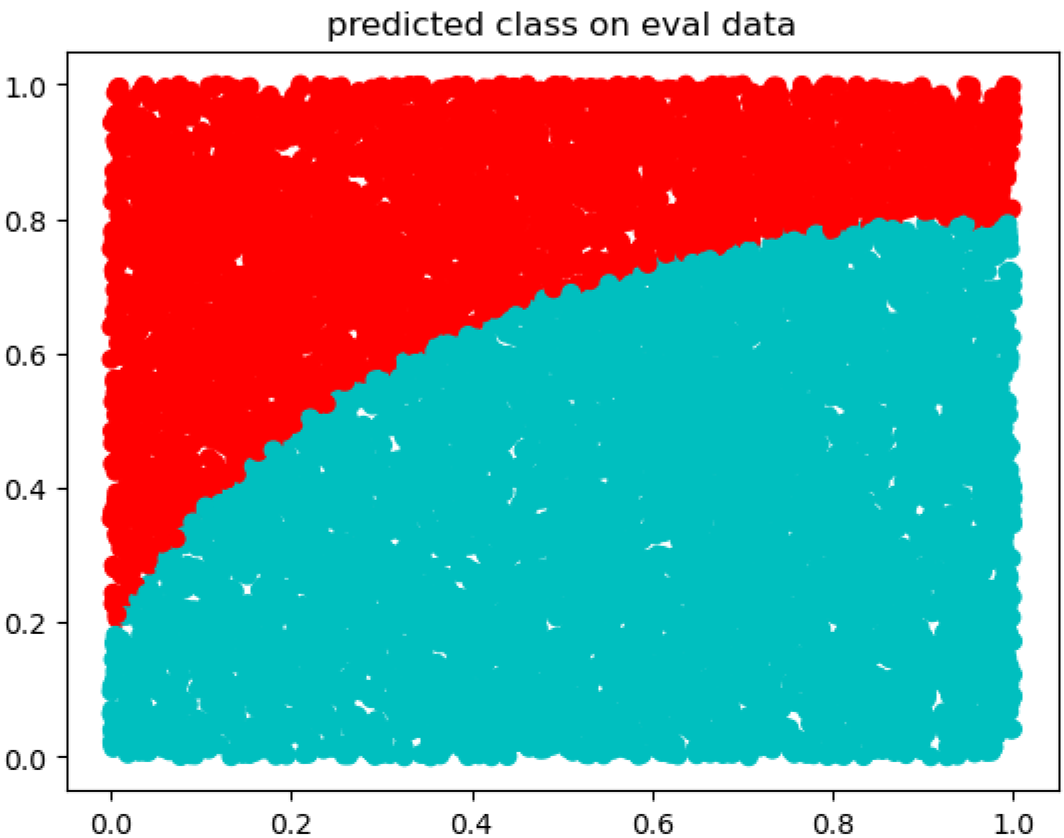
In [4]:
```
error_vec = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat, y_eval))]
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==0 else 'r' for i in error_vec])
plt.title('errors')
plt.show()

print('Errors: '+ str(sum(error_vec)))
```



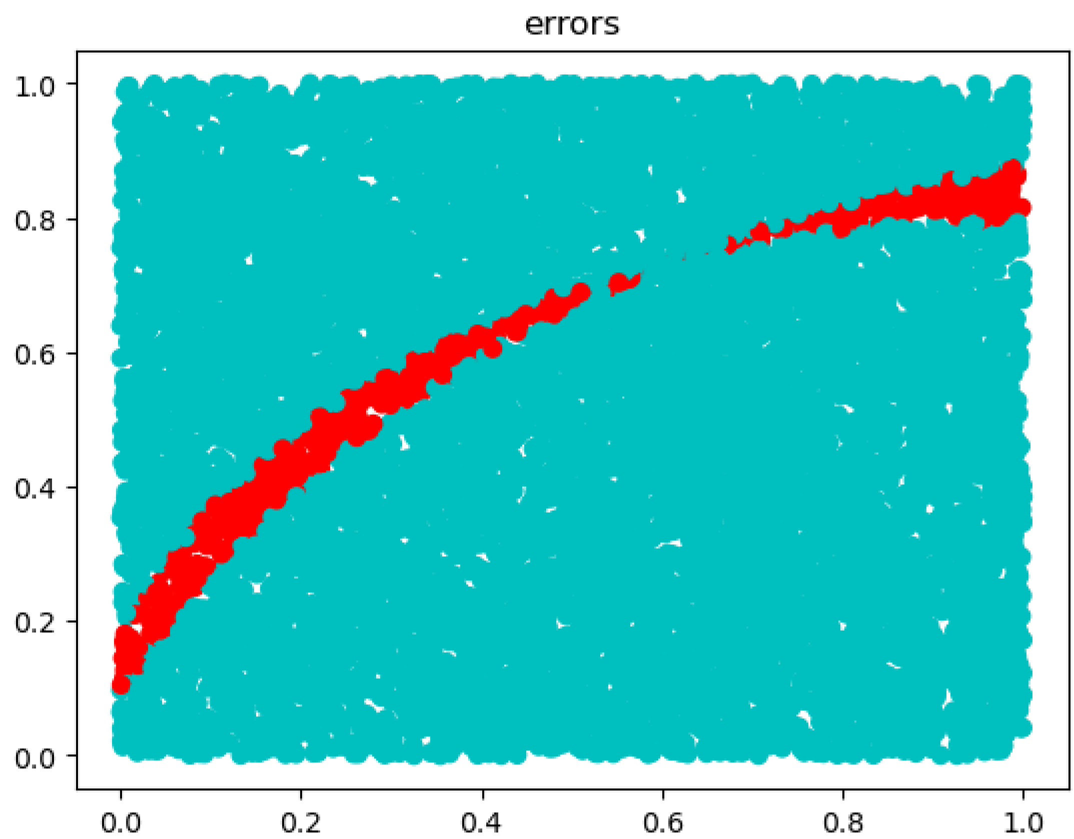

```
Errors: 1102
```

2b)

In [5]:
```
## Classifier 2
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_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()

# Problem 2b comment:
# The curved decision boundary fits the training data much better as
# the training data appears to be split along a curve

# % error = 542/10000 = 0.0542 = 5.42%
```



In [6]:
```
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)))
```

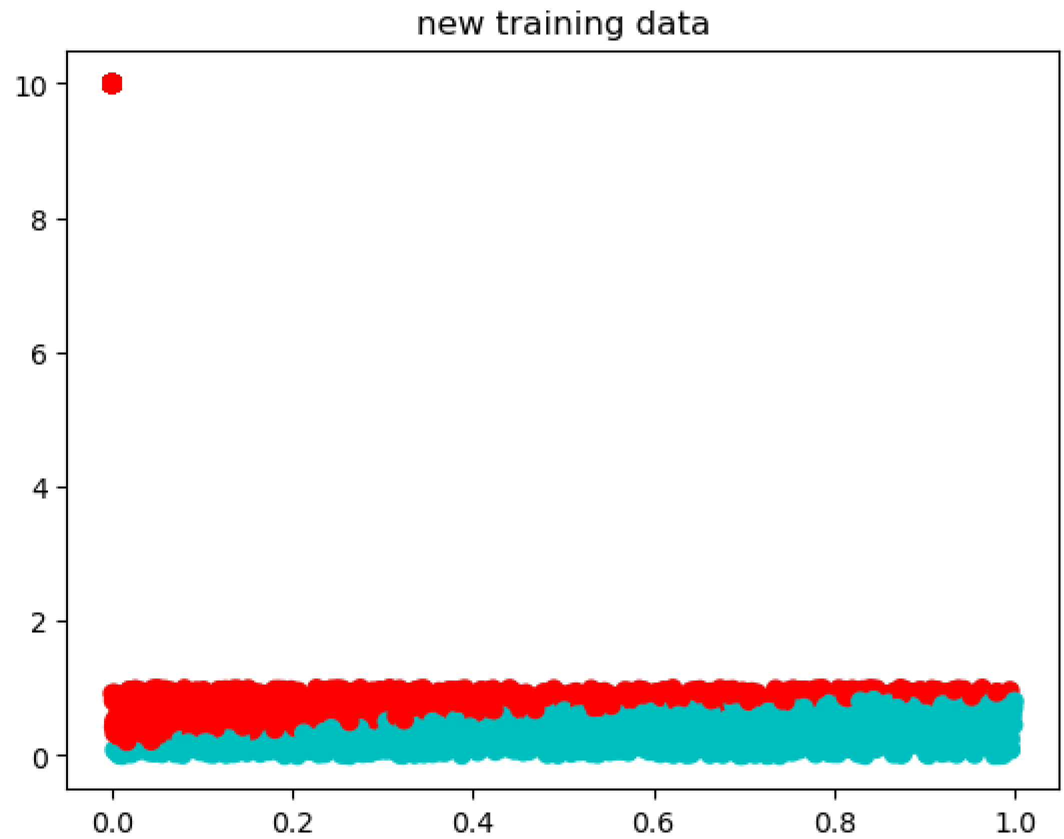



```
Error: 542
```

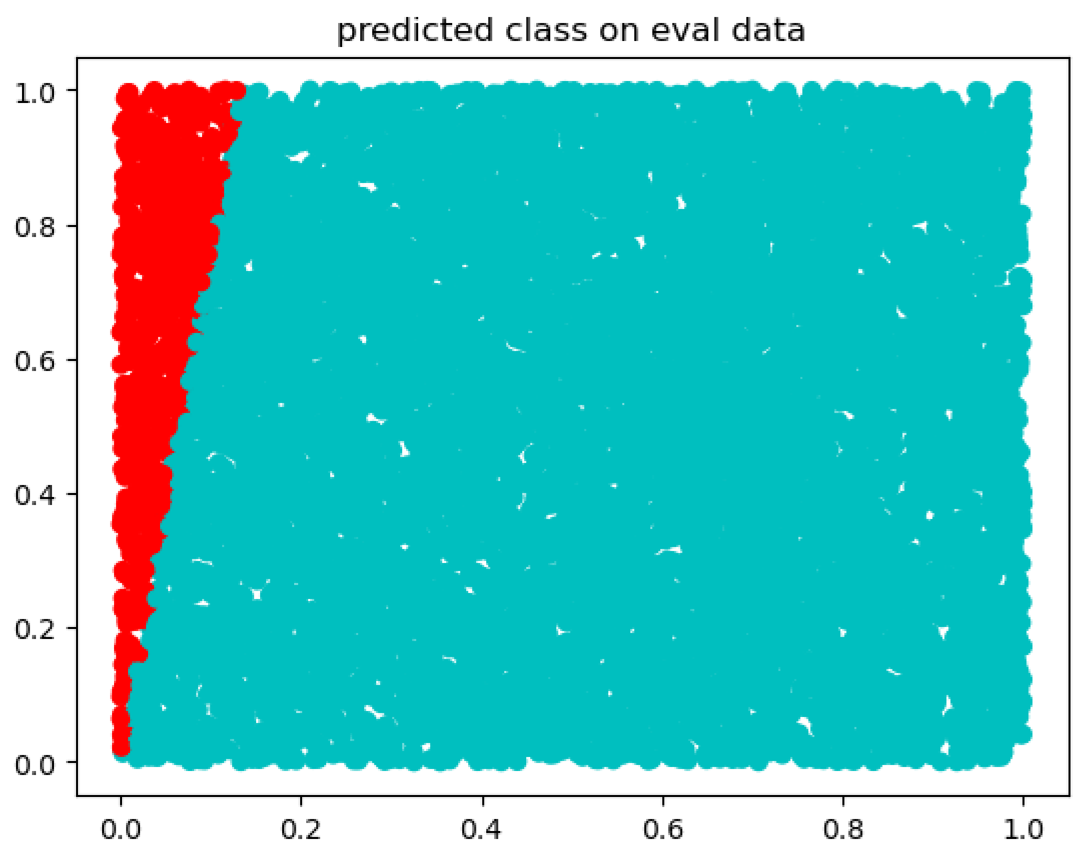
2c)

In [7]:

```
## create new, correctly labeled points
n_new = 1000 #number of new datapoints
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
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
plt.title('new training data')
plt.show()

# Problem 2c comment:
# The decision boundary shifts towards the outliers
# the further away the outliers are, the more it tilts towards them
# and the # of errors increases more and more
#
```

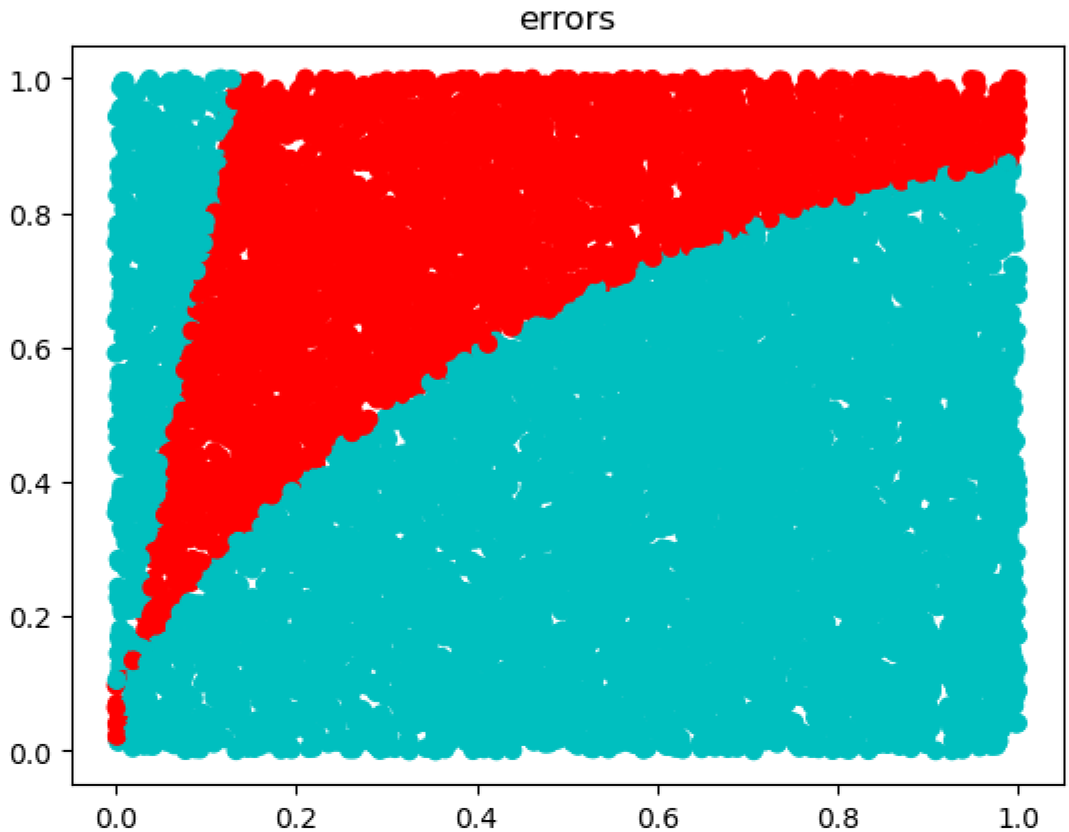


In [8]:
```
#train with new data
w_opt_outlier = np.linalg.inv(x_train_outlier.transpose()@x_train_outlier)@x_train_outli
y_hat_outlier = np.sign(x_eval@w_opt_outlier)

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



In [9]:
```
error_vec = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat_outlier, y_eval))]
```

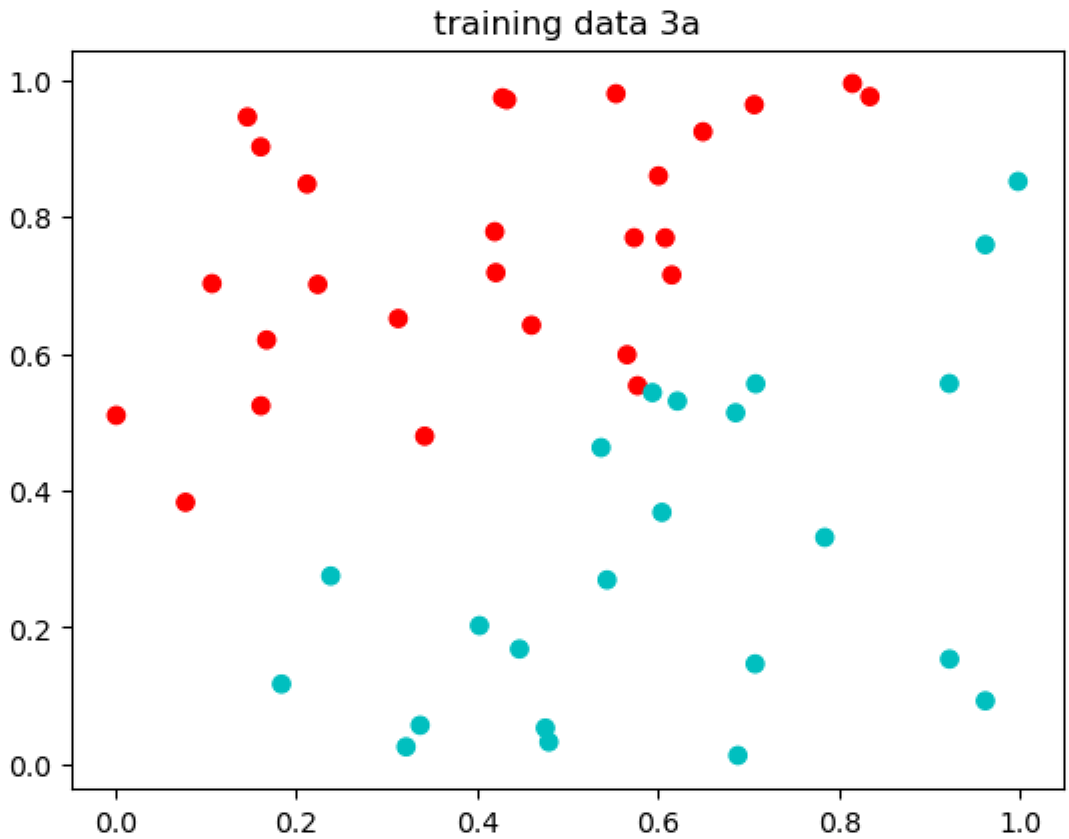
```
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==0 else 'r' for i in error_vec])
plt.title('errors')
plt.show()

print('Errors: '+ str(sum(error_vec)))
```




```
Errors: 3277
```

In [10]:

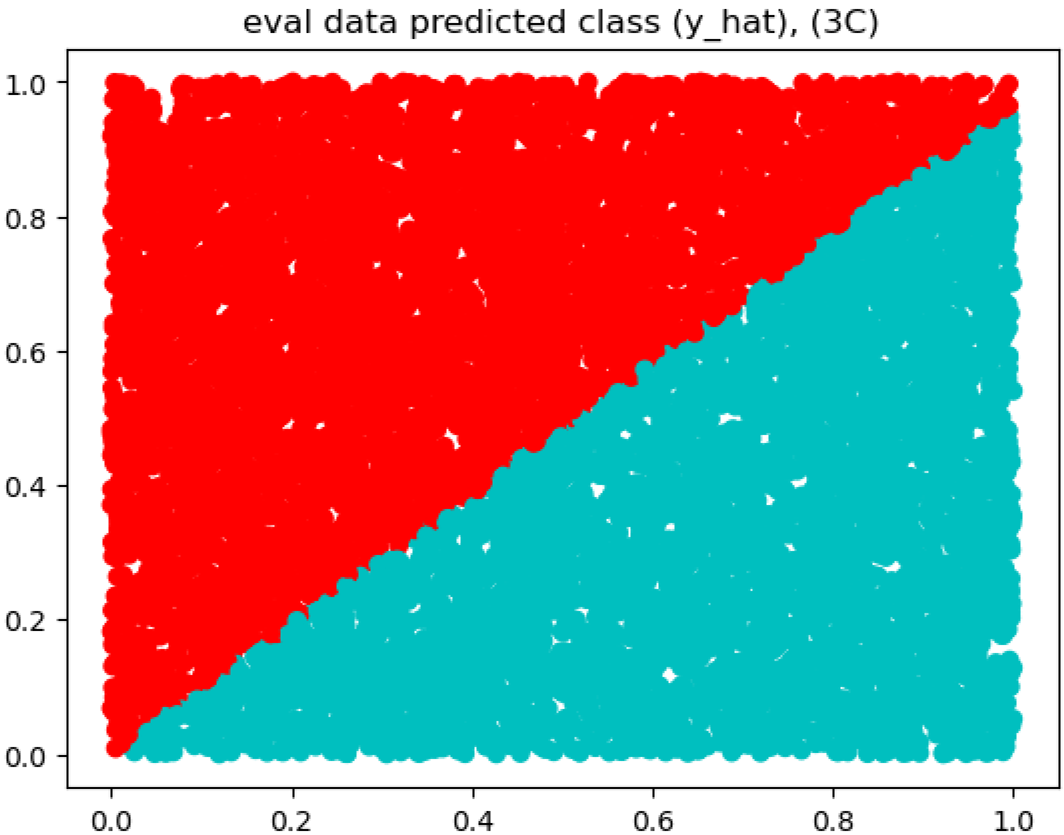
```
### 3a ###

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 da

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)

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 3a')
plt.show()
```



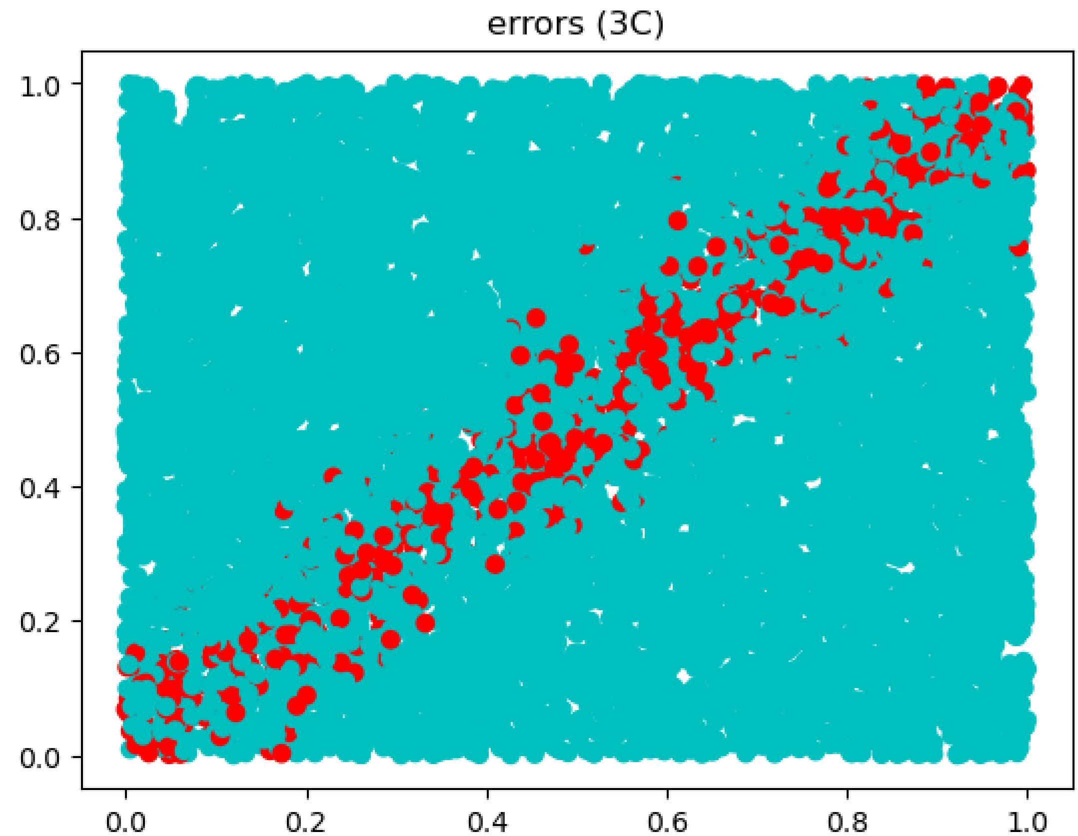
In [11]:
```
### 3b ###
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==-1 else 'r' for i in y_eval[:,0]])
plt.title('eval data true class, (3B)')
plt.show()
```

In [12]:
```
### 3c ###
## Classifier 1

# 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), (3C)')
plt.show()
```

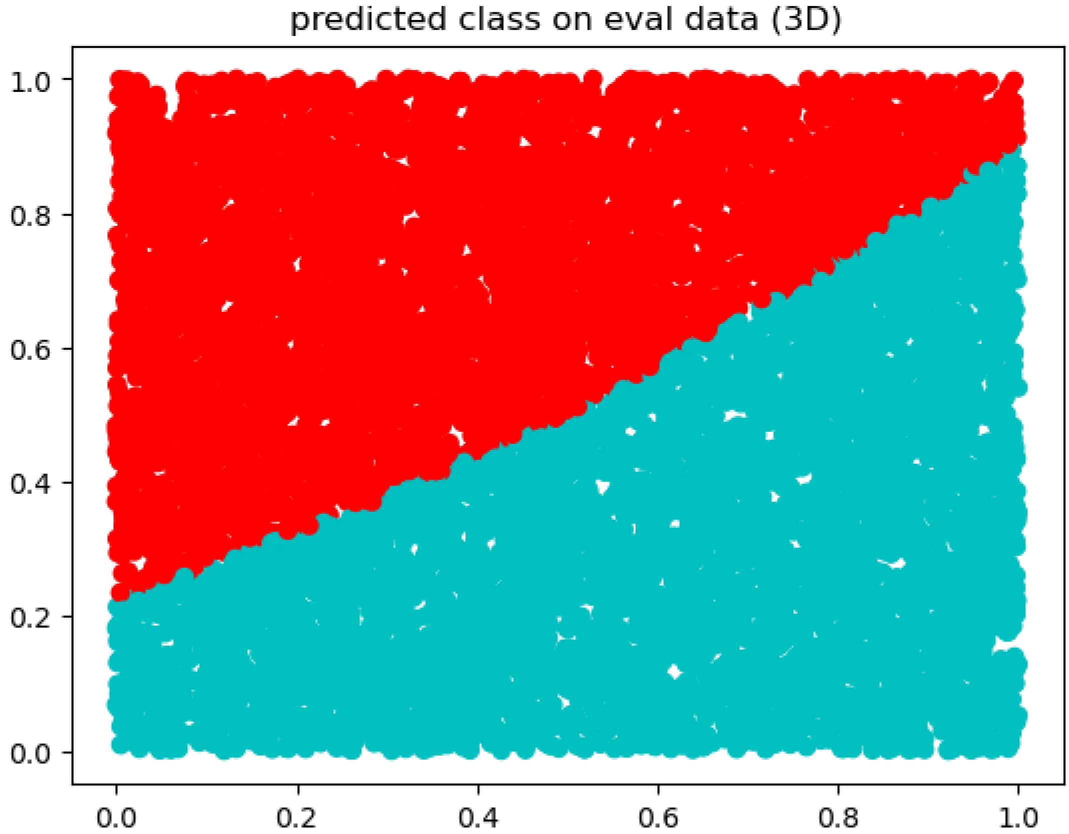


In [13]:
```
error_vec = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat, y_eval))]
plt.scatter(x_eval[:,0],x_eval[:,1], color=['c' if i==0 else 'r' for i in error_vec])
plt.title('errors (3C)')
plt.show()

print('Errors: '+ str(sum(error_vec)))
print('Total Samples = ' ,x_eval.shape[0])
print("Percentage error = ", sum(error_vec)/x_eval.shape[0])
```

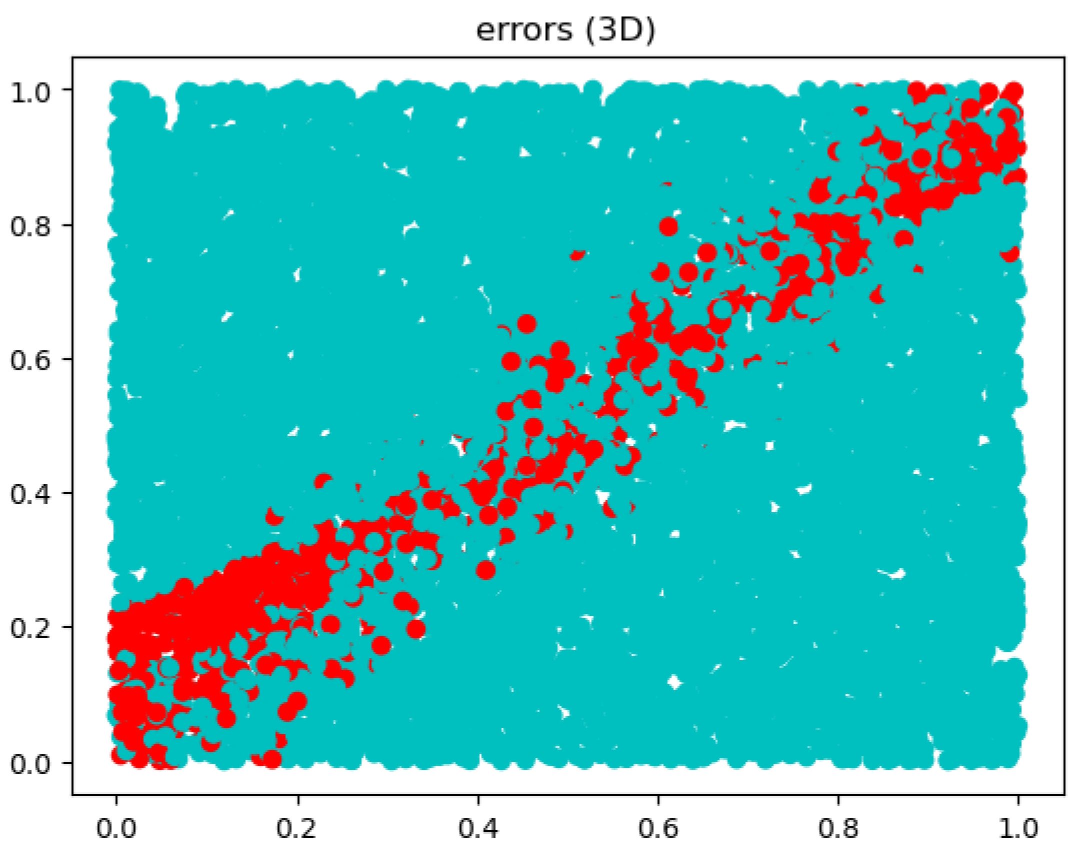



```
Errors: 759
Total Samples =  10000
Percentage error =  0.0759
```

In [14]:
```
### 3d ###

## Classifier 2
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_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 (3D)')
plt.show()
```



In [15]:
```
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 (3D)')
plt.show()

print('Error: '+ str(sum(error_vec_2)))
```

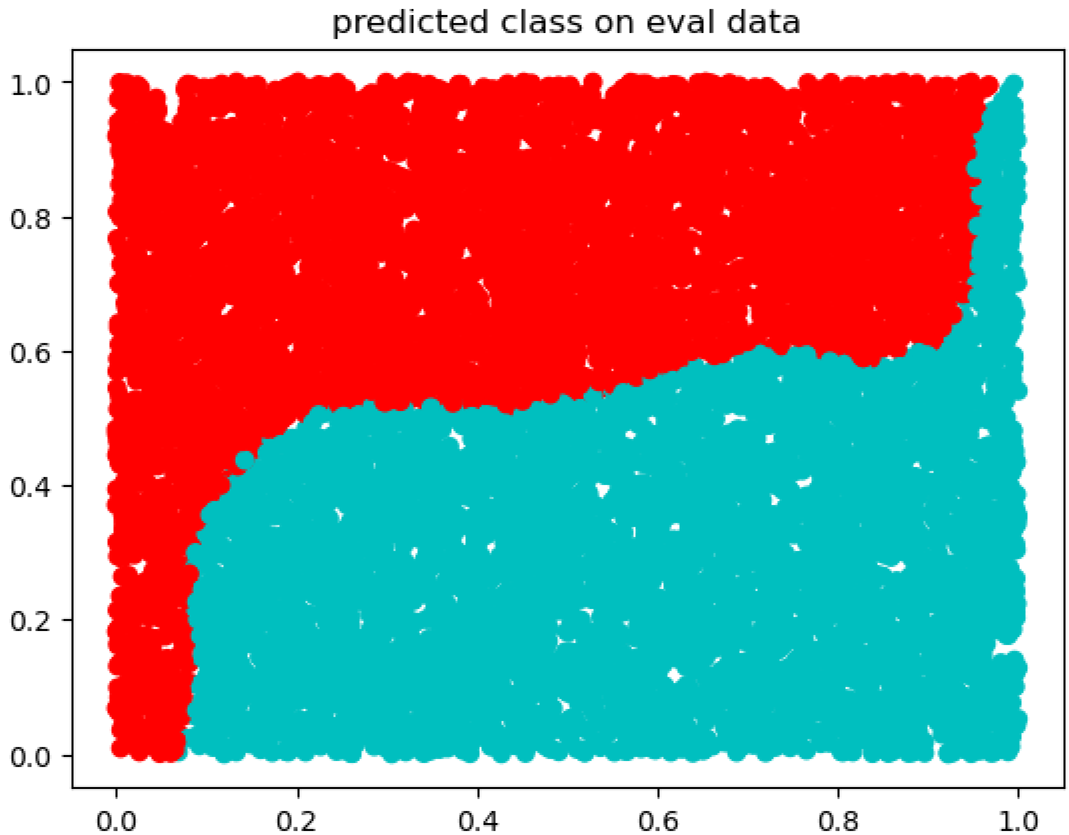



```
Error: 1066
```

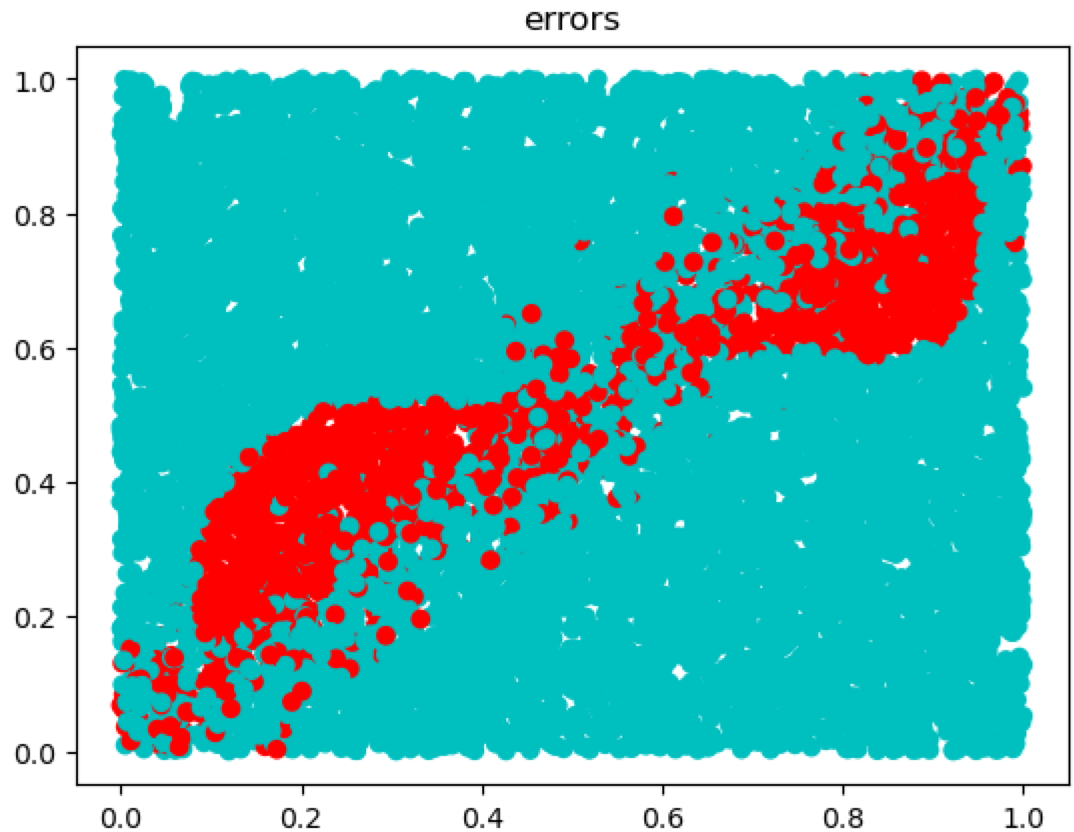
In [16]:
```
### 3e ###
```

```
## Classifier 3
x_train_3 = np.hstack((x_train**6, x_train**5, x_train**4, x_train**3, x_train**2, x_tra
x_eval_3 = np.hstack((x_eval**6, x_eval**5, x_eval**4, x_eval**3, x_eval**2, x_eval, np.

w_opt_3 = np.linalg.inv(x_train_3.transpose()@x_train_3)@x_train_3.transpose()@y_train
print(w_opt_3.shape, x_eval_3.shape)
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()
```

```
(13, 1) (10000, 13)
```



In [19]:
```
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)))
```




```
Error: 1677
```

In [ ]:

```
### 3f ###
# The highest order classifier 3 performed the worst because it overfits
# to noise in the small training sample set
```