①

(a_i, b_i) ; $i = 1, \dots, m$ → we want $w(a)$ so that $w(a_i) \approx b_i$ for $i = 1, \dots, m$.

a) $w(a)$ is a degree p polynomial.

degree p polynomial ⇒ $\alpha_0 + \alpha_1 x + \alpha_2 x^2 + \dots + \alpha_p x^p$

Now, we have m point as (a_i, b_i), $i = 1, \dots, m$. Then to form a polynomial with those points we will need that $p = m-1$

$$\left.\begin{aligned} w(a_1) &= \alpha_0 + \alpha_1 b_1 + \dots + \alpha_p b_1^p \\ w(a_2) &= \alpha_0 + \alpha_1 b_2 + \dots + \alpha_p b_2^p \end{aligned}\right\} \Rightarrow w(a_i) = x_0 + x_1 a_i + x_2 a_i^2 + \dots + x_p a_i^p$$

b)

$$\alpha = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} w(a_1) \\ w(a_2) \\ \vdots \\ w(a_m) \end{bmatrix}, \quad A = \begin{bmatrix} 1 & a_1 & a_1^2 & \dots & a_1^p \\ 1 & a_2 & a_2^2 & \dots & a_2^p \\ \vdots & & & \ddots & \\ 1 & a_m & a_m^2 & \dots & a_m^p \end{bmatrix}, \quad x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}$$

$$Ax = \alpha \Rightarrow \begin{bmatrix} 1 & a_1 & a_1^2 & \dots & a_1^p \\ 1 & a_2 & a_2^2 & \dots & a_2^p \\ \vdots & & & \ddots & \\ 1 & a_m & a_m^2 & \dots & a_m^p \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

c) $p = 1 \Rightarrow x_0 + a_1 x_1 = b_1$
$p = 2 \Rightarrow x_0 + x_1 a_1 + x_2 a_2^2 = b_2$
$p = 3 \Rightarrow x_0 + x_1 a_1 + x_2 a_2^1 + x_3 a_3^3 = b_3$

②

a) $\min_w \|x - Tw\|_2^2$; T is n-by-r of orthonormal columns.

Let w^* the w that minimize $\|x - Tw\|_2^2$

Then, $Tw^* - \vec{x} = \text{proj}_{C(A)} \vec{x} - \vec{x} \Rightarrow Tw^* - \vec{x} \in C(T)^\perp$

$\Rightarrow C(T) = N(T) \Rightarrow Tw^* - \vec{x} \in N(T^T) \Rightarrow T^T(Tw^* - \vec{x}) = \vec{0}$
$\Rightarrow T^T T w^* - T^T x = 0 \Rightarrow T^T T w^* = T^T x$

b) $X = [x_1\ x_2 \dots x_p]$, X is n by p ; x_i is n by 1

$W = [w_1\ w_2 \dots w_p]$

We know thanks to ex 2.a) $T^T T w_i = T^T x_i$.

T is a n by r matrix with orthonormal columns ⇒ T has inverse. ⇒ $T^T T$ has an inverse.

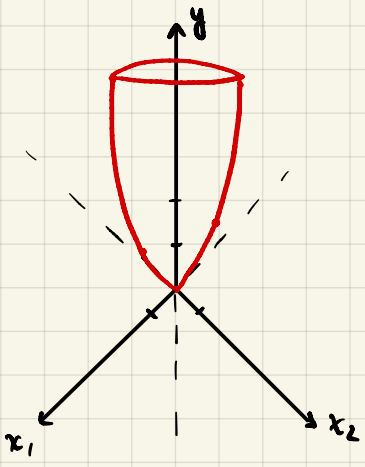
Then $(T^T T)^{-1} T^T T w_i = (T^T T)^{-1} T^T x_i \Rightarrow w_i = (T^T T)^{-1} T^T x_i$

$\therefore W = (T^T T)^{-1} T^T X$

④ $Q = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$

a) $\text{rank}(Q) = 2 \Rightarrow Q$ is full rank $\Rightarrow Q > 0$

b) $y = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & 2x_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1^2 + 2x_2^2$



⑤ $P > 0$ and $Q > 0$ (simetric) positive definite $n \times n$ matrices

$[QPQ > 0]$

As Q is simetric and positive definite $\Rightarrow$ $x^T Q x > 0$ and $Qx \neq 0$ ①

Then $x^T QPQx = (Qx)^T P (Qx) > 0$ since $P > 0$ and ①

Or to know if $QPQ > 0$ we need to see that every determinant is > 0

Then $|QPQ| = \overbrace{|Q||P||Q|}^{>0 \; >0 \; >0 \text{ because } P>0 \text{ & } Q>0} > 0 \Rightarrow QPQ > 0$

③ a) Set of orthonormal basis found:

```
[[ 4.47213595e-01 -3.65148372e-01 -6.32455532e-01 -5.16397779e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -2.20934382e-14]
 [ 4.47213595e-01  5.47722558e-01  3.16227766e-01 -3.87298335e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00  5.00000000e-01]
 [ 4.47213595e-01 -3.65148372e-01  2.24693342e-15  6.45497224e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00  5.00000000e-01]
 [ 4.47213595e-01  5.47722558e-01 -3.16227766e-01  3.87298335e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -5.00000000e-01]
 [ 4.47213595e-01 -3.65148372e-01  6.32455532e-01 -1.29099445e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -5.00000000e-01]]
```

we can see that the first basis found is $\begin{bmatrix} 4.47e\text{-}01 \\ 4.47e\text{-}01 \\ 4.47e\text{-}01 \\ 4.47e\text{-}01 \\ 4.47e\text{-}01 \end{bmatrix} = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$

$\Rightarrow$ the first basis is equal to t_1.

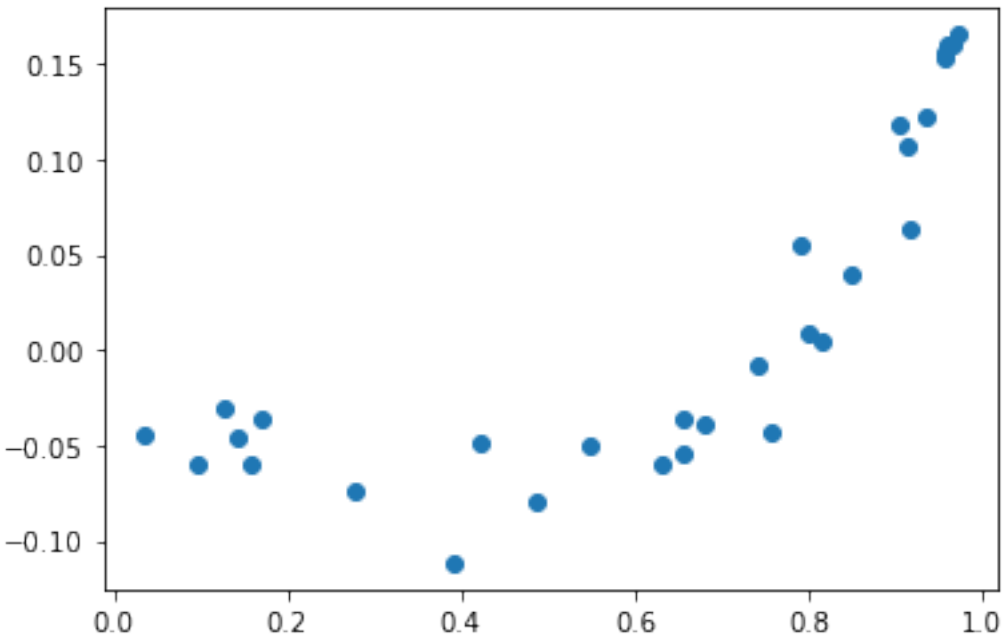
1 c)

gram_schmidt_orth

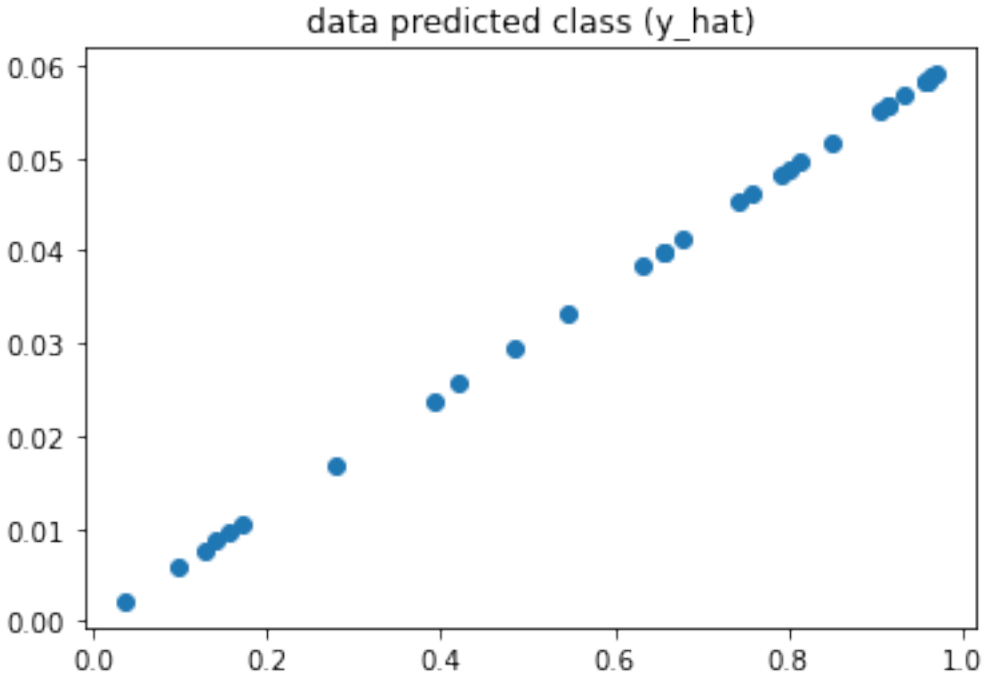
February 21, 2022

[11]:
```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

in_data = loadmat('polydata.mat')
# print([key for key in in_data])

n_train = np.size(y_train)

x_train = in_data['a']
y_train = in_data['b']

plt.scatter(x=x_train,y=y_train)
```

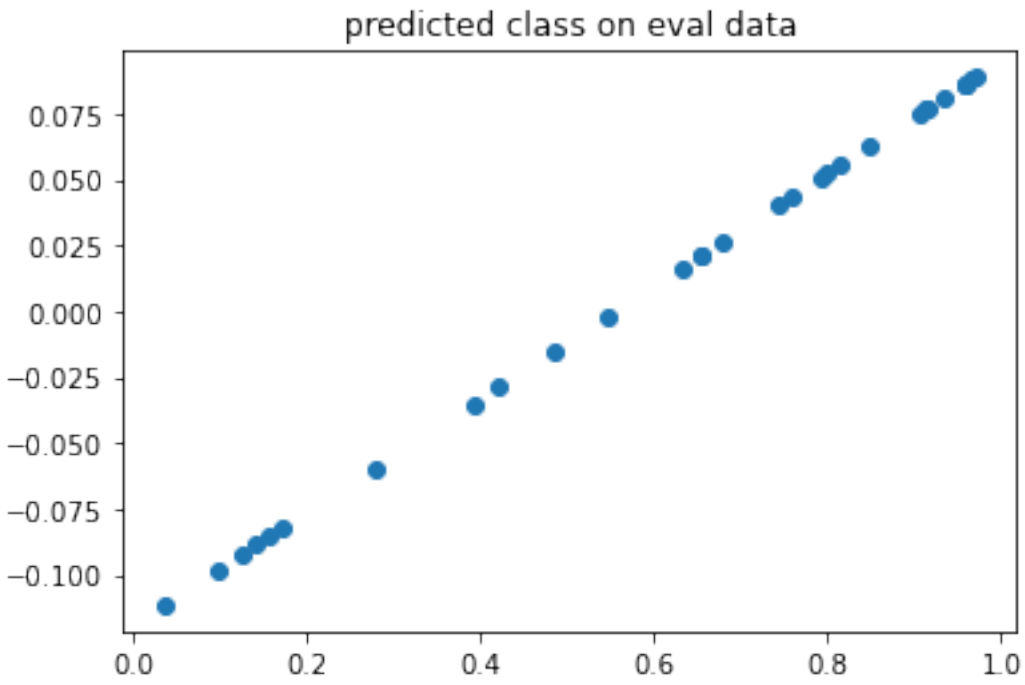
[11]:
```
<matplotlib.collections.PathCollection at 0x7fdc8b2e0e50>
```



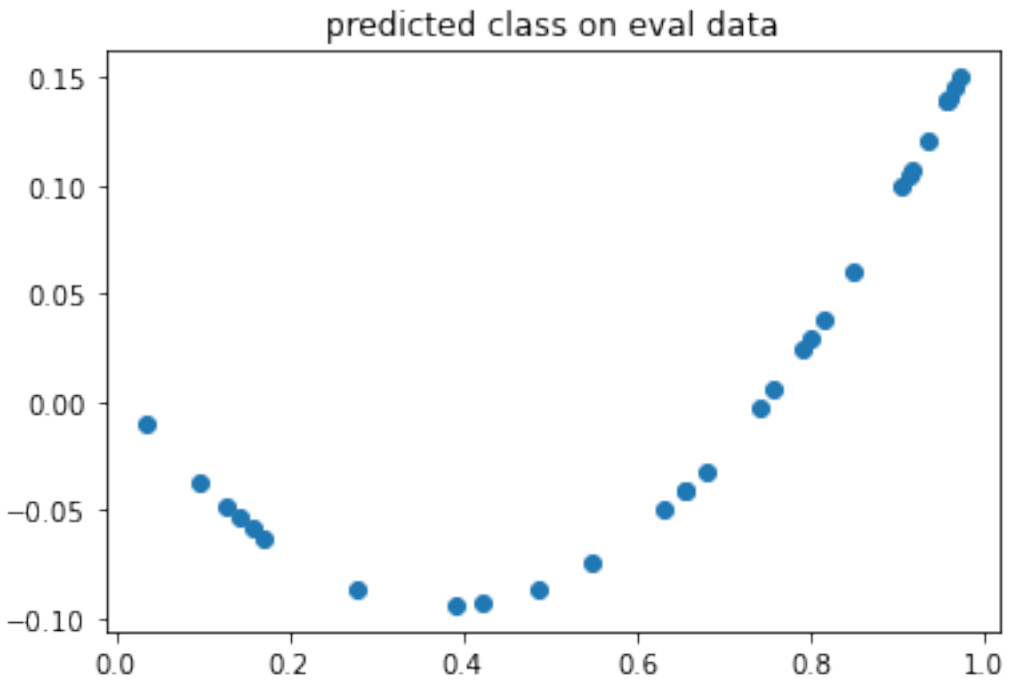
[36]:
```
## Classifier 1 -> p=1

# 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 = x_train@w_opt

plt.scatter(x=x_train, y=y_hat)
plt.title('data predicted class (y_hat)')
plt.show()
```



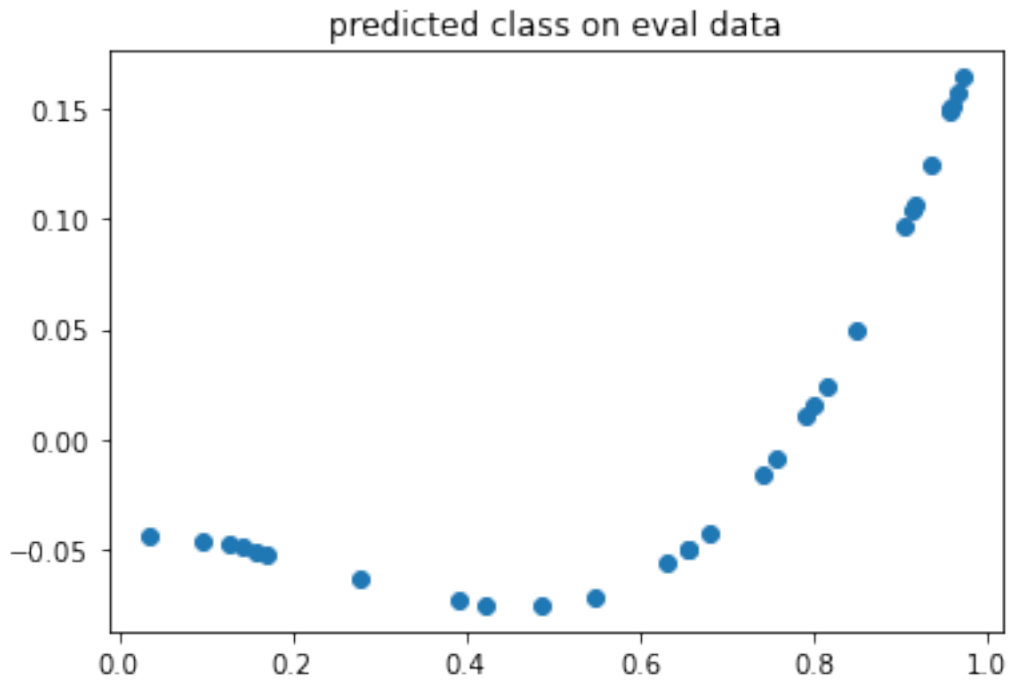
[41]:
```
## p = 1
x_train_1 = np.hstack((x_train, np.ones((n_train,1)) ))

w_opt_1 = np.linalg.inv(x_train_1.transpose()@x_train_1)@x_train_1.
 ↪transpose()@y_train
y_hat_1 = x_train_1@w_opt_1

plt.scatter(x=x_train, y=y_hat_1)
plt.title('predicted class on eval data')
plt.show()
```



[42]:
```
## p = 2
x_train_2 = np.hstack((x_train**2, x_train, np.ones((n_train,1)) ))

w_opt_2 = np.linalg.inv(x_train_2.transpose()@x_train_2)@x_train_2.
 ↪transpose()@y_train
y_hat_2 = x_train_2@w_opt_2

plt.scatter(x=x_train, y=y_hat_2)
plt.title('predicted class on eval data')
plt.show()
```



[43]:
```
## p = 3
x_train_3 = np.hstack((x_train**3,x_train**2, x_train, np.ones((n_train,1)) ))

w_opt_3 = np.linalg.inv(x_train_3.transpose()@x_train_3)@x_train_3.
 ↪transpose()@y_train
y_hat_3 = x_train_3@w_opt_3

plt.scatter(x=x_train, y=y_hat_3)
plt.title('predicted class on eval data')
plt.show()
```



[67]:
```python
import numpy as np

def gram_schmidt(B):
    """Orthogonalize a set of vectors stored as the columns of matrix B."""
    # Get the number of vectors.
    m, n = B.shape
    # Create new matrix to hold the orthonormal basis
    U = np.zeros([m,n])
    for j in range(n):
        # To orthogonalize the vector in column j with respect to the
        # previous vectors, subtract from it its projection onto
        # each of the previous vectors.
        v = B[:,j].copy()
        for k in range(j):
            v -= np.dot(U[:, k], B[:, j]) * U[:, k]
        if np.linalg.norm(v)>1e-10:
            U[:, j] = v / np.linalg.norm(v)
    return U

if __name__ == '__main__':
    B1 = np.array([[1.0, 1.0, 0.0], [2.0, 2.0, 0.0], [2.0, 2.0, 1.0]])
    A1 = gram_schmidt(B1)
    print(A1)
    A2 = gram_schmidt(np.random.rand(4,2)@np.random.rand(2,5))
```

```
    print(A2.transpose()@A2)
```

```
[[ 0.33333333  0.         -0.2981424 ]
 [ 0.66666667  0.         -0.59628479]
 [ 0.66666667  0.          0.74535599]]
[[ 1.00000000e+00 -4.71844785e-16  0.00000000e+00  0.00000000e+00
   0.00000000e+00]
 [-4.71844785e-16  1.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00]]
```

[69]:
```
from scipy.io import loadmat
import matplotlib.pyplot as plt

in_data = loadmat('movie.mat')
x_data = in_data['X']
x_size = np.size(x_data)

X = np.hstack((np.ones((5,1)), x_data))
gram_schmidt(X)
```

```
[[ 4.47213595e-01 -3.65148372e-01 -6.32455532e-01 -5.16397779e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -2.20934382e-14]
 [ 4.47213595e-01  5.47722558e-01  3.16227766e-01 -3.87298335e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00  5.00000000e-01]
 [ 4.47213595e-01 -3.65148372e-01  2.24693342e-15  6.45497224e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00  5.00000000e-01]
 [ 4.47213595e-01  5.47722558e-01 -3.16227766e-01  3.87298335e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -5.00000000e-01]
 [ 4.47213595e-01 -3.65148372e-01  6.32455532e-01 -1.29099445e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -5.00000000e-01]]
```

[83]:
```
import math
t_1 = np.ones((5,1))*(1/math.sqrt(5))
# X   t_1*W
```

[83]:
```
array([[0.4472136],
       [0.4472136],
       [0.4472136],
       [0.4472136],
       [0.4472136]])
```