①

face_classification_starter

May 2, 2022

[220]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
dataset = loadmat('face_emotion_data.mat')

X, y = dataset['X'], dataset['y']
n, p = np.shape(X)

y[y==-1] = 0  # use 0/1 for labels instead of -1/+1
X = np.hstack((np.ones((n,1)), X))  # append a column of ones
```

0.0.1 1)

a)

[202]:
```
#Logistic function => f(z) = 1/(1+e^(-z))
```

[205]:
```
## Train NN
Xb = np.hstack((np.ones((n,1)), X))
q = np.shape(y)[1] #number of classification problems
M = 32 #number of hidden nodes

## initial weights
V = np.random.randn(M, q);
W = np.random.randn(p+2, M);

def logsig(_x):
    return 1/(1+np.exp(-_x))

H = logsig(Xb@W)
Yhat = logsig(H@V)
```

[194]:
```
Yhat = [1 if i > 0.5 else 0 for i in Yhat ]
Yhat
```

```
[194]: [1,
 1,
 1,
 1,
 1,
 0,
 1,
 0,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
 0,
 0,
 1,
 1,
 1,
 0,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
 1,
 0,
 1,
 1,
 0,
 1,
 1,
 1,
 1,
 0,
 1,
```

```
1,
1,
1,
1,
0,
1,
1,
0,
1,
1,
1,
0,
1,
0,
1,
0,
1,
1,
0,
0,
0,
1,
1,
1,
1,
0,
1,
0,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
0,
1,
0,
1,
1,
0,
1,
0,
1,
```

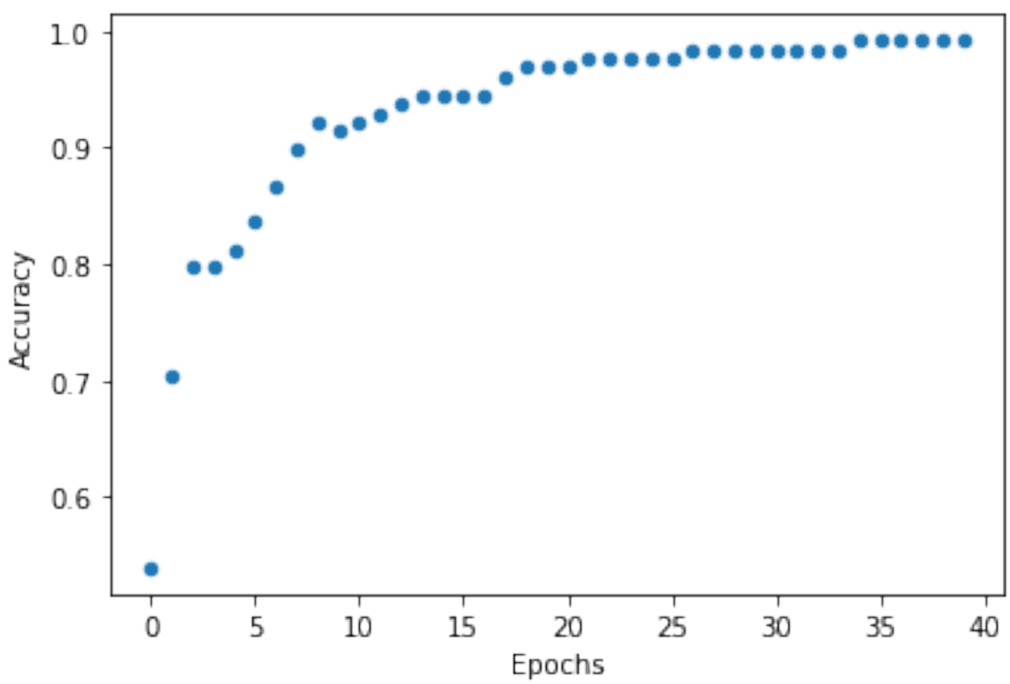
```
 0,
 1,
 0,
 0,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
 1,
 0,
 1,
 1,
 0,
 1,
 0,
 1,
 0,
 0,
 0,
 1,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1]
```

b)

[170]:
```
L = 40
## initial weights
V = np.random.randn(M+1, q);
W = np.random.randn(p+2, M);

alpha = 0.05
accuracy = []
epochs = []
for epoch in range(L):
    ind = np.random.permutation(n)
    epochs.append(epoch)
```

```python
    for i in ind:
        # Forward-propagate
        H = logsig(np.hstack((np.ones((1,1)), Xb[[i],:]@W)))
        Yhat = logsig(H@V)
         # Backpropagate
        delta = (Yhat-y[[i],:])*Yhat*(1-Yhat)
        Vnew = V-alpha*H.T@delta
        gamma = delta@V[1:,:].T*H[:,1:]*(1-H[:,1:])
        Wnew = W - alpha*Xb[[i],:].T@gamma
        V = Vnew
        W = Wnew
    H = logsig(np.hstack((np.ones((n,1)), Xb@W)))
    Yhat = logsig(H@V)
    Yhat = [1 if i > 0.5 else 0 for i in Yhat ]
    accuracy.append(accuracy_score( Yhat, y ))
df = pd.DataFrame({"Epochs":epochs,"Accuracy":accuracy})
df.plot.scatter("Epochs","Accuracy")
```

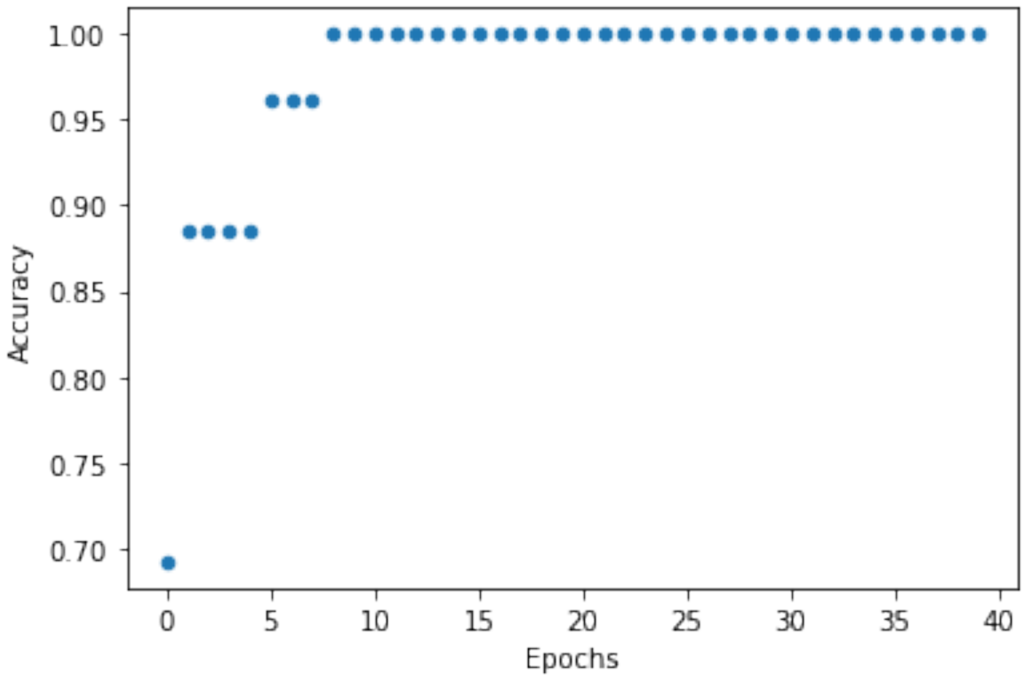
[170]: <AxesSubplot:xlabel='Epochs', ylabel='Accuracy'>



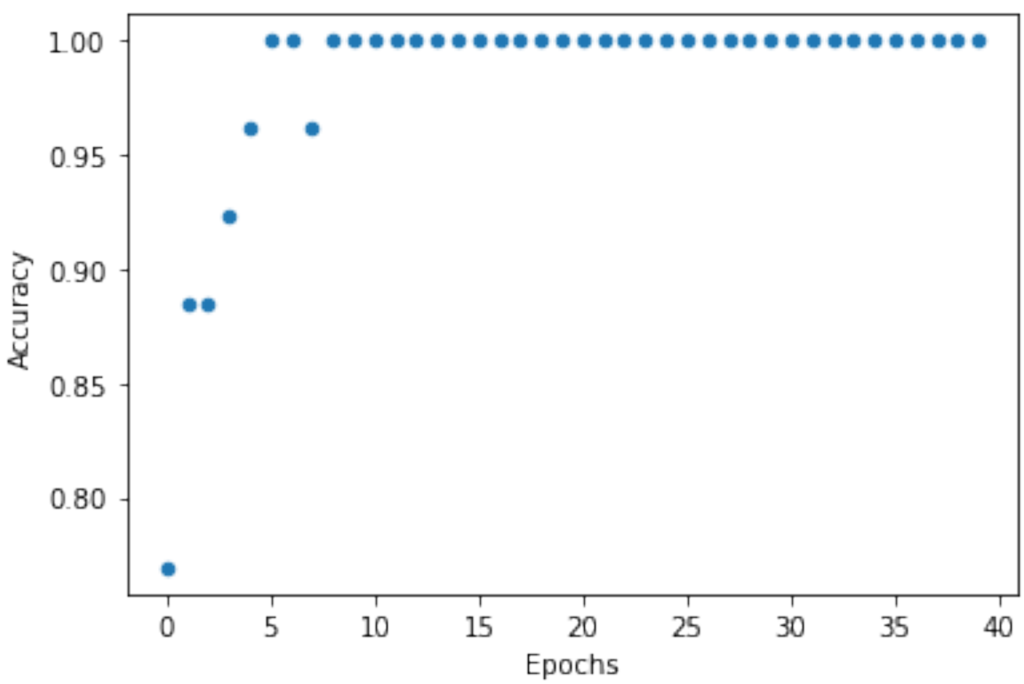
It takes about 35 epochs to reach 0% of error in the training set.

c)

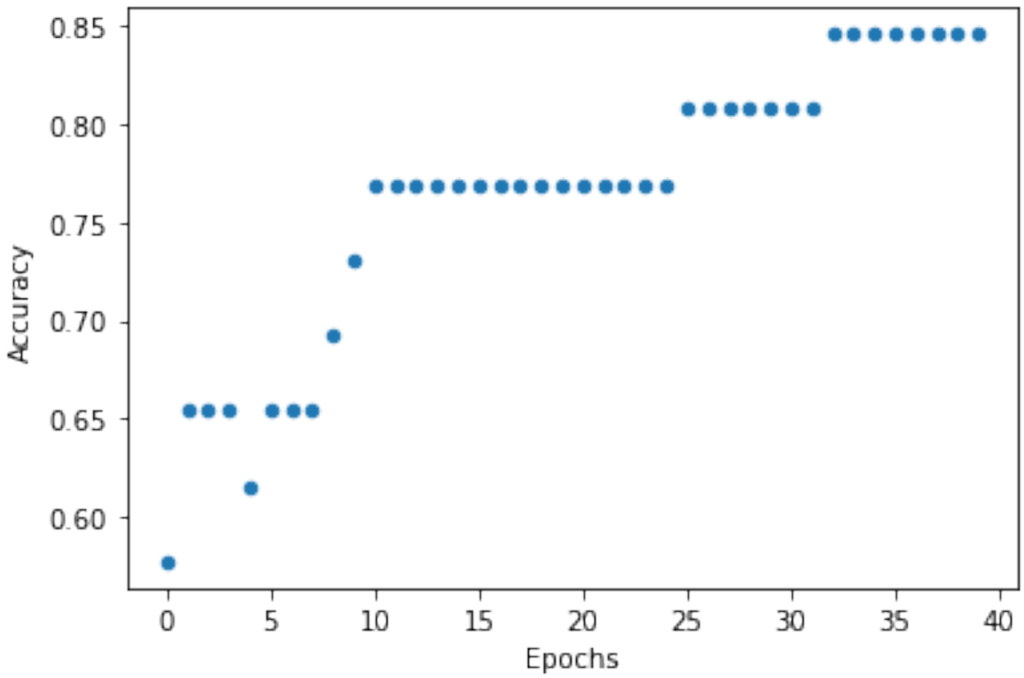
[224]:
```
L = 40
alpha = 0.05

for j in range(0,8):
    X_train, X_test, Y_train, Y_test = train_test_split(Xb, y, test_size = 0.2)
    q = np.shape(Y_train)[1]
    M = 32
    V = np.random.randn(M+1, q);
    W = np.random.randn(p+2, M);
    accuracy = []
    epochs = []
    for epoch in range(L):
        ind = np.random.permutation(len(X_train))
        epochs.append(epoch)
        for i in ind:
            # Forward-propagate
            H = logsig(np.hstack((np.ones((1,1)), X_train[[i],:]@W)))
            Yhat = logsig(H@V)
             # Backpropagate
            delta = (Yhat-Y_train[[i],:])*Yhat*(1-Yhat)
            Vnew = V-alpha*H.T@delta
            gamma = delta@V[1:,:].T*H[:,1:]*(1-H[:,1:])
            Wnew = W - alpha*X_train[[i],:].T@gamma
            V = Vnew
            W = Wnew
        H = logsig(np.hstack((np.ones((len(X_test),1)), X_test@W)))
        Yhat = logsig(H@V)
        Yhat = [1 if i > 0.5 else 0 for i in Yhat ]
        accuracy.append(accuracy_score( Yhat, Y_test ))
    df = pd.DataFrame({"Epochs":epochs,"Accuracy":accuracy})
    df.plot.scatter("Epochs","Accuracy")
```

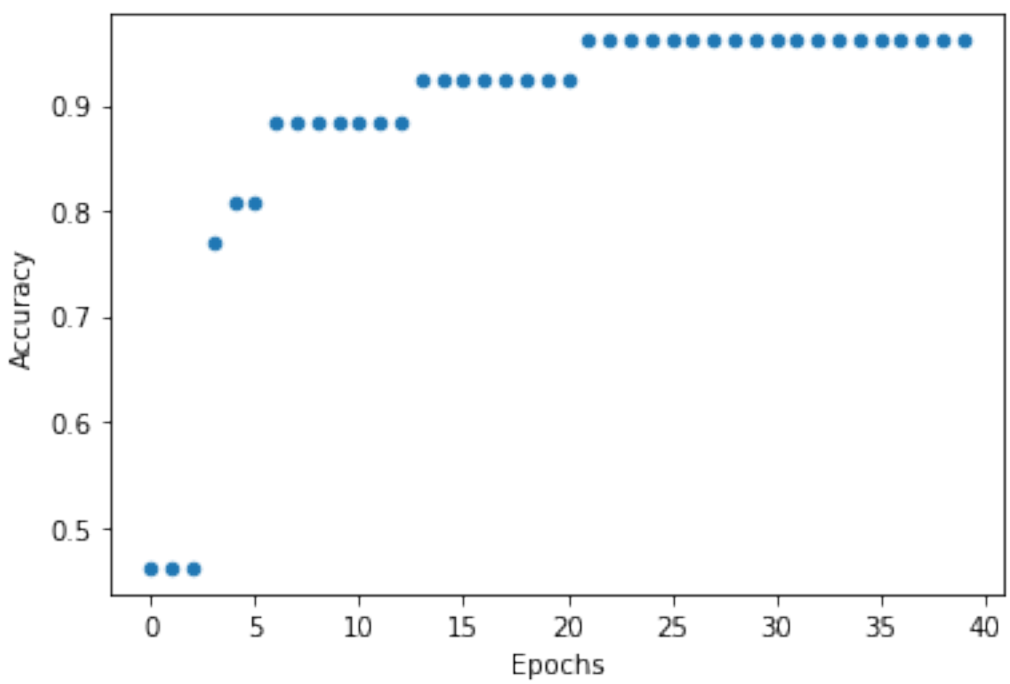


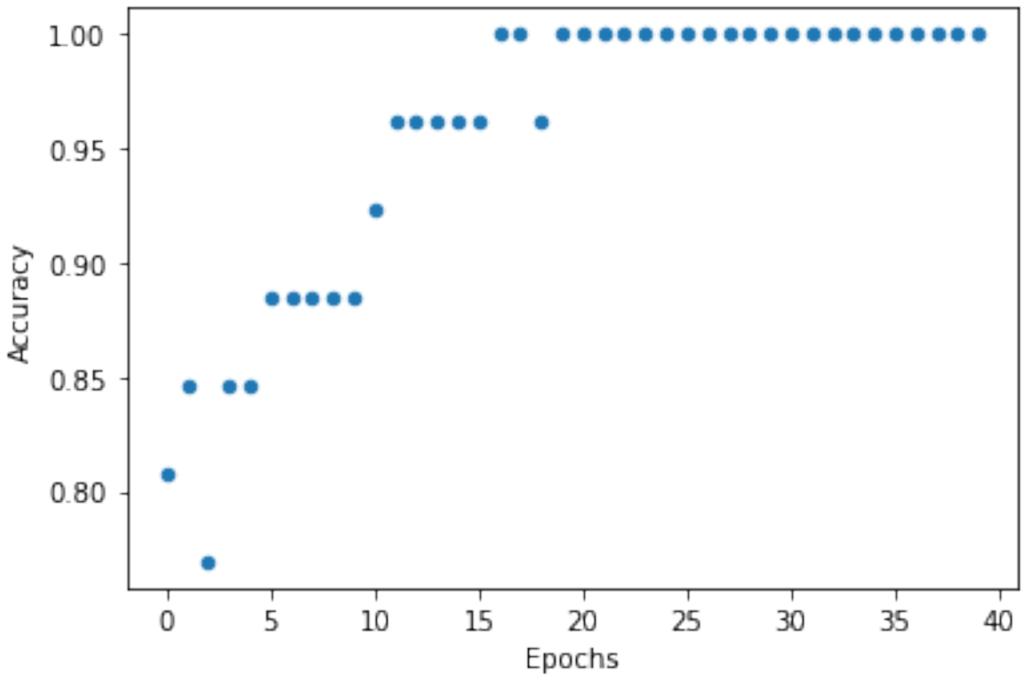
Accuracy
Epochs
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
5
10
15
20
25
30
35
40



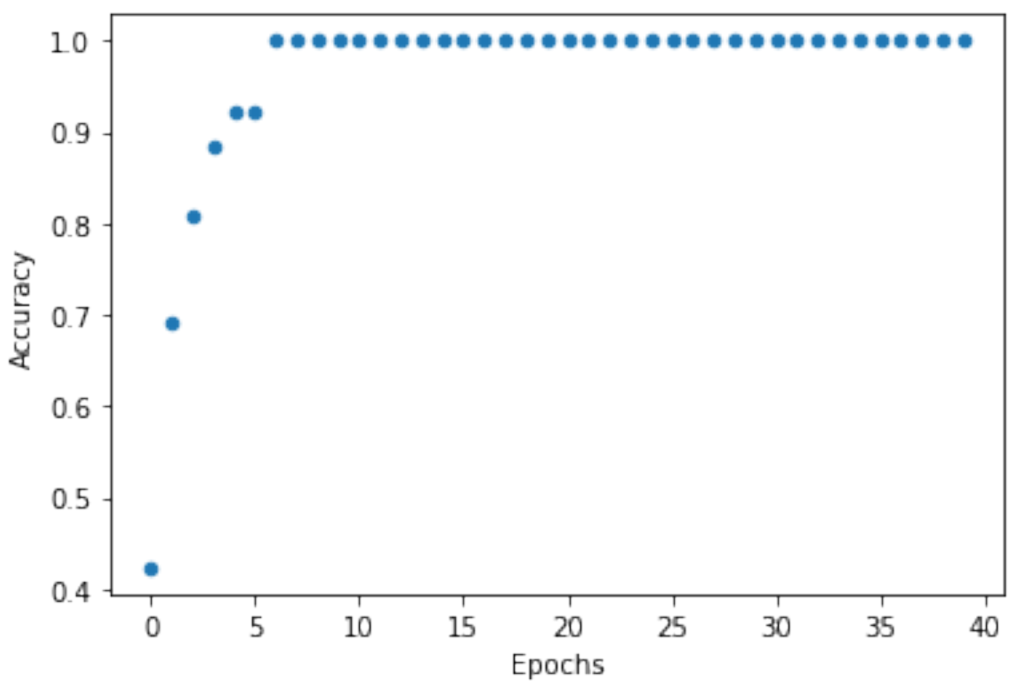
Accuracy
Epochs
1.00
0.95
0.90
0.85
0.80
0
5
10
15
20
25
30
35
40


0.85
0.80
0.75
0.70
0.65
0.60
Accuracy
0
5
10
15
20
25
30
35
40
Epochs

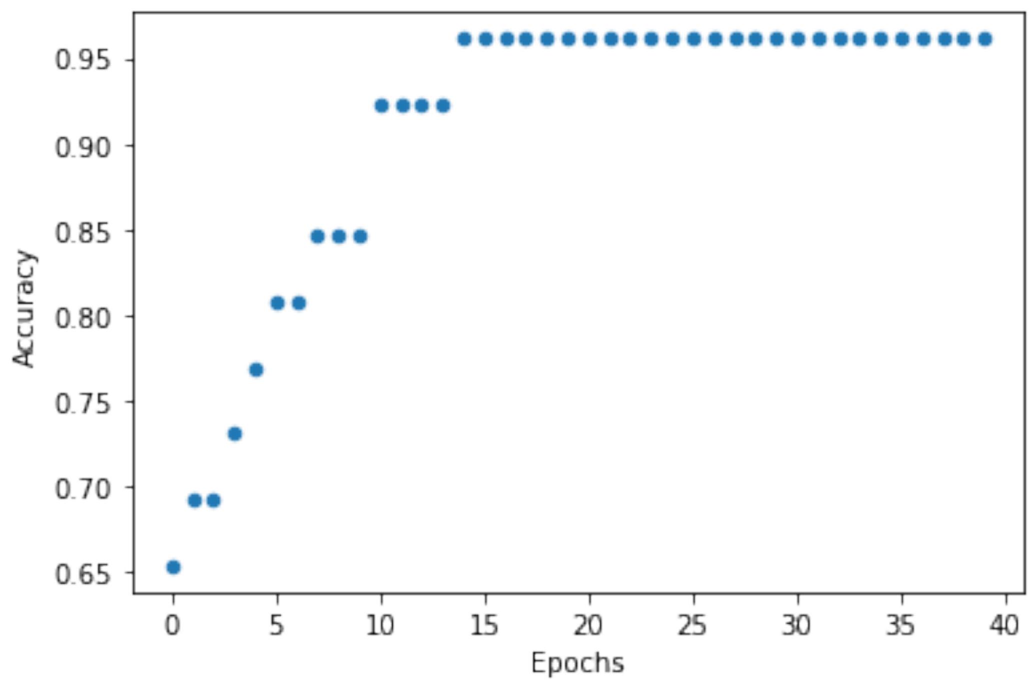

0.9
0.8
0.7
0.6
0.5
Accuracy
0
5
10
15
20
25
30
35
40
Epochs

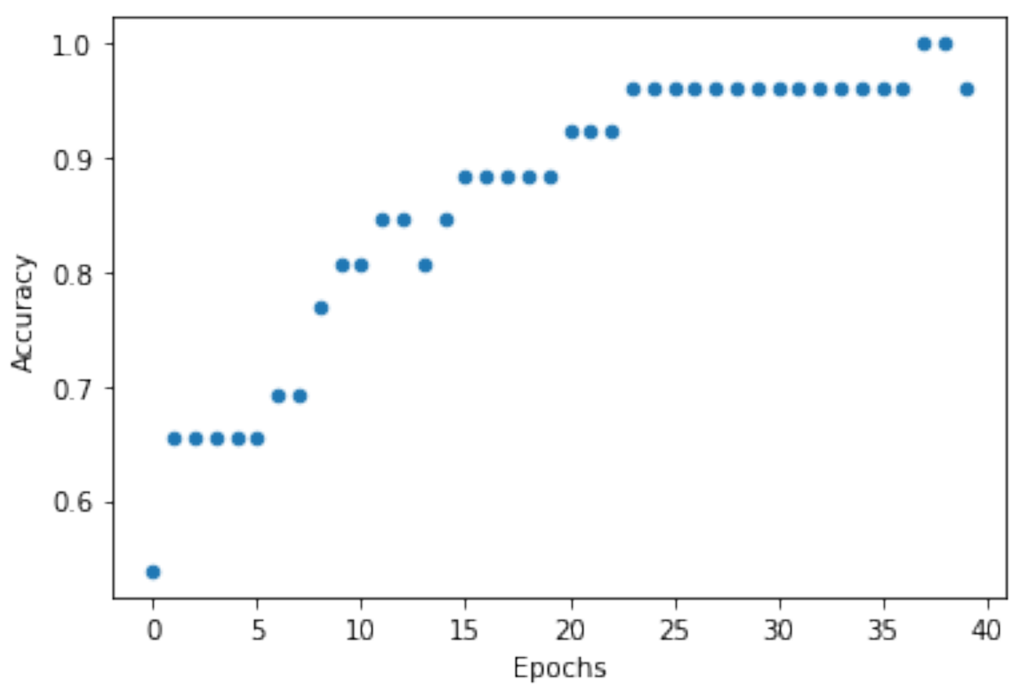


1.00
0.95
0.90
0.85
0.80
Accuracy
0
5
10
15
20
25
30
35
40
Epochs



1.0
0.9
0.8
0.7
0.6
0.5
0.4
Accuracy
0
5
10
15
20
25
30
35
40
Epochs





Now as wee can see on the graphs, that the amount of epochs variates depending on the way we select the training and the testing set.

Also we reach a perfect Accuracy on a lower epoch while using cross-validation

2

kernel_classifier

May 2, 2022

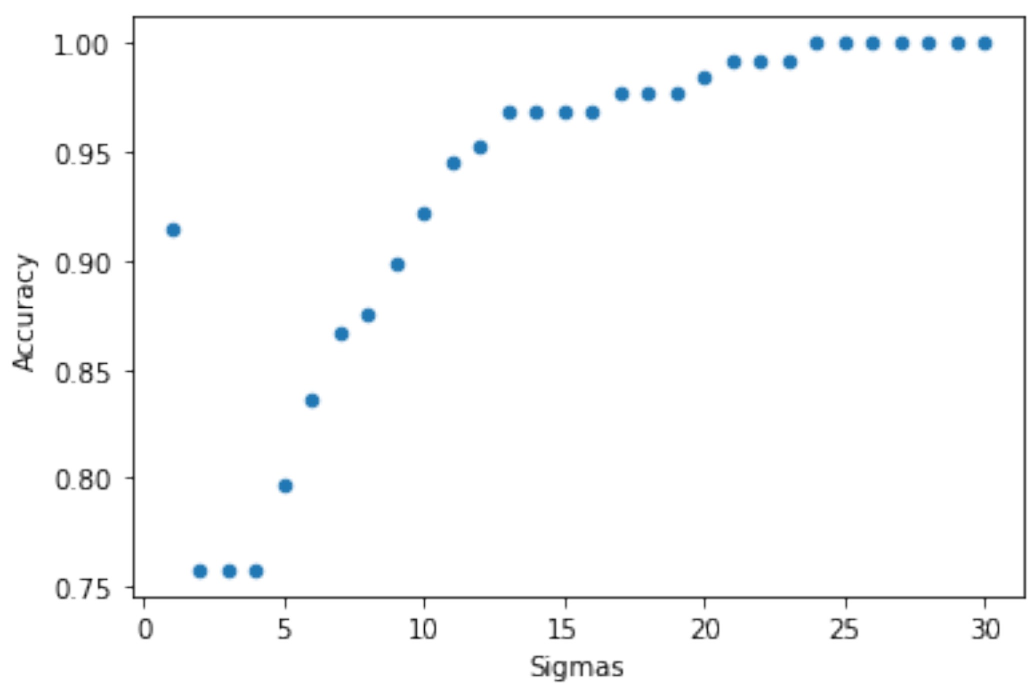
[16]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
import random
from sklearn.model_selection import train_test_split

dataset = loadmat('face_emotion_data.mat')

X, y = dataset['X'], dataset['y']
n = len(X)

y[y==-1] = 0  # use 0/1 for labels instead of -1/+1
X = np.hstack((np.ones((n,1)), X))  # append a column of ones
n, p = np.shape(X)
```

[17]:
```
# Train Classifier 1
sigmas = [i+1 for i in range(30)]
lam = 0.5

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        d = np.linalg.norm(X[i,:]-X[j,:])
        distsq[i,j]=d**2

accuracy = []
Y_hat = np.zeros((n,n))
first = True
# Predict labels
for sigma in sigmas:
    y_hat = []
    K = np.exp(-distsq/(2*sigma**2))
    alpha = np.linalg.inv(K+lam*np.identity(n))@y
    for i in range(len(X)):
```

```
        for j in range(len(X)):
            Y_hat[i][j] = np.exp(-np.linalg.norm(X[i] - X[j])**2/
↪(2*sigma**2))*alpha[i][0]
        y_hat.append(Y_hat[i].sum())
    y_aux = [1 if i > 0.5 else 0 for i in y_hat ]
    accuracy.append(accuracy_score(y_aux,y))
    if first and accuracy_score(y_aux,y) == 1.0:
        no_error_sigma = sigma
        first = False
```

[18]:
```
df = pd.DataFrame({"Sigmas":sigmas,"Accuracy":accuracy})
df.plot.scatter("Sigmas","Accuracy")
```

[18]: <AxesSubplot:xlabel='Sigmas', ylabel='Accuracy'>



[20]:
```
print("We achieve a 100% accuracy on the training set with a sigma␣
↪of",no_error_sigma)
```

We achieve a 100% accuracy on the training set with a sigma of 24

c)

[21]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state=42)
n = len(X_train)
```

```
distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        d = np.linalg.norm(X_train[i,:]-X_train[j,:])
        distsq[i,j]=d**2

accuracy = []
best_acc = None
best_sigma = None
Y_hat = np.zeros((n,n))
# Calculate best sigma
for sigma in sigmas:
    y_hat = []
    K = np.exp(-distsq/(2*sigma**2))
    alpha = np.linalg.inv(K+lam*np.identity(n))@Y_train
    for i in range(n):
        for j in range(n):
            Y_hat[i][j] = np.exp(-np.linalg.norm(X[i] - X[j])**2/
 ↪(2*sigma**2))*alpha[i][0]
        y_hat.append(Y_hat[i].sum())
    y_aux = [1 if i > 0.5 else 0 for i in y_hat ]
    current_acc = accuracy_score(y_aux,Y_train)
    accuracy.append(current_acc)
    if best_acc == None or best_acc < current_acc:
        best_acc = current_acc
        best_sigma = sigma
```

[23]:
```
n = len(X_test)
distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        d = np.linalg.norm(X_train[i,:]-X_train[j,:])
        distsq[i,j]=d**2

K = np.exp(-distsq/(2*best_sigma**2))
alpha = np.linalg.inv(K+lam*np.identity(n))@Y_test
Y_hat = np.zeros((n,n))
y_hat = []
for i in range(n):
    for j in range(n):
        Y_hat[i][j] = np.exp(-np.linalg.norm(X_test[i] - X[j])**2/
 ↪(2*sigma**2))*alpha[i][0]
    y_hat.append(Y_hat[i].sum())
y_aux = [1 if i > 0.5 else 0 for i in y_hat ]
accuracy_score(y_aux,Y_test)
```

[23]: 1.0

We can see from the accuracy_score that we got a perfect accuracy on the testing set

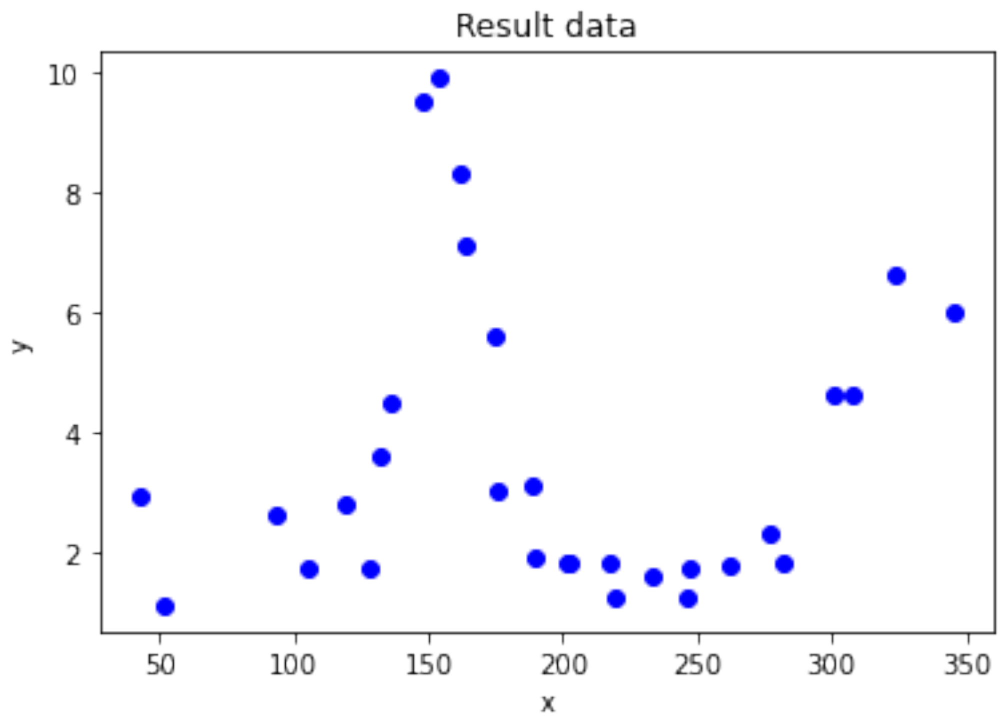
3

lake_mendota_clarity_starter

May 2, 2022

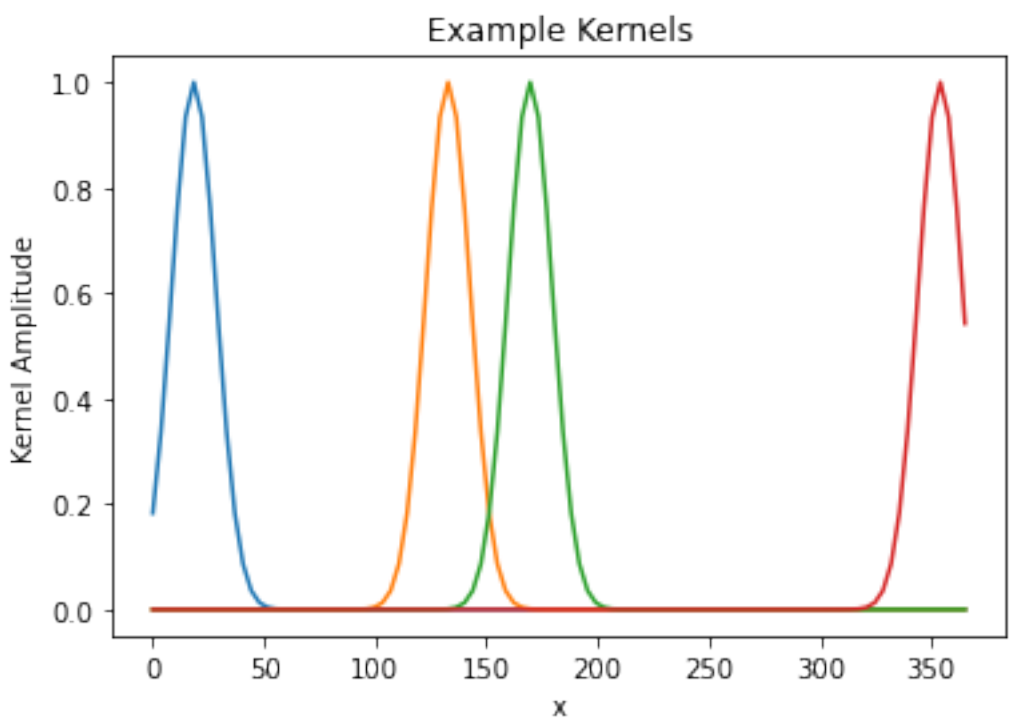
[14]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

df = pd.read_csv('mendota_secchi_depth.txt', delimiter='\t')
x = df['day_of_year']
y = df['secchi_depth']
n = len(x)
```

[15]:
```
plt.plot(x,y,'bo')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Result data')
plt.show()
```



[17]:
```
sigma = 10 #defines Gaussian kernel width
p = 100 #number of points on x-axis

# Display examples of the kernels
x_test = np.linspace(0,365,p)  # uniformly sample interval [0,365] (1 year)
j_list = [5, 36, 46, 96]  #list of indices for example kernels

Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j]= np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()
```

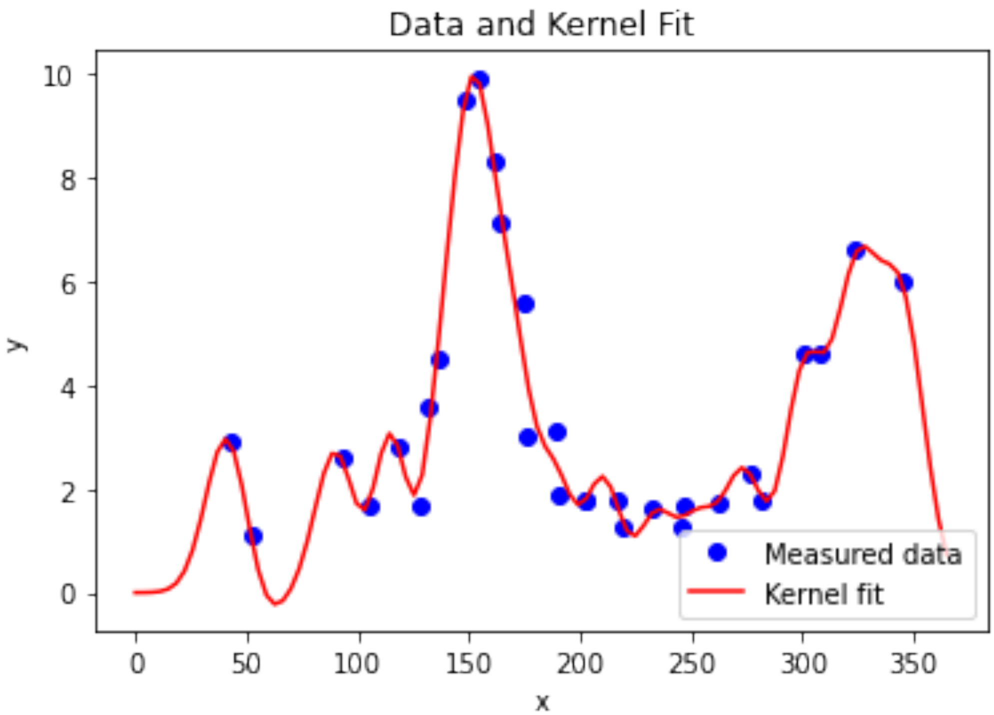
```
Sigma =  10
```



[30]:
```
# Kernel fitting to data
lam = 0.01 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@y
```

[31]:
```
# Generate smooth curve corresponding to data fit

distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test)  # noise free data for␣
 ↪comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,y,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
# plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
Sigma =  10
Lambda =  0.01
```



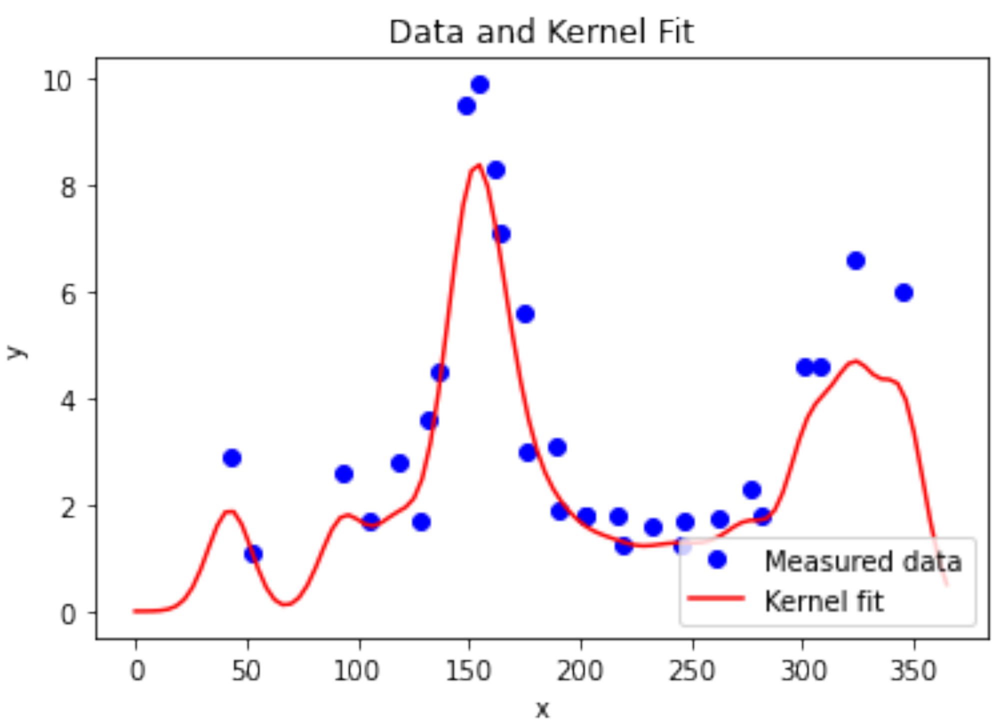
This parameters, lambda = 0.01 and sigma = 10 cleary overfit the data.

[52]:
```
# Kernel fitting to data
lam = .5 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@y
```

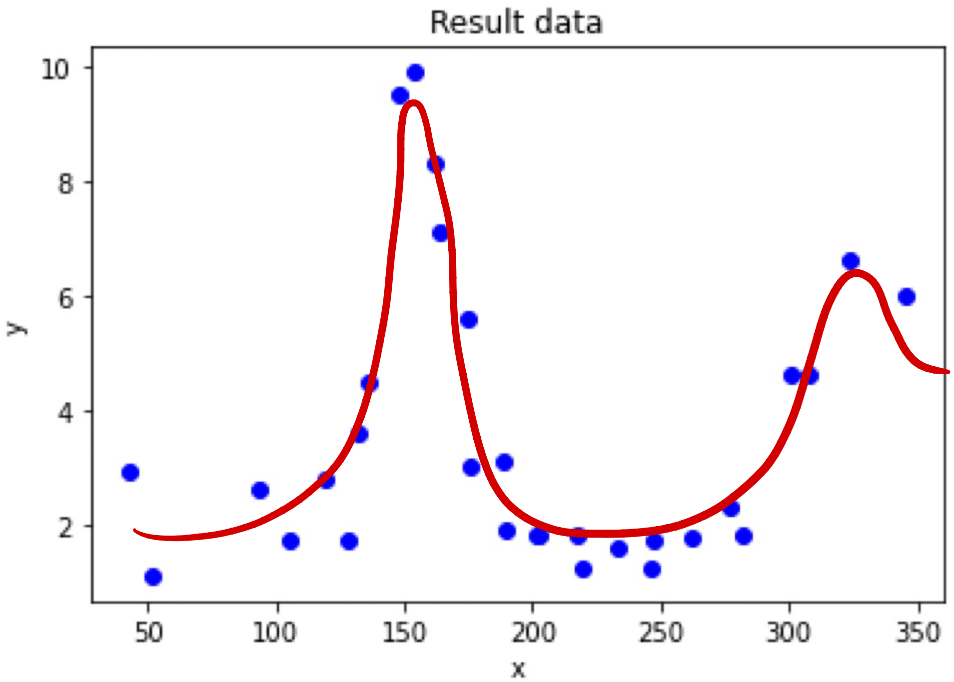
[53]:
```
# Generate smooth curve corresponding to data fit

distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha
```

```
dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test)  # noise free data for␣
 ↪comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,y,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
# plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
Sigma =  10
Lambda =  0.5
```



If we try a bigger lambda, as .5 we can see that the model does not overfit the data. However, because of the gaussian function we cannot find a perfect model that does not overfit nor underfit the data. The model that we want will be something like:



b) First we need to split our data into training and testing set. By doing this we then could use the testing set to test the accuracy of the model. Then we the training set into a second training set and a validation set. So we will train the model different parameters and test it with the validation set.