Pattern Recognition and Machine Learning

Assignment 1
Code

Group No. 19

Ranjit T
EE18B146

Sai Bandawar
EE18B150

Jay shah
EE18B158

17,March 2021

1 Task 1

1.1 Code for curve fitting for models without regularisation

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


#Taking input from csv file and taking x and y out
data=pd.read_csv("function3.csv")
data=data.to_numpy()
data=np.delete(data,0,axis=1)
x=data[:200,0] #The size is set here in x and y
y=data[:200,1]
power=2 #Setting the degree

#Calculating the Weights
A=np.empty((power+1,power+1))

for i in range(power+1):
for p in range(power+1):
A[i][p]=np.sum(np.power(x,i+p))

C=np.empty((power+1,1))

for i in range(power+1):
C[i]=np.dot(y.T,np.power(x,i))


W=np.linalg.solve(A,C)

#X values for plotting
X=np.linspace(-2,2,num=100)

#finding the outputs for differnt X in trained model
fx=[]
print(x)
for i in range(len(X)):
tp=0
for p in range(power+1):
tp+=W[p]*(X[i]**p)
fx.append(tp)

#Plotting the graph
plt.plot(X,fx,color='red',label='Polynomial Curve')
plt.scatter(x,y,marker='o',label='Dataset Points')
plt.xlabel('x')
plt.ylabel('t')
plt.text(2,34,'M=2',verticalalignment='top',horizontalalignment='right',fontsize=15)
plt.legend()
plt.savefig('M=2_N=200.png')
plt.show()
```

1.2 Code for curve fitting for models using regularisation

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


#Taking input from csv file and taking x and y out
data=pd.read_csv("function3.csv")
data=data.to_numpy()
data=np.delete(data,0,axis=1)
x=data[:10,0] #The size is set here in x and y
y=data[:10,1]
power=6 #Setting the degree
hyp=0.001 #Hyperparameter

#Calculating the Weights
A=np.empty((power+1,power+1))

for i in range(power+1):
for p in range(power+1):
A[i][p]=np.sum(np.power(x,i+p))
if i==p:
A[i][p]+=hyp #Adding hyperparameter to the diagonal


C=np.empty((power+1,1))

for i in range(power+1):
C[i]=np.dot(y.T,np.power(x,i))


W=np.linalg.solve(A,C)


#X values for plotting
X=np.linspace(-2,2,num=100)
```

```
#finding the outputs for differnt X in trained model
fx=[]
print(x)
for i in range(len(X)):
tp=0
for p in range(power+1):
tp+=W[p]*(X[i]**p)
fx.append(tp)

#Plotting the graph
plt.plot(X,fx,color='red',label='Polynomial Curve')
plt.scatter(x,y,marker='o',label='Dataset Points')
plt.xlabel('x')
plt.ylabel('t')
plt.text(2,12,'M=6, '+r'$\lambda$'+'=0.001',verticalalignment='top',horizontalalignment='right',fontsize=
plt.legend()
plt.savefig('M=6_N=10_lamb=0.001.png')
plt.show()
```

1.3 Code for calculating & plotting E_{rms} without regularisation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math


#Taking input from csv file and taking x and y out
data=pd.read_csv("function3.csv")
data=data.to_numpy()
data=np.delete(data,0,axis=1)

size=10 #Size of dataset

x=data[:size,0]
y=data[:size,1]

totsize=math.floor(size*10/7) #Finding the total size

#Validation data
x_valid=data[size+1:size+math.floor(totsize/5)+1,0]
y_valid=data[size+1:size+math.floor(totsize/5)+1,1]

#Testing data
x_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,0]
```

```
y_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,1]


powers=[2,3,6,9] #Powers to find rms for

finvals1=[]
finvals2=[]
finvals3=[]

#The below for loop calculates rms for training data
for power in powers:

#Calculating the Weights
A=np.empty((power+1,power+1))

for i in range(power+1):
for p in range(power+1):
A[i][p]=np.sum(np.power(x,i+p))

C=np.empty((power+1,1))

for i in range(power+1):
C[i]=np.dot(y.T,np.power(x,i))

W=np.linalg.solve(A,C)
fx=[]
for i in range(len(x)):
tp=0
for p in range(power+1):
tp+=W[p]*(x[i]**p)
fx.append(tp)
fx=np.array(fx)
fx=fx.reshape(fx.shape[0],)
rms=np.sqrt(np.mean((fx-y)**2,axis=0))
finvals1.append(rms)

#The below for loop calculates rms for validation data
for power in powers:
A=np.empty((power+1,power+1))

for i in range(power+1):
for p in range(power+1):
A[i][p]=np.sum(np.power(x,i+p))

C=np.empty((power+1,1))

for i in range(power+1):
C[i]=np.dot(y.T,np.power(x,i))
```

```
W=np.linalg.solve(A,C)
fx=[]
for i in range(len(x_valid)):
tp=0
for p in range(power+1):
tp+=W[p]*(x_valid[i]**p)
fx.append(tp)
fx=np.array(fx)
fx=fx.reshape(fx.shape[0],)
rms=np.sqrt(np.mean((fx-y_valid)**2,axis=0))
finvals2.append(rms)


#The below for loop calculates rms for testing data
for power in powers:
A=np.empty((power+1,power+1))

for i in range(power+1):
for p in range(power+1):
A[i][p]=np.sum(np.power(x,i+p))

C=np.empty((power+1,1))

for i in range(power+1):
C[i]=np.dot(y.T,np.power(x,i))

W=np.linalg.solve(A,C)
fx=[]
for i in range(len(x_test)):
tp=0
for p in range(power+1):
tp+=W[p]*(x_test[i]**p)
fx.append(tp)
fx=np.array(fx)
fx=fx.reshape(fx.shape[0],)
rms=np.sqrt(np.mean((fx-y_test)**2,axis=0))
finvals3.append(rms)
cols = ['Polynomial Order', 'Training Data','Validation Data','Testing Data']
df=pd.DataFrame(zip(powers,finvals1,finvals2,finvals3),columns=cols)
df[cols] = df[cols].round(4)

fig, ax = plt.subplots(2,1)
# hide axes
# ax.xaxis.set_visible(False)
# ax.yaxis.set_visible(False)
# fig.patch.set_visible(False)
ax[0].axis('off')
```

```
ax[0].axis('tight')
# plt.title('$E_{rms}$ value for N=200')
table=ax[0].table(cellText=df.values, colLabels=df.columns, loc='center')
table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1.25, 1.25)  # may help
# plt.savefig('Task1_N=10_table.png')
# plt.show()


#Plotting the graph
ax[1].plot(powers,finvals1,color='red',marker='o',label='Training Data')
ax[1].plot(powers,finvals2,color='blue',marker='o',label='Validation Data')
ax[1].plot(powers,finvals3,color='green',marker='o',label='Testing Data')
ax[1].set_xlabel('M')
ax[1].set_ylabel('$E_{rms}$')
ax[1].legend()
ax[1].set_title('$E_{rms}$ values for N=10')
# plt.savefig('Task1_N=10.png')
plt.show()
```

1.4 Code for calculating & plotting $E_{rms}$ with regularisation

```
    import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

plt.rcParams['figure.figsize'] = 9,6
# plt.rcParams['font.size'] = 18
# plt.rcParams['text.usetex'] = True
#Taking input from csv file and taking x and y out
data=pd.read_csv("function3.csv")
data=data.to_numpy()
data=np.delete(data,0,axis=1)

size=200  #Size of dataset

x=data[:size,0]
y=data[:size,1]

totsize=math.floor(size*10/7)  #Finding the total size

#Validation data
x_valid=data[size+1:size+math.floor(totsize/5)+1,0]
y_valid=data[size+1:size+math.floor(totsize/5)+1,1]
```

```
#Testing data
x_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,0]
y_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,1]

power=6 #we find regularised graph for each power

finvals1=[]
finvals2=[]
finvals3=[]



hyper=np.logspace(-3,0,10) #We vary the hyperparameter

#The below for loop calculates rms for training data
for hyp in hyper:
A=np.empty((power+1,power+1))
for i in range(power+1):
for p in range(power+1):
A[i][p]=np.sum(np.power(x,i+p))
if i==p:
A[i][p]+=hyp

C=np.empty((power+1,1))

for i in range(power+1):
C[i]=np.dot(y.T,np.power(x,i))

W=np.linalg.solve(A,C)


fx=[]
for i in range(len(x)):
tp=0
for p in range(power+1):
tp+=W[p]*(x[i]**p)
fx.append(tp)
fx=np.array(fx)
fx=fx.reshape(fx.shape[0],)
rms=np.sqrt(np.mean((fx-y)**2,axis=0))
finvals1.append(rms)

#The below for loop calculates rms for validation data
for hyp in hyper:
A=np.empty((power+1,power+1))
for i in range(power+1):
for p in range(power+1):
A[i][p]=np.sum(np.power(x,i+p))
```

```
if i==p:
A[i][p]+=hyp

C=np.empty((power+1,1))

for i in range(power+1):
C[i]=np.dot(y.T,np.power(x,i))

W=np.linalg.solve(A,C)

fx=[]
for i in range(len(x_valid)):
tp=0
for p in range(power+1):
tp+=W[p]*(x_valid[i]**p)
fx.append(tp)
fx=np.array(fx)
fx=fx.reshape(fx.shape[0],)
rms=np.sqrt(np.mean((fx-y_valid)**2,axis=0))
finvals2.append(rms)

#The below for loop calculates rms for testing data
for hyp in hyper:
A=np.empty((power+1,power+1))
for i in range(power+1):
for p in range(power+1):
A[i][p]=np.sum(np.power(x,i+p))
if i==p:
A[i][p]+=hyp

C=np.empty((power+1,1))

for i in range(power+1):
C[i]=np.dot(y.T,np.power(x,i))

W=np.linalg.solve(A,C)

fx=[]
for i in range(len(x_test)):
tp=0
for p in range(power+1):
tp+=W[p]*(x_test[i]**p)
fx.append(tp)
fx=np.array(fx)
fx=fx.reshape(fx.shape[0],)
rms=np.sqrt(np.mean((fx-y_test)**2,axis=0))
finvals3.append(rms)
```

```
cols = ['$\lambda$', 'Training Data','Validation Data','Testing Data']
df=pd.DataFrame(zip(hyper,finvals1,finvals2,finvals3),columns=cols)
df[cols] = df[cols].round(4)

fig, ax = plt.subplots(2,1)
ax[0].axis('off')
ax[0].axis('tight')

table=ax[0].table(cellText=df.values, colLabels=df.columns, loc='center')
table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1.25, 1.25)

#Plotting the graph
ax[1].semilogx(hyper,finvals1,color='red',marker='o',label='Training Data')
ax[1].semilogx(hyper,finvals2,color='blue',marker='o',label='Validation Data')
ax[1].semilogx(hyper,finvals3,color='green',marker='o',label='Testing Data')
ax[1].set_xlabel('$\lambda$')
ax[1].set_ylabel('$E_{rms}$')
ax[1].legend()
ax[1].set_title('$E_{rms}$ value for N=200 and M=6 with varying $\lambda$')
# plt.savefig('Task1_reg_N=200_M=9.png')
plt.show()
```

2 Task 2

2.1 Code for curve fitting for models without regularisation

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


def calci(X1,X2,W,power):  #function to find the output for an input
ct=0
val=0
for p in range(power+1):
tp=p
while tp>=0:
val=val+W[ct]*(X1**tp)*(X2**(p-tp))
tp=tp-1
ct=ct+1

return val



if __name__ == "__main__":

#The below lines take the csv file as input and make changes to the input dat accordingly
data=pd.read_csv("function3_2d.csv")
data=data.to_numpy()
data=np.delete(data,0,axis=1)

size=500 #Setting size of data to be trained on
x1=data[:size,0]
x2=data[:size,1]
y=data[:size,2]

power=6 #degree of the polynomial

des_mat=np.empty((size,int(((power+2)*(power+1))/2)))




for i in range(size):
ct=0
for p in range(power+1):
tp=p
while tp>=0:
des_mat[i][ct]=(x1[i]**tp)*(x2[i]**(p-tp)) #Constructing the design matrix
```

```
tp=tp-1
ct=ct+1

#Obtaining the W Matrix
A=np.dot(des_mat.T,des_mat)
B=np.dot(des_mat.T,y)

W=np.linalg.solve(A,B)


#Setting X1 and X2 values for 3-D graph plotting
X1=np.linspace(-18,18,200)
X2=np.linspace(-18,18,200)

X1,X2 = np.meshgrid(X1,X2)

fx=[]


#The below for loop generates the output for different X1's and X2's and stores them in fx
for i in range(200):
tp=[]
for p in range(200):
tp.append(calci(X1[i][p],X2[i][p],W,power))
fx.append(tp)


#Plotting is done below
fig=plt.figure()
fx=np.array(fx)
ax=plt.axes(projection='3d')


ax.plot_surface(X1,X2,fx)
ax.scatter3D(x1,x2,y,color="red",label='Dataset Points')
ax.set_title('Surface plot with M=6 N=500')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
ax.legend(bbox_to_anchor=(1.1, 0.97), bbox_transform=ax.transAxes)
plt.savefig('M=6_N=500.png')
plt.show()
```

2.2 Code for curve fitting for models using regularisation

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


def calci(X1,X2,W,power):   #function to find the output for an input
ct=0
val=0
for p in range(power+1):
tp=p
while tp>=0:
val=val+W[ct]*(X1**tp)*(X2**(p-tp))
tp=tp-1
ct=ct+1

return val



if __name__ == "__main__":

#The below lines take the csv file as input and make changes to the input dat accordingly
data=pd.read_csv("function3_2d.csv")
data=data.to_numpy()
data=np.delete(data,0,axis=1)

size=50   #Setting size of data to be trained on
x1=data[:size,0]
x2=data[:size,1]
y=data[:size,2]

power=6  #degree of the polynomial

des_mat=np.empty((size,int(((power+2)*(power+1))/2)))

hyp=1  #Hyperparameter lambda

for i in range(size):
ct=0
for p in range(power+1):
tp=p
while tp>=0:
des_mat[i][ct]=(x1[i]**tp)*(x2[i]**(p-tp))  #Constructing the design matrix
tp=tp-1
ct=ct+1


#Obtaining the W Matrix
A=np.dot(des_mat.T,des_mat)
```

```
size=A.shape[0]

for i in range(size):
for p in range(size):
if i==p:
A[i][p]+=hyp  #Lambda is addded to diagonals

B=np.dot(des_mat.T,y)

W=np.linalg.solve(A,B)

#Setting X1 and X2 values for 3-D graph plotting
X1=np.linspace(-18,18,200)
X2=np.linspace(-18,18,200)

X1,X2 = np.meshgrid(X1,X2)

fx=[]

#The below for loop generates the output for different X1's and X2's and stores them in fx
for i in range(200):
tp=[]
for p in range(200):
tp.append(calci(X1[i][p],X2[i][p],W,power))
fx.append(tp)



#Plotting is done below
fig=plt.figure()
fx=np.array(fx)
ax=plt.axes(projection='3d')

ax.plot_surface(X1,X2,fx)
ax.scatter3D(x1,x2,y,color="red",label='Dataset Points')
ax.set_title('Surface plot with M=6 N=50 and $\lambda$=1')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
ax.legend(bbox_to_anchor=(1.1, 0.92), bbox_transform=ax.transAxes)
plt.savefig('M=6_N=50_lambda=1.png')
plt.show()
```

2.3 Code for calculating & plotting E_{rms} without regularisation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

def calci(X1,X2,W,power): #function to find the output for an input
ct=0
val=0
for p in range(power+1):
tp=p
while tp>=0:
val=val+W[ct]*(X1**tp)*(X2**(p-tp))
tp=tp-1
ct=ct+1

return val



if __name__ == "__main__":

#The below lines take the csv file as input and make changes to the input dat accordingly
data=pd.read_csv("function3_2d.csv")
data=data.to_numpy()
data=np.delete(data,0,axis=1)

size=50 #Setting size of data to be trained on
x1=data[:size,0]
x2=data[:size,1]
y=data[:size,2]

totsize=math.floor(size*10/7) #Since 50 is the size of only the training data here we calculate the size

#Validation data
x1_valid=data[size+1:size+math.floor(totsize/5)+1,0]
x2_valid=data[size+1:size+math.floor(totsize/5)+1,1]
y_valid=data[size+1:size+math.floor(totsize/5)+1,2]
#Testing data
x1_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,0]
x2_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,1]
y_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,2]

powers=[2,3,6] #degree of the polynomial

finvals1=[]
finvals2=[]
```

```
finvals3=[]

for power in powers: #Traversing the powers
des_mat=np.empty((size,int(((power+2)*(power+1))/2)))

for i in range(size):
ct=0
for p in range(power+1):
tp=p
while tp>=0:
des_mat[i][ct]=(x1[i]**tp)*(x2[i]**(p-tp))
tp=tp-1
ct=ct+1

#Obtaining the W Matrix
A=np.dot(des_mat.T,des_mat)
B=np.dot(des_mat.T,y)

W=np.linalg.solve(A,B)


fx=[]

#The below loop is rms on training data
for i in range(x1.shape[0]):
fx.append(calci(x1[i],x2[i],W,power))

fx=np.array(fx)
rms=np.sqrt(np.mean((fx-y)**2,axis=0)) #Calcuating the rms value using model and target output
finvals1.append(rms)



fx=[]
#The below loop is rms on validation data
for i in range(x1_valid.shape[0]):
fx.append(calci(x1_valid[i],x2_valid[i],W,power))


fx=np.array(fx)
fx=fx.reshape(fx.shape[0],)
rms=np.sqrt(np.mean((fx-y_valid)**2,axis=0))
finvals2.append(rms)

fx=[]
#The below loop is rms on testing data
for i in range(x1_test.shape[0]):
fx.append(calci(x1_test[i],x2_test[i],W,power))
```

```
fx=np.array(fx)
fx=fx.reshape(fx.shape[0],)
rms=np.sqrt(np.mean((fx-y_test)**2,axis=0))
finvals3.append(rms)

#Plotting the power and rms values
plt.plot(powers,finvals1,color='red',marker='o',label='Training Data')
plt.plot(powers,finvals2,color='blue',marker='o',label='Validation Data')
plt.plot(powers,finvals3,color='green',marker='o',label='Testing Data')
plt.xlabel('M')
plt.ylabel('$E_{rms}$')
plt.legend()
plt.title('$E_{rms}$ value for N=50')
plt.savefig('Task2_N=50.png')
plt.show()
```

2.4 Code for calculating & plotting E_{rms} with regularisation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

def calci(X1,X2,W,power): #function to find the output for an input
ct=0
val=0
for p in range(power+1):
tp=p
while tp>=0:
val=val+W[ct]*(X1**tp)*(X2**(p-tp))
tp=tp-1
ct=ct+1

return val



if __name__ == "__main__":

data=pd.read_csv("function3_2d.csv")
data=data.to_numpy()
data=np.delete(data,0,axis=1)

size=500
```

```
x1=data[:size,0]
x2=data[:size,1]
y=data[:size,2]

totsize=math.floor(size*10/7)

x1_valid=data[size+1:size+math.floor(totsize/5)+1,0]
x2_valid=data[size+1:size+math.floor(totsize/5)+1,1]
y_valid=data[size+1:size+math.floor(totsize/5)+1,2]

x1_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,0]
x2_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,1]
y_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,2]


power=3
finvals1=[]
finvals2=[]
finvals3=[]
hyper=np.logspace(5,8,10)

#Here instead of varying power we vary the hyperparameter
for hyp in hyper:

des_mat=np.empty((size,int(((power+2)*(power+1))/2)))
for i in range(size):
ct=0
for p in range(power+1):
tp=p
while tp>=0:
des_mat[i][ct]=(x1[i]**tp)*(x2[i]**(p-tp))
tp=tp-1
ct=ct+1



A=np.dot(des_mat.T,des_mat)

tp=A.shape[0]

for i in range(tp):
for p in range(tp):
if i==p:
A[i][p]+=hyp

B=np.dot(des_mat.T,y)

W=np.linalg.solve(A,B)
```

```
fx=[]


for i in range(x1.shape[0]):
fx.append(calci(x1[i],x2[i],W,power))

fx=np.array(fx)
rms=np.sqrt(np.mean((fx-y)**2,axis=0))
finvals1.append(rms)



fx=[]

for i in range(x1_valid.shape[0]):
fx.append(calci(x1_valid[i],x2_valid[i],W,power))


fx=np.array(fx)
fx=fx.reshape(fx.shape[0],)
rms=np.sqrt(np.mean((fx-y_valid)**2,axis=0))
finvals2.append(rms)

fx=[]

for i in range(x1_test.shape[0]):
fx.append(calci(x1_test[i],x2_test[i],W,power))

fx=np.array(fx)
fx=fx.reshape(fx.shape[0],)
rms=np.sqrt(np.mean((fx-y_test)**2,axis=0))
finvals3.append(rms)


#Plotting the power and rms values
hyper=np.log10(hyper)
plt.plot(hyper,finvals1,color='red',marker='o',label='Training Data')
plt.plot(hyper,finvals2,color='blue',marker='o',label='Validation Data')
plt.plot(hyper,finvals3,color='green',marker='o',label='Testing Data')
plt.xlabel('log$\lambda$')
plt.ylabel('$E_{rms}$')
plt.legend()
plt.title('$E_{rms}$ value for N=500 and M=3 with varying $\lambda$')
plt.savefig('Task2_reg_N=500_M=3.png')
plt.show()
```

2.5 code for scatter plot

```
    import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

def calci(X1,X2,W,power):  #function to find the output for an input
ct=0
val=0
for p in range(power+1):
tp=p
while tp>=0:
val=val+W[ct]*(X1**tp)*(X2**(p-tp))
tp=tp-1
ct=ct+1

return val

if __name__ == "__main__":

#The below code is same as rms
data=pd.read_csv("function3_2d.csv")
data=data.to_numpy()
data=np.delete(data,0,axis=1)

size=500
x1=data[:size,0]
x2=data[:size,1]
y=data[:size,2]

totsize=math.floor(size*10/7)

x1_valid=data[size+1:size+math.floor(totsize/5)+1,0]
x2_valid=data[size+1:size+math.floor(totsize/5)+1,1]
y_valid=data[size+1:size+math.floor(totsize/5)+1,2]

x1_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,0]
x2_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,1]
y_test=data[size+math.floor(totsize/5)+1:size+math.floor(totsize/5)+math.floor(totsize/10)+1,2]

power=6
finvals1=[]
finvals2=[]
finvals3=[]
hyp=0  #Vary this according to best model

des_mat=np.empty((size,int(((power+2)*(power+1))/2)))
```

```
for i in range(size):
ct=0
for p in range(power+1):
tp=p
while tp>=0:
des_mat[i][ct]=(x1[i]**tp)*(x2[i]**(p-tp))
tp=tp-1
ct=ct+1



A=np.dot(des_mat.T,des_mat)

tp=A.shape[0]

for i in range(tp):
for p in range(tp):
if i==p:
A[i][p]+=hyp

B=np.dot(des_mat.T,y)

W=np.linalg.solve(A,B)


fx=[]

for i in range(x1.shape[0]): #The x1 could be varied to x1_test for testing data
fx.append(calci(x1[i],x2[i],W,power))

fx=np.array(fx)

rms=np.sqrt(np.mean((fx-y)**2,axis=0))
finvals1.append(rms)

#Plotting the scatter values and the straight line
plt.scatter(y,fx,marker='o',color='red',label='Estimated Outputs')
plt.plot(y,y,label='Actual Outputs')

plt.xlabel('t$_n$')
plt.ylabel('Model Output')
plt.legend()
plt.title('Training Data with M=6 N=500 and $\lambda$=0')
plt.text(170,12,'$E_{rms}$=%f'%(rms),fontsize=11)
plt.savefig('Scatter_3_train.png')
plt.show()
```

3 Task 3

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

data=pd.read_csv("1_bias_clean.csv")
data=data.dropna() # dropping rows with Nan

data=data.to_numpy()
size=int(np.ceil(0.7*data.shape[0])) # 70% as Training data

X_train=data[:size]
t1=X_train[:,-2]
t2=X_train[:,-1]

X=X_train[:,0:-2]

k=49 # number of clusters

#  Using scikit-learn to perform K-Means clustering
# from sklearn.cluster import KMeans

# Specify the number of clusters (3) and fit the data X
# kmeans = KMeans(n_clusters=k, random_state=0).fit(X)

# Get the cluster centroids

# means=kmeans.cluster_centers_

def Kmeans(k):
    ''' Kmeans Clustering Algo implemented from scratch
        Input= no. of n_clusters
        Output= means (centriods) as row vectors stacked on top of eachother
    '''
    means = X[np.random.choice(range(len(X)), k, replace=False)]

    z_prev=np.full([1, len(X)], None)

    convergence=True
    count=0
    while(convergence):

        b= np.full([1, len(X)], None)
        for i in range(k):
            dist=(np.linalg.norm(X-means[i],axis=1))**2 # Calculates the Eucledian Distance
            b=np.vstack((b,dist))
        b=np.delete(b,0,axis=0)
        # Here b is an array in which each column represents the distance of a point from each of
        # the k clusters

        z=np.argmin(b, axis=0) # finds the minimum of each column
        # z a colum vector containing the cluster to which each data row belongs
        if (z == z_prev).all():
            break # condition for convergence
        z_prev=np.copy(z)


        Z=np.zeros((len(X),k)) # form a Z ( indicator ) matrix

        for i in range(len(X)):
            Z[i,z[i]]=1

        sum=np.sum(Z,axis=0) # sum conatins the number of elements belonging
                             # to each cluster

        SUM=np.matmul(X.T,Z)
        for i in range(k):
            SUM[:,i]=SUM[:,i]/sum[i]

        means=SUM.T
        count=count+1

    return means

means=Kmeans(k)

#Phi function transforms input variable vector x into the basis function evaluated at x
def Q_i(x, mu_i):
    return math.exp(-1*((np.linalg.norm(x-mu_i))**2)/(sigma**2))

sigma = 1200
l=0

def phi_matrix(X,means,k): #design matrix
    Q=np.zeros((len(X),k+1))
    Q[:,0]=np.ones(len(X))
    for i in range(len(X)):
        for j in range(1,k+1):
            Q[i][j]=Q_i(X[i], means[j-1])
    return Q

Q=phi_matrix(X,means,k)
```

```
Q_inv=np.matmul(np.linalg.inv(np.matmul(np.transpose(Q),Q)+l*np.identity(len(Q[1]))),np.transpose(Q))

#Code for design matrix and calculations for Tikhonov

# def phi_matrix(X,means,k):
#     Q=np.zeros((len(X),k))
# #     Q[:,0]=np.ones(len(X))
#     for i in range(len(X)):
#         for j in range(k):
#             Q[i][j]=Q_i(X[i], means[j])
#     return Q
#
# Q=phi_matrix(X,means,k)
#
#
# # Q_inv=np.linalg.pinv(Q)
#
# phi_tilde=np.zeros((k,k))
# # phi_tilde[0,0]=1
#
# for i in range(k):
#     for j in range(k):
#         phi_tilde[i][j]=Q_i(means[i], means[j])
# Q_inv=np.matmul(np.linalg.inv(np.matmul(np.transpose(Q),Q)+l*phi_tilde),np.transpose(Q))


W1=np.matmul(Q_inv,np.transpose(t1))
W2=np.matmul(Q_inv,np.transpose(t2))

def y_estimate(Q,w):
    return np.matmul(Q,w)

y1_estimate=y_estimate(Q,W1)
y2_estimate=y_estimate(Q,W2)

se1= (y1_estimate-t1)**2
se2=(y2_estimate-t2)**2


rms1=np.sqrt(np.mean(se1,axis=0))
rms2=np.sqrt(np.mean(se2,axis=0))
print("\nTraining Data\n")
print(rms1,rms2)

plt.rcParams['figure.figsize'] = 9,6
plt.rcParams['font.size'] = 15
plt.rcParams['text.usetex'] = True
```

```
fig1=plt.figure(1)
plt.scatter(t1,y1_estimate,marker='o',color='r')
plt.plot(t1,t1)
plt.xlabel('$t_1$ (i.e $2^{nd}$ last column)',fontsize=18)
plt.ylabel('$t_{1estimate}$',fontsize=18)
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using training data with {k+1} clusters and $\sigma$= {sigma} (with L2 regularization $\lambd
plt.text(37.5,22.5,"$E_{rms}$=%f"%(rms1),verticalalignment='top',horizontalalignment='right',fontsize=18)
plt.savefig('1.5000.png')
plt.show()

fig2=plt.figure(2)
plt.scatter(t2,y2_estimate,marker='o',color='r')
plt.plot(t2,t2)
plt.xlabel('$t_2$ (i.e last column)',fontsize=18)
plt.ylabel('$t_{2estimate}$',fontsize=18)
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using training data with {k+1} clusters and $\sigma$= {sigma} (with L2 regularization $\lambd
plt.text(30,15,"$E_{rms}$=%f"%(rms2),verticalalignment='top',horizontalalignment='right',fontsize=18)
plt.savefig('2.5000.png')
plt.show()


###########################################################################################################
# Validation Dataset
fin=data.shape[0]
fin=int(np.ceil(0.9*fin))
X_val=data[size+1:fin]
X_valid=X_val[:,0:-2]
t1_val=X_val[:,-2]
t2_val=X_val[:,-1]

Q_val=phi_matrix(X_valid,means,k)

y1_estimate_val=y_estimate(Q_val,W1)
y2_estimate_val=y_estimate(Q_val,W2)


se1_val= (y1_estimate_val-t1_val)**2
se2_val=(y2_estimate_val-t2_val)**2

rms1_val=np.sqrt(np.mean(se1_val,axis=0))
rms2_val=np.sqrt(np.mean(se2_val,axis=0))
print("\nValidation Data\n")
print(rms1_val,rms2_val)

fig3=plt.figure(3)
```

```
plt.scatter(t1_val,y1_estimate_val,marker='o',color='r')
plt.plot(t1_val,t1_val)
plt.xlabel('$t_1$ (i.e $2^{nd}$ last column)',fontsize=18)
plt.ylabel('$t_{1estimate}$',fontsize=18)
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using validation data with {k+1} clusters and $\sigma$= {sigma} (with L2 regularization $\lam
plt.text(38.5,20.5,"$E_{rms}$=%f"%(rms1_val),verticalalignment='top',horizontalalignment='right',fontsize
plt.savefig('3.5000.png')
plt.show()

fig4=plt.figure(4)
plt.scatter(t2_val,y2_estimate_val,marker='o',color='r')
plt.plot(t2_val,t2_val)
plt.xlabel('$t_2$ (i.e last column)',fontsize=18)
plt.ylabel('$t_{2estimate}$',fontsize=18)
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using validation data with {k+1} clusters and $\sigma$= {sigma} (with L2 regularization $\lam
plt.text(27.5,14,"$E_{rms}$=%f"%(rms2_val),verticalalignment='top',horizontalalignment='right',fontsize=1
plt.savefig('4.5000.png')
plt.show()

##########################################################################################################
# Test Dataset
X_test=data[fin+1:]
X_t=X_test[:,0:-2]
t1_t=X_test[:,-2]
t2_t=X_test[:,-1]

Q_t=phi_matrix(X_t,means,k)

y1_estimate_t=y_estimate(Q_t,W1)
y2_estimate_t=y_estimate(Q_t,W2)


se1_t= (y1_estimate_t-t1_t)**2
se2_t=(y2_estimate_t-t2_t)**2

rms1_t=np.sqrt(np.mean(se1_t,axis=0))
rms2_t=np.sqrt(np.mean(se2_t,axis=0))

print("\nTest Data\n")
print(rms1_t,rms2_t)

fig5=plt.figure(5)
plt.scatter(t1_t,y1_estimate_t,marker='o',color='r')
plt.plot(t1_t,t1_t)
plt.xlabel('$t_1$ (i.e $2^{nd}$ last column)',fontsize=18)
plt.ylabel('$t_{1estimate}$',fontsize=18)
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using test data with {k+1} clusters and $\sigma$= {sigma} (with L2 regularization $\lambda$=
plt.text(37,22,"$E_{rms}$=%f"%(rms1_t),verticalalignment='top',horizontalalignment='right',fontsize=18)
plt.savefig('5.5000.png')
plt.show()

fig6=plt.figure(6)
plt.scatter(t2_t,y2_estimate_t,marker='o',color='r')
plt.plot(t2_t,t2_t)
plt.xlabel('$t_2$ (i.e last column)',fontsize=18)
plt.ylabel('$t_{2estimate}$',fontsize=18)
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using test data with {k+1} clusters and $\sigma$= {sigma} (with L2 regularization $\lambda$=
plt.text(28.5,14,"$E_{rms}$=%f"%(rms2_t),verticalalignment='top',horizontalalignment='right',fontsize=18)
plt.savefig('6.5000.png')
plt.show()
```

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt

        data=pd.read_csv("1_bias_clean.csv")
```

```
In [2]: data=data.dropna()
```

```
In [3]: data=data.to_numpy()
        size=int(np.ceil(0.7*data.shape[0]))
        X_train=data[:size]
```

```
In [4]: data.shape
```

```
Out[4]: (7588, 24)
```

```
In [5]: t1=X_train[:,-2]
        t2=X_train[:,-1]
```

```
In [6]: X=X_train[:,0:-2]
        X.shape
```

```
Out[6]: (5312, 22)
```

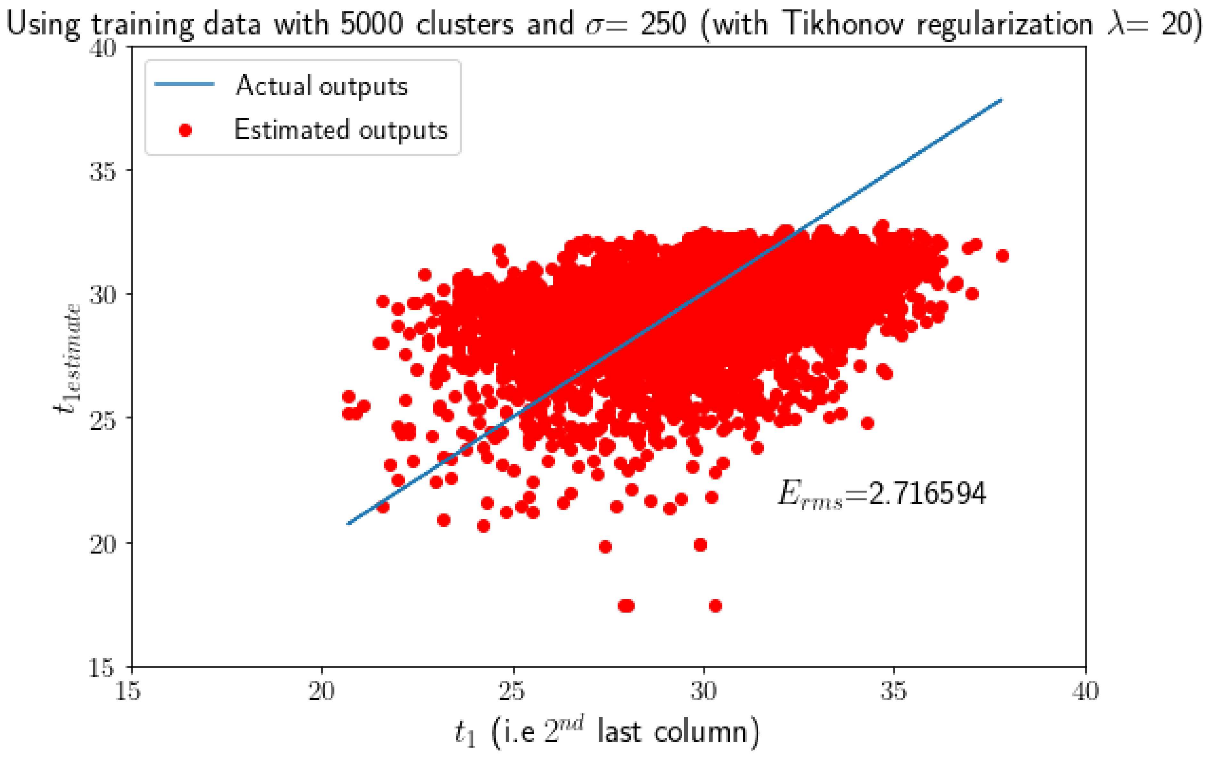
```
In [7]: k=5000 # number of clusters

        # Using scikit-learn to perform K-Means clustering
        from sklearn.cluster import KMeans

        # Specify the number of clusters (3) and fit the data X
        kmeans = KMeans(n_clusters=k, random_state=0).fit(X)

        # Get the cluster centroids

        means=kmeans.cluster_centers_
        # np.random.seed(0)
        # means = X[np.random.choice(range(len(X)), k, replace=False)]

        # z_prev=np.full([1, len(X)], None)

        # #Code perfoms clustering

        # convergence=True
        # count=0
        # while(convergence):

        #     b= np.full([1, len(X)], None)
        #     for i in range(k):
        #         dist=(np.linalg.norm(X-means[i],axis=1))**2
        #         b=np.vstack((b,dist))


        #     b=np.delete(b,0,axis=0)

        #     z=np.argmin(b, axis=0)

        #     if (z == z_prev).all():
        #         break
        #     z_prev=np.copy(z)
#     Z=np.zeros((len(X),k))

#     for i in range(len(X)):
#         Z[i,z[i]]=1

#     sum=np.sum(Z,axis=0)

#     SUM=np.matmul(X.T,Z)
#     for i in range(k):
#         SUM[:,i]=SUM[:,i]/sum[i]

#     means=SUM.T
#     count=count+1
```

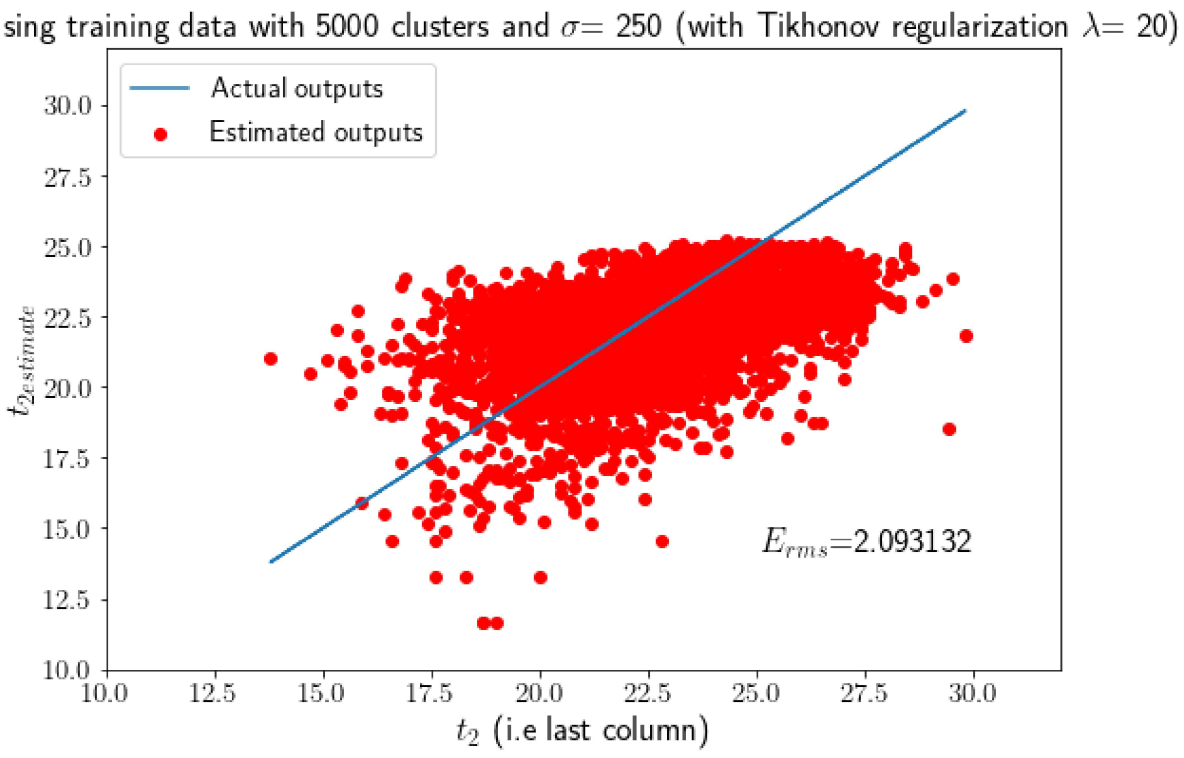
In [10]:
```
import math
#Phi function transforms input variable vector x into the basis function evaluated a
def Q_i(x, mu_i):
    return math.exp(-1*((np.linalg.norm(x-mu_i))**2)/(sigma**2))

sigma =250
l=20

def phi_matrix(X,means,k):
    Q=np.zeros((len(X),k))
#    Q[:,0]=np.ones(len(X))
    for i in range(len(X)):
        for j in range(k):
            Q[i][j]=Q_i(X[i], means[j])
    return Q

Q=phi_matrix(X,means,k)


# Q_inv=np.linalg.pinv(Q)

phi_tilde=np.zeros((k,k))
# phi_tilde[0,0]=1

for i in range(k):
    for j in range(k):
        phi_tilde[i][j]=Q_i(means[i], means[j])
Q_inv=np.matmul(np.linalg.inv(np.matmul(np.transpose(Q),Q)+l*phi_tilde),np.transpose


W1=np.matmul(Q_inv,np.transpose(t1))
W2=np.matmul(Q_inv,np.transpose(t2))

def y_estimate(Q,w):
    return np.matmul(Q,w)

y1_estimate=y_estimate(Q,W1)
y2_estimate=y_estimate(Q,W2)

se1= (y1_estimate-t1)**2
se2=(y2_estimate-t2)**2


rms1=np.sqrt(np.mean(se1,axis=0))
rms2=np.sqrt(np.mean(se2,axis=0))

print(rms1,rms2)
```

```
2.71659379024003 2.0931316261265143
```

In [11]:
```
%matplotlib inline
```

```
plt.rcParams['figure.figsize'] = 9,6
plt.rcParams['font.size'] = 15
plt.rcParams['text.usetex'] = True


fig1=plt.figure(1)
plt.scatter(t1,y1_estimate,marker='o',color='r')
plt.plot(t1,t1)
plt.xlabel('$t_1$ (i.e $2^{nd}$ last column)',fontsize=18)
plt.ylabel('$t_{1estimate}$',fontsize=18)
plt.xlim([15,40])
plt.ylim([15,40])
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using training data with {k} clusters and $\sigma$= {sigma} (with Tikhon
plt.text(37.5,22.5,"$E_{rms}$=%f"%(rms1),verticalalignment='top',horizontalalignment
plt.savefig('1.50005.png')
plt.show()

fig2=plt.figure(2)
plt.scatter(t2,y2_estimate,marker='o',color='r')
plt.plot(t2,t2)
plt.xlim([10,32])
plt.ylim([10,32])
plt.xlabel('$t_2$ (i.e last column)',fontsize=18)
plt.ylabel('$t_{2estimate}$',fontsize=18)
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using training data with {k} clusters and $\sigma$= {sigma} (with Tikhon
plt.text(30,15,"$E_{rms}$=%f"%(rms2),verticalalignment='top',horizontalalignment='ri
plt.savefig('2.50005.png')
plt.show()
```





In [12]:
```
fin=data.shape[0]
fin
```

Out[12]:
```
7588
```

In [13]:
```
fin=int(np.ceil(0.9*fin))
fin
```

Out[13]:
```
6830
```

In [14]:
```
X_val=data[size+1:fin]
```

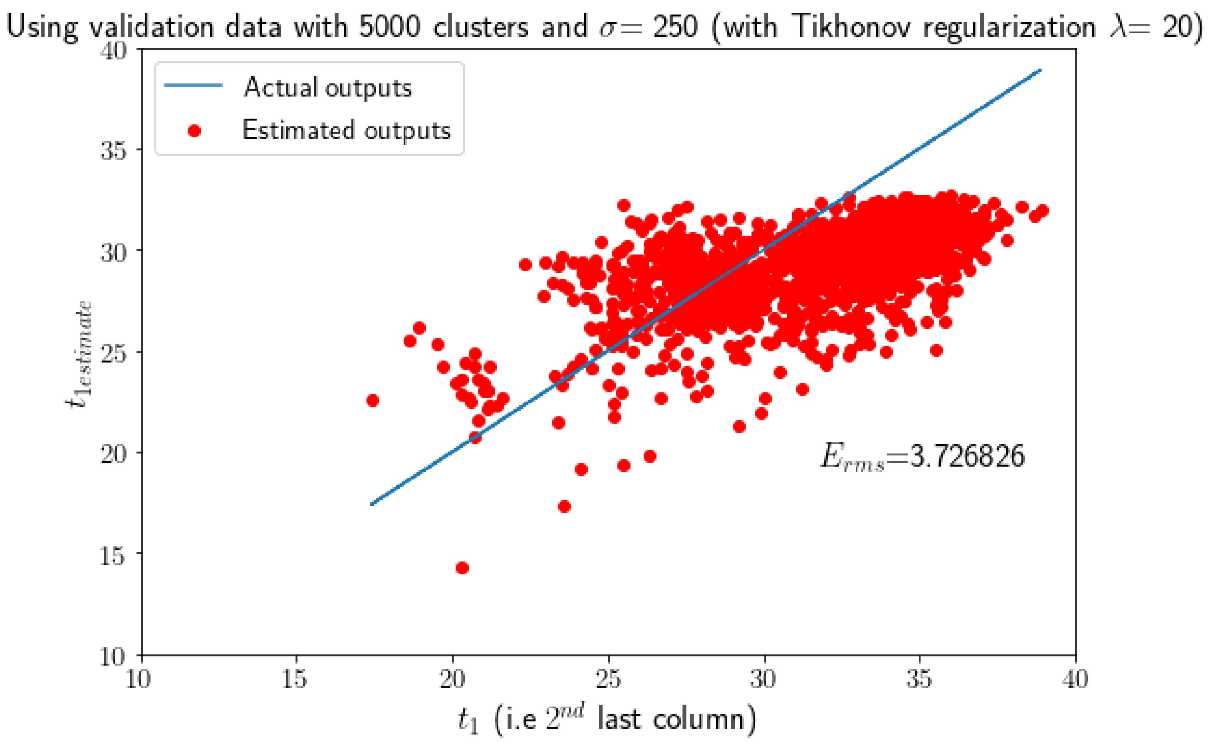
In [15]:
```
X_valid=X_val[:,0:-2]
```

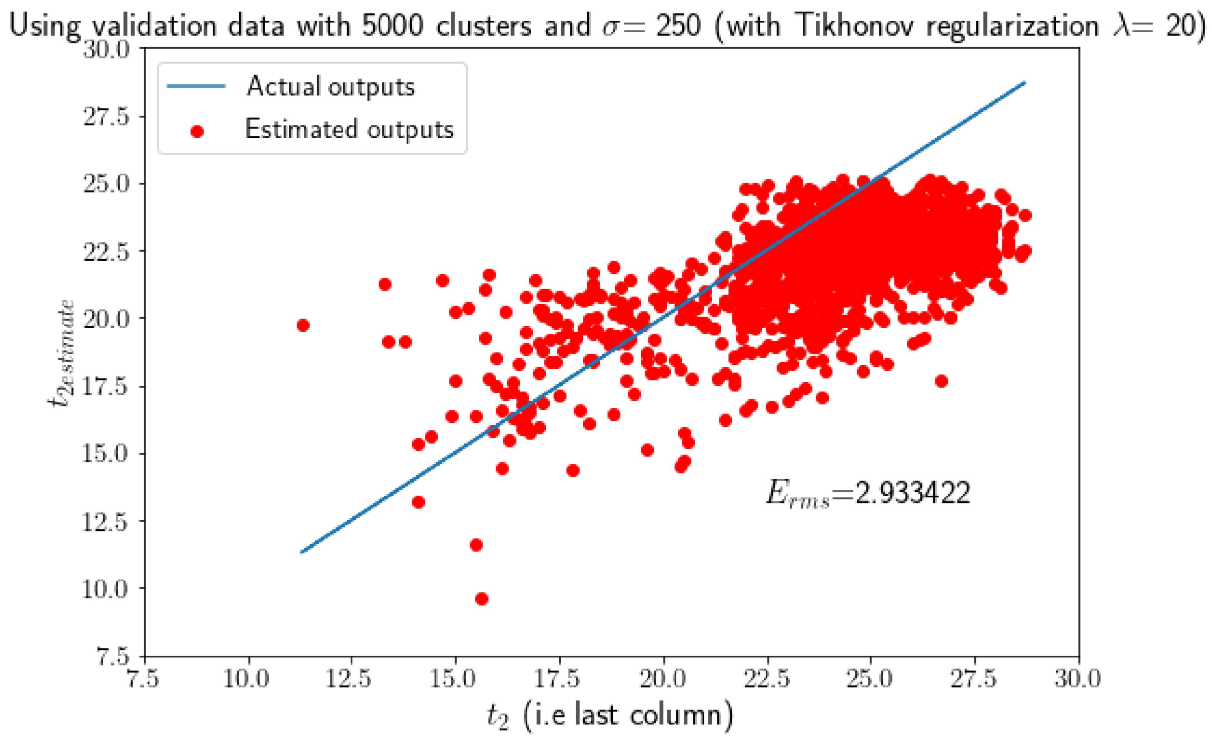
In [16]:
```
t1_val=X_val[:,-2]
t2_val=X_val[:,-1]
```

In [17]:
```
Q_val=phi_matrix(X_valid,means,k)

y1_estimate_val=y_estimate(Q_val,W1)
y2_estimate_val=y_estimate(Q_val,W2)


se1_val= (y1_estimate_val-t1_val)**2
se2_val=(y2_estimate_val-t2_val)**2

rms1_val=np.sqrt(np.mean(se1_val,axis=0))
rms2_val=np.sqrt(np.mean(se2_val,axis=0))

print(rms1_val,rms2_val)
```

```
3.7268255477776693 2.933421683883224
```

In [18]:
```
fig3=plt.figure(3)
plt.scatter(t1_val,y1_estimate_val,marker='o',color='r')
plt.plot(t1_val,t1_val)
plt.xlabel('$t_1$ (i.e $2^{nd}$ last column)',fontsize=18)
plt.ylabel('$t_{1estimate}$',fontsize=18)
plt.xlim([10,40])
plt.ylim([10,40])
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using validation data with {k} clusters and $\sigma$= {sigma} (with Tikh
plt.text(38.5,20.5,"$E_{rms}$=%f"%(rms1_val),verticalalignment='top',horizontalalign
plt.savefig('3.50005.png')
plt.show()

fig4=plt.figure(4)
plt.scatter(t2_val,y2_estimate_val,marker='o',color='r')
plt.plot(t2_val,t2_val)
plt.xlabel('$t_2$ (i.e last column)',fontsize=18)
plt.ylabel('$t_{2estimate}$',fontsize=18)
plt.xlim([7.5,30])
plt.ylim([7.5,30])
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using validation data with {k} clusters and $\sigma$= {sigma} (with Tikh
plt.text(27.5,14,"$E_{rms}$=%f"%(rms2_val),verticalalignment='top',horizontalalignme
plt.savefig('4.3000.png')
plt.show()
```



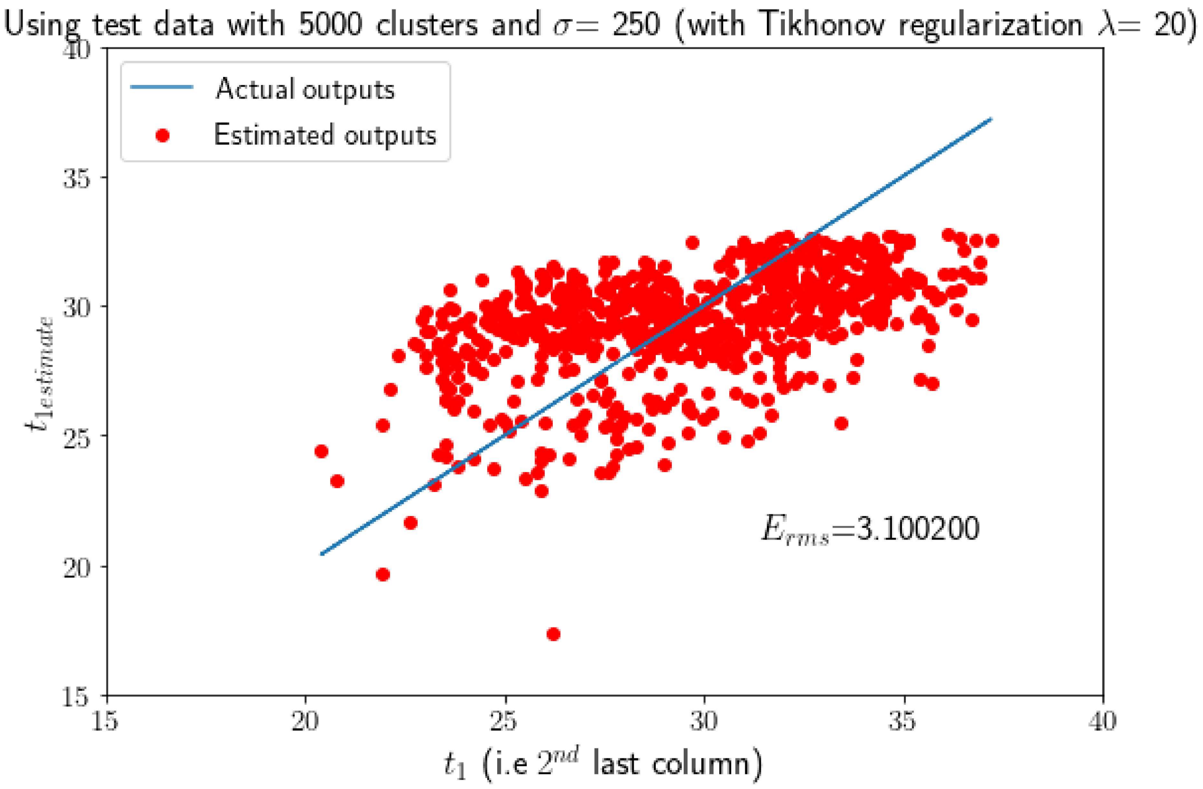


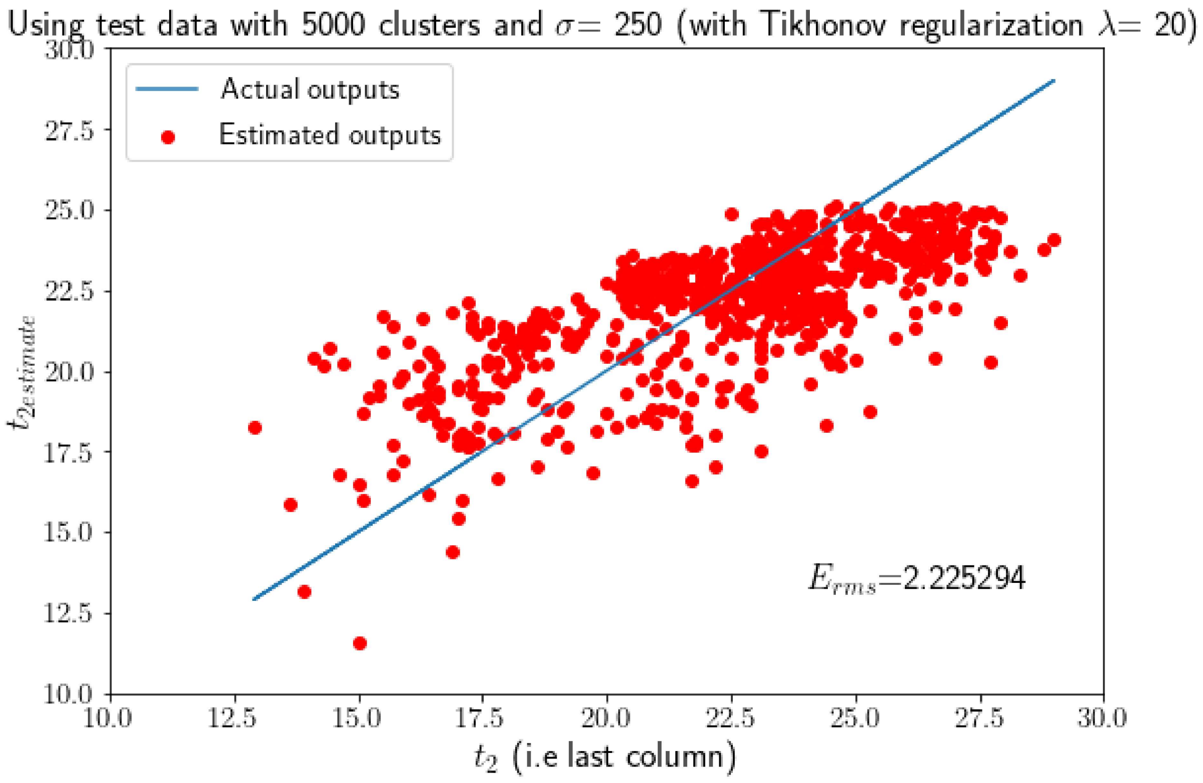
```
In [19]: X_test=data[fin+1:]
```

```
In [20]: X_t=X_test[:,0:-2]
         t1_t=X_test[:,-2]
         t2_t=X_test[:,-1]
```

```
In [21]: Q_t=phi_matrix(X_t,means,k)

         y1_estimate_t=y_estimate(Q_t,W1)
         y2_estimate_t=y_estimate(Q_t,W2)


         se1_t= (y1_estimate_t-t1_t)**2
         se2_t=(y2_estimate_t-t2_t)**2

         rms1_t=np.sqrt(np.mean(se1_t,axis=0))
         rms2_t=np.sqrt(np.mean(se2_t,axis=0))

         print(rms1_t,rms2_t)

         3.1001997873441742 2.225293708752691
```

```
In [22]: fig5=plt.figure(5)
         plt.scatter(t1_t,y1_estimate_t,marker='o',color='r')
         plt.plot(t1_t,t1_t)
         plt.xlabel('$t_1$ (i.e $2^{nd}$ last column)',fontsize=18)
         plt.ylabel('$t_{1estimate}$',fontsize=18)
         plt.xlim([15,40])
         plt.ylim([15,40])
         plt.legend(["Actual outputs" ,"Estimated outputs"])
         plt.title(f"Using test data with {k} clusters and $\sigma$= {sigma} (with Tikhonov r
         plt.text(37,22,"$E_{rms}$=%f"%(rms1_t),verticalalignment='top',horizontalalignment='
         plt.savefig('5.50005.png')
         plt.show()

         fig6=plt.figure(6)
         plt.scatter(t2_t,y2_estimate_t,marker='o',color='r')
         plt.plot(t2_t,t2_t)
         plt.xlim([10,30])
         plt.ylim([10,30])
plt.xlabel('$t_2$ (i.e last column)',fontsize=18)
plt.ylabel('$t_{2estimate}$',fontsize=18)
plt.legend(["Actual outputs" ,"Estimated outputs"])
plt.title(f"Using test data with {k} clusters and $\sigma$= {sigma} (with Tikhonov r
plt.text(28.5,14,"$E_{rms}$=%f"%(rms2_t),verticalalignment='top',horizontalalignment
plt.savefig('6.50005.png')
plt.show()
```


Using test data with 5000 clusters and σ= 250 (with Tikhonov regularization λ= 20)
Actual outputs
● Estimated outputs
E_{rms}=3.100200
$t_{1estimate}$
t_1 (i.e 2^{nd} last column)


Using test data with 5000 clusters and σ= 250 (with Tikhonov regularization λ= 20)
Actual outputs
● Estimated outputs
E_{rms}=2.225294
$t_{2estimate}$
t_2 (i.e last column)