Project2_part2_notebook_Final

November 13, 2022

LINMA2472 : Project 2 - part 2, Random Fourier Features

Author: Remi Delogne, remi.delogne@uclouvain.be

Import the following packages and functions. Refer to their documentation on the internet for more information on installation and usage.

[1]:
```
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import datasets, svm
from sklearn.svm import LinearSVC
from sklearn import metrics
from sklearn.metrics import accuracy_score

from keras.datasets import mnist #Contains the dataset
from matplotlib import pyplot
import matplotlib.pyplot as plt

import time #Used to find the execution time of a part of the code

from IPython.display import display, HTML #For visual comfort
display(HTML("<style>.container { width:80% !important; }</style>"))
```

```
2022-11-13 20:52:48.402984: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

```
<IPython.core.display.HTML object>
```

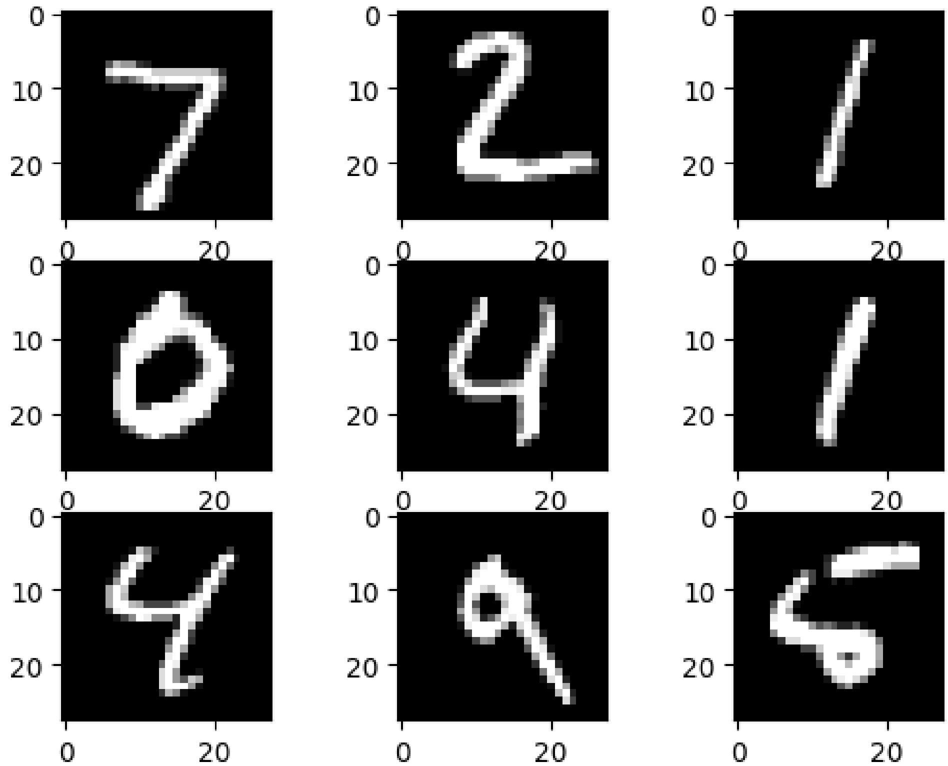
[2]:
```
import numpy
```

[3]:
```
import sys
!{sys.executable} -m pip install tabulate

import tabulate as tab
import pandas as pd
```

```
Requirement already satisfied: tabulate in
/Users/Pierre/.pyenv/versions/3.10.0/lib/python3.10/site-packages (0.9.0)
WARNING: You are using pip version 21.2.3; however, version 22.3.1 is
available.
You should consider upgrading via the
'/Users/Pierre/.pyenv/versions/3.10.0/bin/python -m pip install --upgrade pip'
command.
```

[4]:

```
# load dataset and rescale the data to [0,15]
''' load dataset: (we use the larger subset for testing and the smaller
for training to demonstrate the efficiency of evaluating of new instances with␣
↪RFF)'''
(testX,testy),(trainX,trainy) = mnist.load_data()
#Rescaling
trainX = np.floor(trainX/16)
testX = np.floor(testX/16)

#Plot some images, for fun
for i in range(9):
    pyplot.subplot(330+1+i)
    pyplot.imshow(trainX[i], cmap=pyplot.get_cmap('gray'))
pyplot.show()

#Put the data in vector form
trainX=trainX.reshape((10000,-1))
testX=testX.reshape((60000,-1))
print(trainX.shape)
```




```
(10000, 784)
```

0.0.1 Use the *time* package functions to time the executions of parts of your code:

[5]:
```
#Example of timing a piece of code
tik=time.perf_counter() # Start
for i in range (1000000):
    i=i+1
tok=time.perf_counter() # Finish
print(f'Total time: {tok-tik:.3f} seconds')#prints the result to 3 decimal␣
↪places
```

```
Total time: 0.228 seconds
```

0.0.2 Train a linear SVM on the training data and evaluate it on the testing data

Use the tik-tok method to see how long the classifier takes to evaluate the 60.000 testing instances.

Use the accuracy metric to judge the quality of your classifier.

[6]:
```
#Define the classifier
clfLin=svm.SVC(kernel="linear")
```

```
#Train the classifier
tik = time.perf_counter() # Start measuring training time
clfLin.fit(trainX,trainy)
tok = time.perf_counter() # Stop measuring training time
linear_training_time = tok - tik

#Evaluate its accuracy
tik = time.perf_counter()
predicted=clfLin.predict(testX)
score_linear=accuracy_score(testy,predicted)
tok = time.perf_counter()
linear_testing_time = tok - tik

print(f"Training finished in {linear_training_time:.3f} seconds,")
print(f"Testing Finished in {linear_testing_time:.3f} seconds with accuracy of␣
↪{score_linear:.3f}.")
```

```
Training finished in 10.655 seconds,
Testing Finished in 98.010 seconds with accuracy of 0.907.
```

0.0.3 Train a Kernel SVM with the Gaussian Kernel on the training data and evaluate it on the testing data

Use the tik-tok method to see how long the classifier takes to evaluate the 60000 testing instances.

Use the accuracy metric to judge the quality of your classifier.

You may stick to the default parameters of sci-kit learn.

[7]:
```
#Define the classifier
clfKernel=svm.SVC(kernel='rbf')

#Use it
tik = time.perf_counter()
clfKernel.fit(trainX, trainy)
tok = time.perf_counter()
rbf_training_time = tok - tik

tik = time.perf_counter()
predicted=clfKernel.predict(testX)
score_rbf=accuracy_score(testy,predicted)
tok = time.perf_counter()
rbf_testing_time = tok - tik

print(f"Training finished in {rbf_training_time:.3f} seconds,")
print(f"Testing Finished in {rbf_testing_time:.3f} seconds with accuracy of␣
↪{score_rbf:.3f}.")
```

```
Training finished in 19.983 seconds,
```

```
Testing Finished in 260.767 seconds with accuracy of 0.956.
```

0.0.4 *TO DO*: Use the following functions to implement Random Fourier Features

You are here going to try to approximate the Gaussian kernel used in the second classfier.

Use the first function to generate your ω_i (using an appropriate distribution) and your b_i (using appropriate distributions), this should return D vectors $\omega_i \in \mathbb{R}^d$ (in the form of a matrix for example) and D values $b_i \in [0, 2\pi]$.

Use the second function to create the mapping $z(x)$ as described in the slides.

[8]:
```
def generate_freq(N,D,sigma):
    W= np.random.normal(loc=0, scale=1/sigma, size=(N, D))
    b= np.random.uniform(0, 2*np.pi, size=D)
    return W,b

def transform(x,w,b,D):
    Z= np.zeros(D)
    for i in range(D):
        a = w.T[i] @ x + b[i]
        Z[i] = np.sqrt(2/D)* (np.cos(a))
    return Z

print(testX.shape)
print(trainX.shape)
```

```
(60000, 784)
(10000, 784)
```

0.0.5 *TO DO*: Transform your trainX and testX

Use the function you defined to transform your data.

Make sure you only generate W and b once.

Use a standard deviation of $\frac{1}{100}$ et $D = 300$ random features to start with. Watch out, in the original version of the homework it was specified that the variance was $\frac{1}{100}$ but it must be the standard deviation instead.

You may also use the tik-tok method to time the procedure of creating Random features.

[9]:
```
D=300 #Number of sample vectors w_i
sigma=100 #Variance of distributon
d=28*28 #Original number of dimensions
N = trainX.shape[1]

tik = time.perf_counter()
W,b=generate_freq(N,D,sigma)

trainX_rff=[]
```

```
testX_rff=[]

#trainX_rff=transform(trainX,W,b,D)

tik=time.perf_counter() # Start
for x_train in trainX:
    trainX_rff.append(transform(x_train,W,b,D))
    tok=time.perf_counter() # Finish
print('Total time to calculate trainX_rff: %ss' % (int(tok-tik)))

#testX_rff=transform(testX,W,b,D)
tik=time.perf_counter() # Start
for x_test in testX:
    testX_rff.append(transform(x_test,W,b,D))
    tok=time.perf_counter() # Finish
print('Total time to calculate testX_rff: %ss' % (int(tok-tik)))
tok = time.perf_counter()

rff_time = tok - tik

print(f"RFF transformation time : {rff_time:.3f} seconds.")
```

```
Total time to calculate trainX_rff: 27s
Total time to calculate testX_rff: 169s
RFF transformation time : 169.112 seconds.
```

[10]:
```
#Sanity check, do the dimensions of your transformations match your expectation?
#Bear in mind that there are more instances in the test set than in the␣
 ↪trianing set here
print(f"Dimension of trainX after transformation : {np.array(trainX_rff).shape}.
 ↪")
print(f"Dimension of testX after transformation : {np.array(testX_rff).shape}.")
```

```
Dimension of trainX after transformation : (10000, 300).
Dimension of testX after transformation : (60000, 300).
```

0.0.6 Use another linear SVM to classify the transformed data

Now that the instances have been transformed, theory tells us that they are much more ameneable to linear classification than before.

[11]:
```
#Define the classifier
clfRff = svm.SVC(kernel="linear",C=np.inf)

#Use it

tik = time.perf_counter()
clfRff.fit(trainX_rff, trainy)
```

```
tok = time.perf_counter()
training_time_rff = tok - tik
print(f"Training Finished in {training_time_rff:.3f} seconds")

tik = time.perf_counter()
predicted = clfRff.predict(testX_rff)
score_rff = accuracy_score(testy,predicted)
tok = time.perf_counter()
testing_time_rff = tok - tik
print(f"Testing Finished in {testing_time_rff:.3f} seconds with accuracy of␣
 ↪{score_rff:.3f}")
```

```
Training Finished in 6.382 seconds
Testing Finished in 43.694 seconds with accuracy of 0.895
```

0.0.7 Additional workspace

Investigate the relationship between D and the accuracy of the classifier.

[12]:
```
#Define a vector to store the accuracy values you will get

def compute_rff(X,D,W,b):
    N = np.array(X).shape[1]
    X_rff = []
    for x in X:
        X_rff.append(transform(x,W,b,D))
    return X_rff

accuracy=[]
duration_create = []
duration_train = []
duration_predict = []

#Define for which values of D you want to test the RFF

D_values=[10,20,40,80,160,320,640,1000]

#maybe time some operations in the loop as well to see the RFF classifier␣
 ↪becomes too slow as D grows

clf = svm.SVC(kernel="linear")

for i in range(len(D_values)):
    D = D_values[i]
    print('%s° interaction' % (i+1))
    print('Value of D = ',D)
    tik=time.perf_counter() # Start
    N = trainX.shape[1]
```

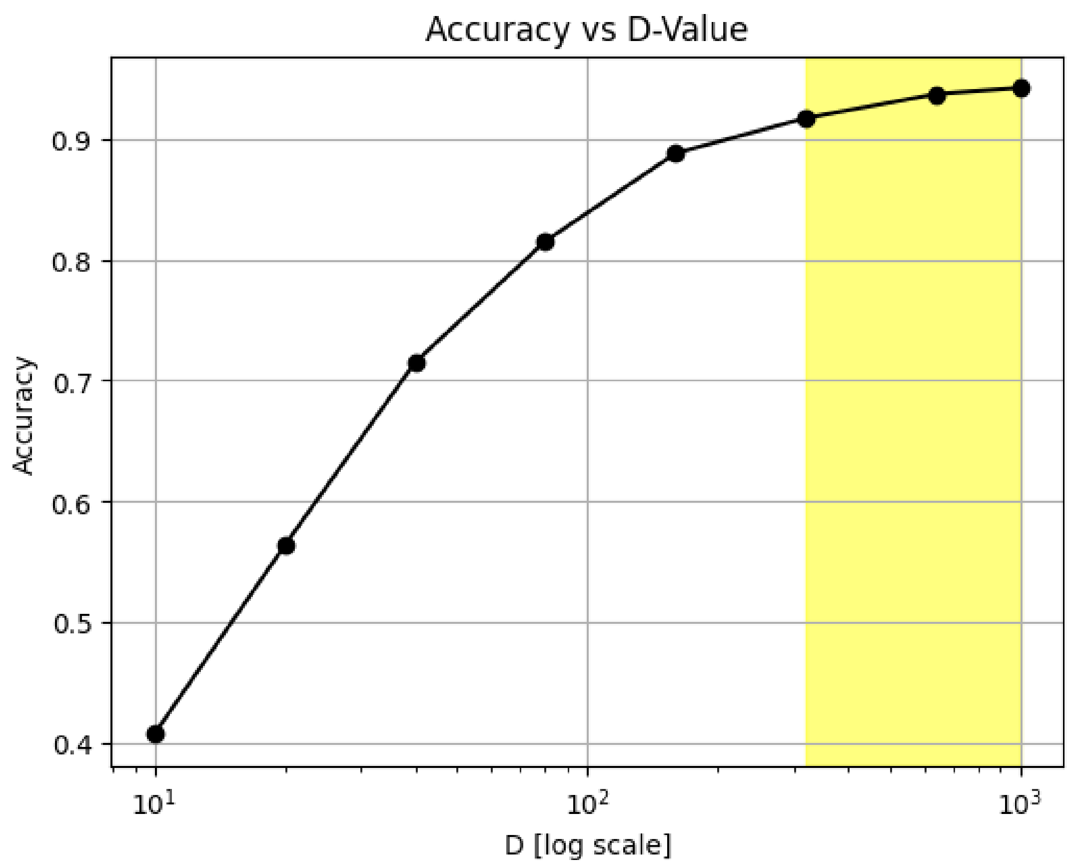
```
    W,b = generate_freq(N,D,sigma=100)
    trainX_rff= compute_rff(trainX,D,W,b)
    testX_rff= compute_rff(testX,D,W,b)
    tok=time.perf_counter() # Finish
    print('RFF Created in: %ss' % (int(tok-tik)))
    duration_create.append(tok-tik)
    #Train and evaluate a linear classifier
    tik=time.perf_counter() # Start
    clf.fit(trainX_rff, trainy)
    tok=time.perf_counter() # Finish
    print('RFF Training Finished in: %ss' % (int(tok-tik)))
    duration_train.append(tok-tik)
    tik=time.perf_counter() # Start
    predicted = clf.predict(testX_rff)
    tok=time.perf_counter() # Finish
    print('Total Prediction Time with RFF is: %ss' % (int(tok-tik)))
    duration_predict.append(tok-tik)
    acc= accuracy_score(testy,predicted)
    accuracy.append(acc)
    print('Accuracy: %s' % (round(acc*100,2)),'%')
    print('')
```

```
1° interaction
Value of D =  10
RFF Created in: 6s
RFF Training Finished in: 6s
Total Prediction Time with RFF is: 32s
Accuracy: 40.72 %

2° interaction
Value of D =  20
RFF Created in: 12s
RFF Training Finished in: 3s
Total Prediction Time with RFF is: 28s
Accuracy: 56.34 %

3° interaction
Value of D =  40
RFF Created in: 26s
RFF Training Finished in: 3s
Total Prediction Time with RFF is: 43s
Accuracy: 71.49 %

4° interaction
Value of D =  80
RFF Created in: 66s
RFF Training Finished in: 4s
```

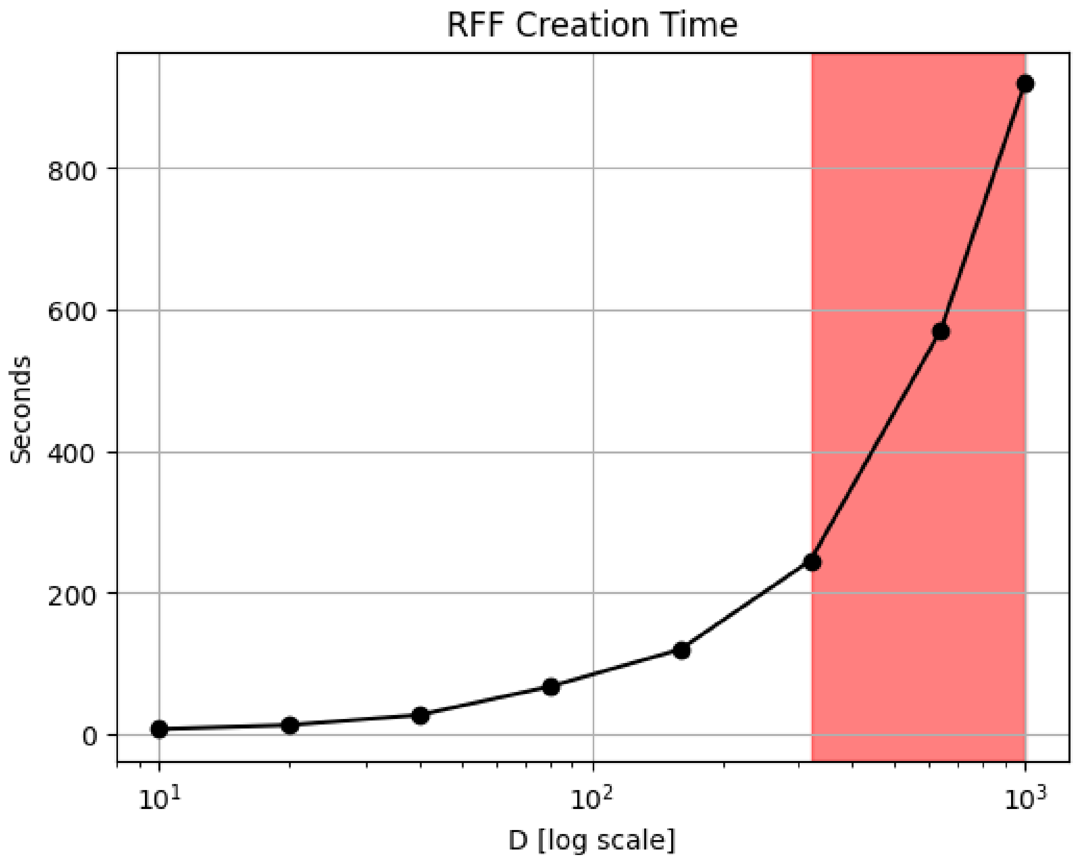
```
Total Prediction Time with RFF is: 25s
Accuracy: 81.53 %

5° interaction
Value of D =  160
RFF Created in: 119s
RFF Training Finished in: 6s
Total Prediction Time with RFF is: 66s
Accuracy: 88.82 %

6° interaction
Value of D =  320
RFF Created in: 244s
RFF Training Finished in: 16s
Total Prediction Time with RFF is: 118s
Accuracy: 91.73 %

7° interaction
Value of D =  640
RFF Created in: 570s
RFF Training Finished in: 21s
Total Prediction Time with RFF is: 201s
Accuracy: 93.72 %

8° interaction
Value of D =  1000
RFF Created in: 919s
RFF Training Finished in: 34s
Total Prediction Time with RFF is: 286s
Accuracy: 94.23 %
```

0.0.8 Don't forget to add plots and other nice things

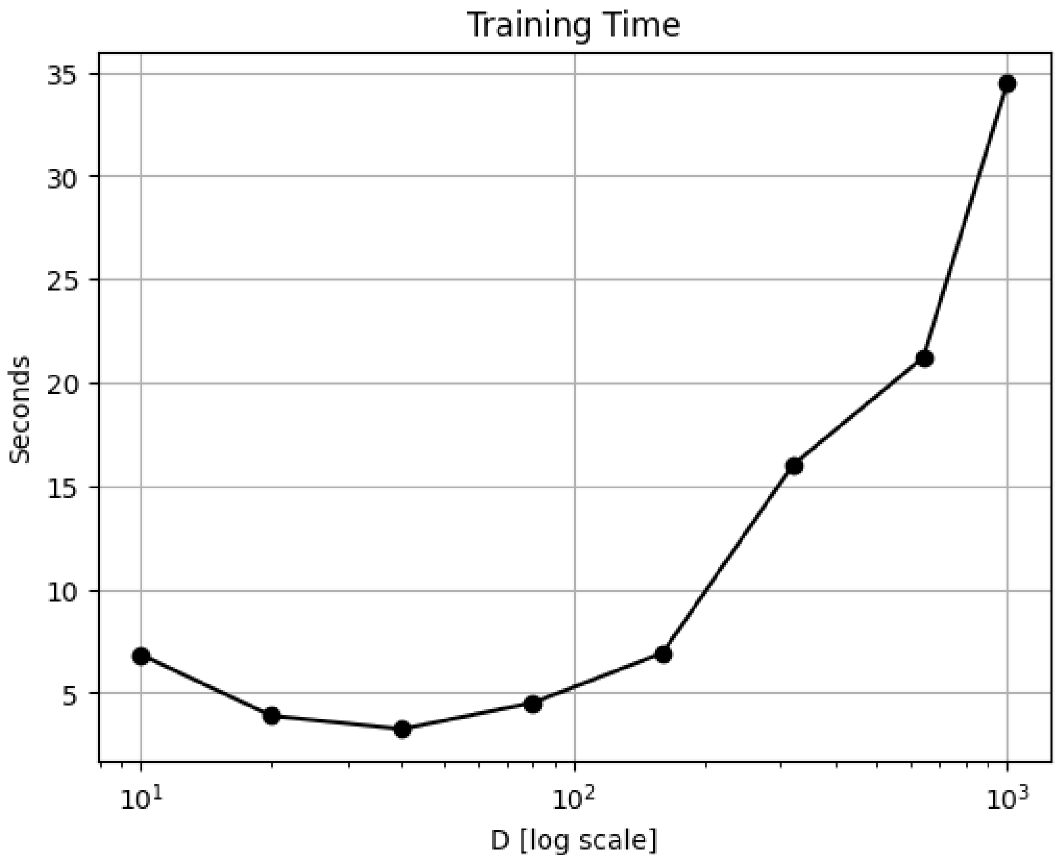
[13]:
```
plt.plot(D_values, accuracy, '-ok')
plt.xscale("log")
plt.xlabel("D [log scale]")
plt.ylabel("Accuracy")
plt.title('Accuracy vs D-Value')
plt.axvspan(D_values[5], D_values[7], color='yellow', alpha=0.5)
plt.grid()
plt.show()
```



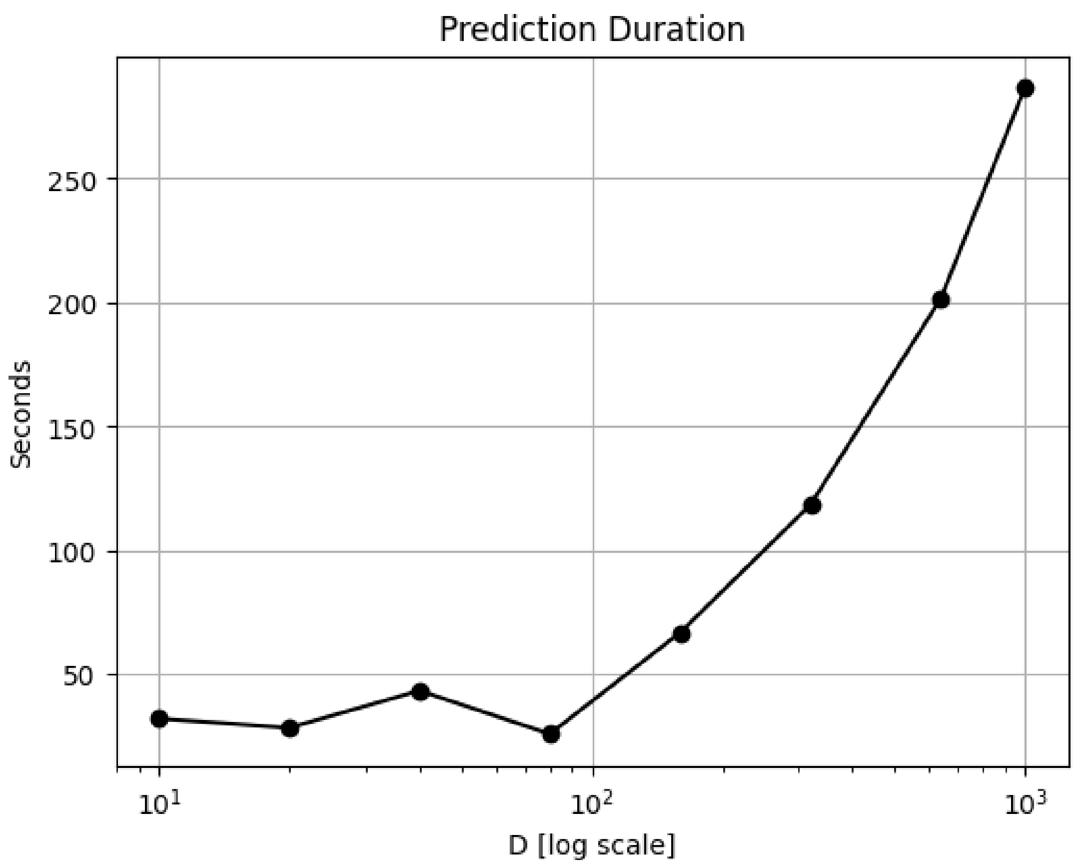
[14]:
```
plt.plot(D_values,duration_create, '-ok')
plt.xscale("log")
plt.xlabel("D [log scale]")
plt.ylabel("Seconds")
plt.title('RFF Creation Time')
plt.axvspan(D_values[5], D_values[7], color='red', alpha=0.5)
plt.grid()
plt.show()
```



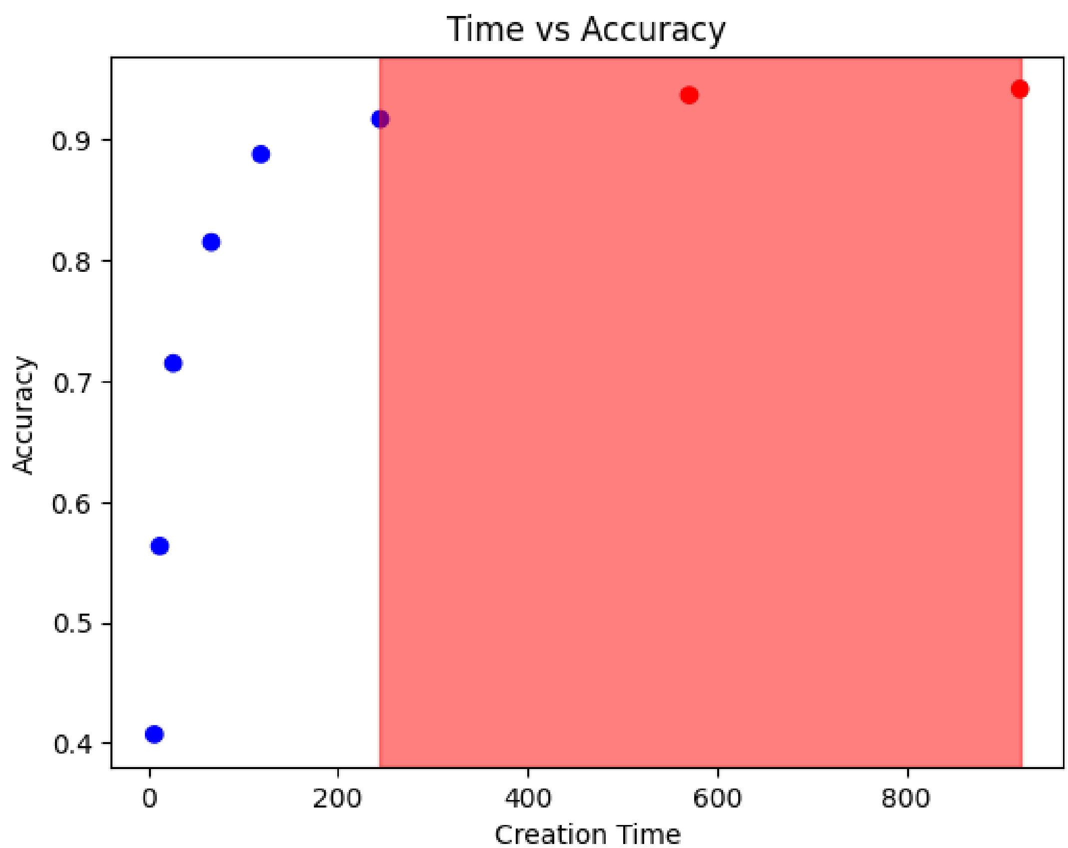
[15]:
```
plt.plot(D_values,duration_train, '-ok')
plt.xscale("log")
plt.xlabel("D [log scale]")
plt.ylabel("Seconds")
plt.title('Training Time')
plt.grid()
plt.show()
```



[16]:
```
plt.plot(D_values,duration_predict, '-ok')
plt.xscale("log")
plt.xlabel("D [log scale]")
plt.ylabel("Seconds")
plt.title('Prediction Duration')
plt.grid()
plt.show()
```



[17]:
```
c = numpy.array(['b', 'b', 'b', 'b', 'b', 'b','r','r'])

plt.scatter(duration_create, accuracy, color= c)
plt.rcParams["figure.figsize"] = (10,10)
plt.xlabel("Creation Time")
plt.ylabel("Accuracy")
plt.title("Time vs Accuracy")
plt.axvspan(duration_create[5], duration_create[7], color='red', alpha=0.5)
plt.show()
```



[18]:
```
h = ['D_values','duration_create', 'duration_train', 'duration_predict',␣
↪'accuracy']
df = pd.DataFrame(list(zip(D_values, duration_create, duration_train,␣
↪duration_predict, accuracy)), columns =h)
df
```

[18]:

	D_values	duration_create	duration_train	duration_predict	accuracy
0	10	6.704312	6.830697	32.018846	0.407250
1	20	12.309756	3.880708	28.306841	0.563350
2	40	26.587763	3.240207	43.277779	0.714917
3	80	66.729798	4.500479	25.837425	0.815300
4	160	119.144771	6.911949	66.743719	0.888183
5	320	244.979044	16.010722	118.313320	0.917333
6	640	570.912530	21.206383	201.549386	0.937200
7	1000	919.798063	34.461210	286.533059	0.942300

[20]:
```
table = [['linear_testing_time', linear_testing_time],␣
↪['rbf_testing_time',rbf_testing_time], ['testing_time_rff',␣
↪testing_time_rff]]
h = ['Method','Testing Time']
```

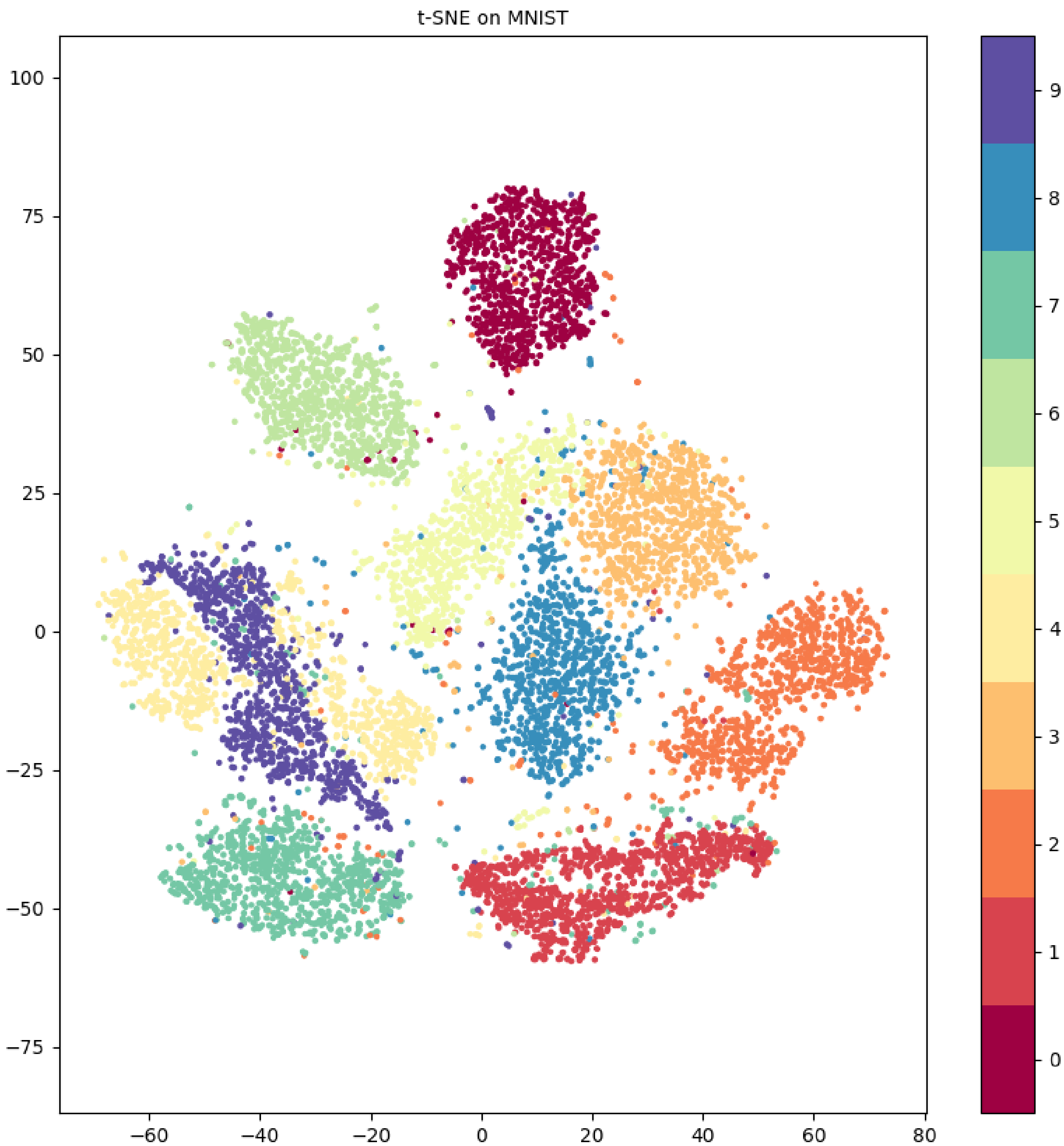
```
tb = tab.tabulate(table, headers=h, tablefmt="fancy_grid")
print(tb)
```

Method	Testing Time
linear_testing_time	98.0101
rbf_testing_time	260.767
testing_time_rff	43.6941

0.0.9 Good luck =D

[21]:
```
from sklearn.manifold import TSNE
tsne = TSNE(random_state = 42, n_components=2, perplexity=50).
 ↪fit_transform(trainX)


plt.scatter(tsne[:, 0], tsne[:, 1], s= 5, c=trainy, cmap='Spectral')
plt.gca().set_aspect('equal', 'datalim')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))
plt.title('t-SNE on MNIST', fontsize=10);
```

```
/Users/Pierre/.pyenv/versions/3.10.0/lib/python3.10/site-
packages/sklearn/manifold/_t_sne.py:800: FutureWarning: The default
initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(
/Users/Pierre/.pyenv/versions/3.10.0/lib/python3.10/site-
packages/sklearn/manifold/_t_sne.py:810: FutureWarning: The default learning
rate in TSNE will change from 200.0 to 'auto' in 1.2.
  warnings.warn(
```


t-SNE on MNIST
100
75
50
25
0
−25
−50
−75
−60
−40
−20
0
20
40
60
80
9
8
7
6
5
4
3
2
1
0