Name:

Hanliang Jiang

Netid:

hj33

CS 441 - HW1: Instance-based Methods

Complete the sections below. You do not need to fill out the checklist.

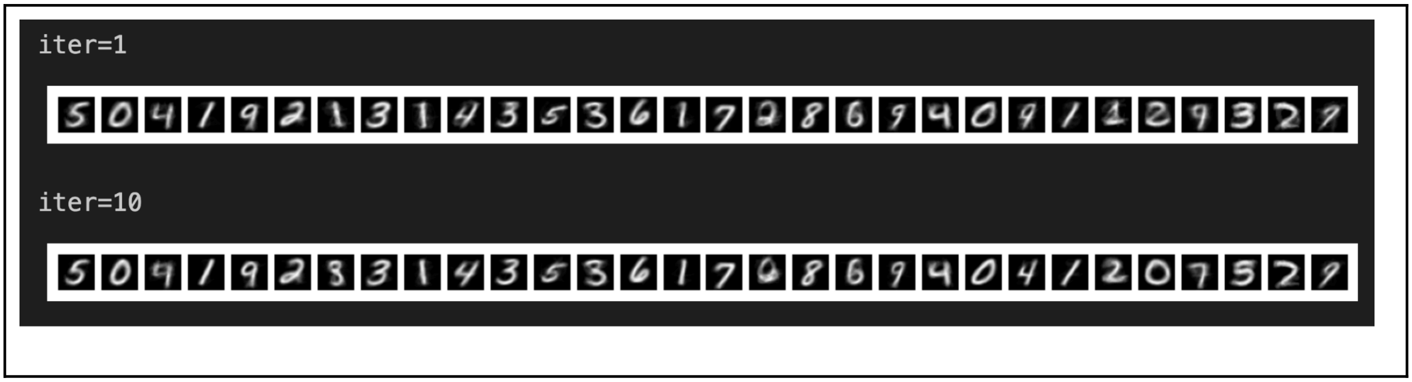
Total Points Available	**[ ] / 145**
1. Retrieval, K-means, 1-NN on MNIST	
a. Retrieval	[ ] / 5
b. K-means	[ ] / 15
c. 1-NN	[ ] / 10
2. Make it fast	
a. K-means plot	[ ] / 15
b. 1-NN error plots	[ ] / 8
c. 1-NN time plots	[ ] / 7
d. Most confused label	[ ] / 5
3. Temperature Regression	
a. RMSE Tables	[ ] / 20
4. Conceptual questions	[ ] / 15
5. Stretch Goals	
a. Evaluate effect of K for MNIST	[ ] / 15
b. Evaluate effect of K for Temp Reg.	[ ] / 15
c. Compare Kmeans more iterations vs. restarts	[ ] / 15

1. Retrieval, K-means, 1-NN on MNIST

a. What index is returned for x_test[1]?

28882

b. Paste the display of clusters after the 1st and 10th iteration for K=30.

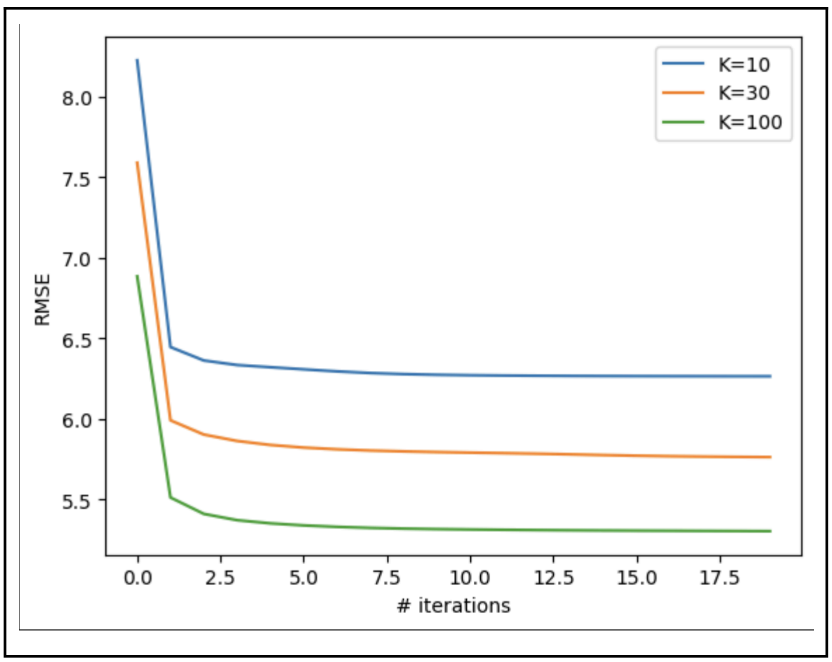


c. Error rate for first 100 test samples, using first 10,000 training samples (x.x)

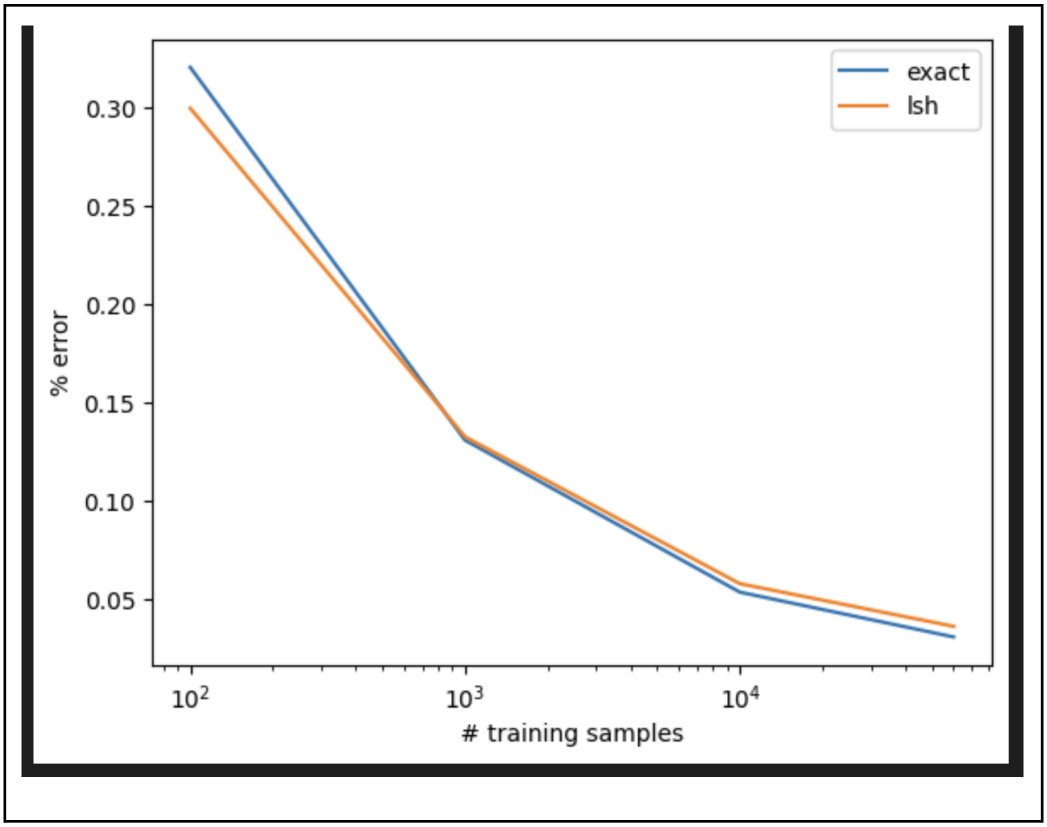
8.0%

2. Make it fast

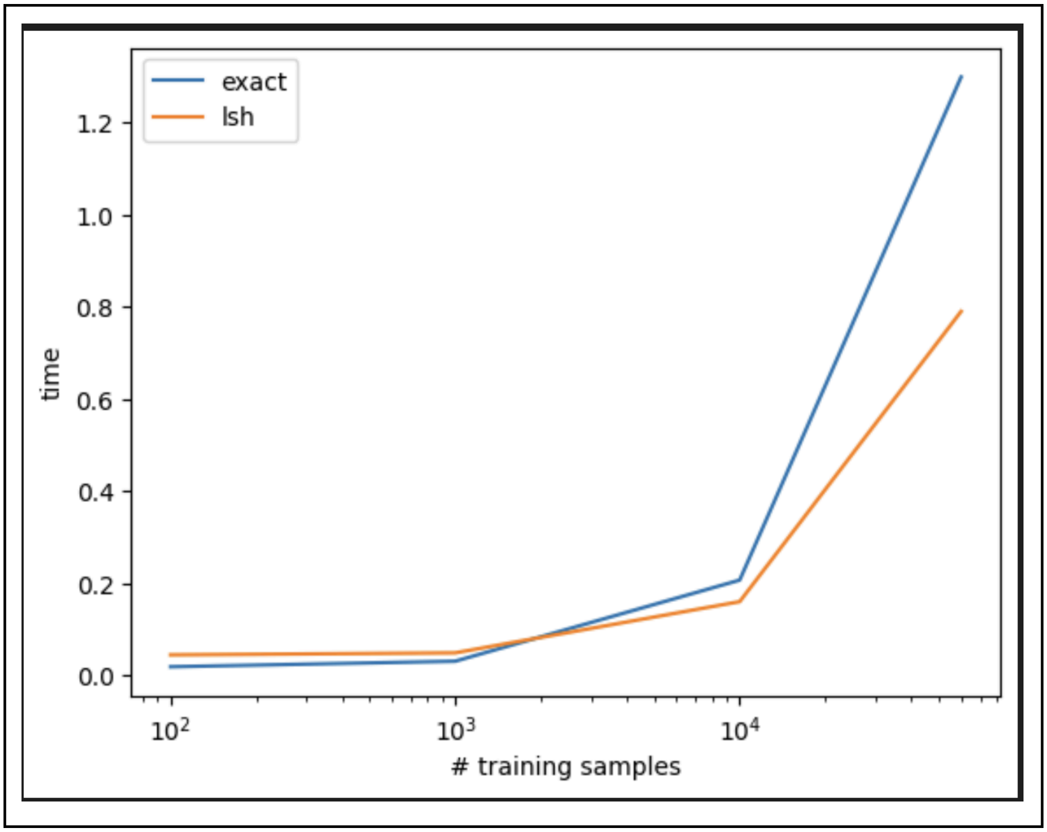
a. KMeans plot of RMSE vs iterations for K=10, 30, 100



b. Nearest neighbor error vs training size plot



c. Nearest neighbor time vs training size plot



d. What label is most commonly confused with ‘2’?

‘1’

3. Temperature Regression

a. Table of RMSE for KNN with K=5 (x.xx)

	KNN (K=5)
Original Features	3.249
Normalized Features	2.977

4. Test your understanding

Fill in the letter corresponding to the answer. If you’re not sure, you can sometimes run small experiments to check.

1. Is K-means guaranteed to decrease RMSE between nearest cluster and samples at each iteration until convergence?

a. Yes
b. No

b

2. If you increase K, is K-means expected or guaranteed to achieve lower RMSE?
    a. Guaranteed
    b. Expected but not guaranteed
    c. Not expected

c

3. In K-NN regression, for training labels y, what is the lowest target value that can possibly be predicted for any query?
    a. Min(y)
    b. Mean(y)
    c. Can't be determined

a

4. Would you expect the "training error" for 1-NN to be higher or lower than 3-NN for classification? Training error is the error if you test on the training data.
    a. Higher
    b. Lower
    c. It's problem-dependent

b

5. Would you expect the test error for 1-NN to be higher or lower than for 3-NN for regression?
    a. Higher
    b. Lower
    c. It's problem-dependent

c

5. Stretch Goals (optional)

a. Select best K parameter for K-NN MNIST classification in K=1, 3, 5, 11, 25. (x.xx)

Validation Set Performance	K=1	K=3	K=5	K=11	K=25

% error	2.88%	2.82%	2.94%	3.14%	3.87%

Best K:

3

Test % error (x.xx)

2.959%

b. Select best K parameter for K-NN temperature regression in K=1, 3, 5, 11, 25. (x.xx)

Validation Set RMSE	K=1	K=3	K=5	K=11	K=25
Original Features	5.99	5.07	4.81	4.635	4.48
Normalized Features	4.96	4.08	3.78	3.629	3.54

Best Setting (K, feature type):

K=25, Original Features

Test RMSE (x.xx)

3.04

c. Kmeans, MNIST: compare average and standard deviation RMSE based on number of iterations and number of restarts

(4 digit precision)

K=30	RMSE avg	RMSE std
20 iterations, 1 restart	5.819	0.00
4 iterations, 5 restarts	5.775	0.00
50 iterations, 1 restart	5.789	0.00
10 iterations, 5 restarts	5.769	0.00

Acknowledgments / Attribution

List any outside sources for code or ideas or “None”.
None

CS441: Applied ML - HW 1

Parts 1-2: MNIST

Include all the code for generating MNIST results below

In [ ]:
```
# initialization code
import numpy as np
from tensorflow import keras
from keras.datasets import mnist
# from tensorflow import keras
%matplotlib inline
from matplotlib import pyplot as plt
from scipy import stats


def load_mnist():
  '''
  Loads, reshapes, and normalizes the data
  '''
  (x_train, y_train), (x_test, y_test) = mnist.load_data() # loads MNIST dat
  x_train = np.reshape(x_train, (len(x_train), 28*28))  # reformat to 768-d
  x_test = np.reshape(x_test, (len(x_test), 28*28))
  maxval = x_train.max()
  x_train = x_train/maxval  # normalize values to range from 0 to 1
  x_test = x_test/maxval
  return (x_train, y_train), (x_test, y_test)

def display_mnist(x, subplot_rows=1, subplot_cols=1):
  '''
  Displays one or more examples in a row or a grid
  '''
  if subplot_rows>1 or subplot_cols>1:
    fig, ax = plt.subplots(subplot_rows, subplot_cols, figsize=(15,15))
    for i in np.arange(len(x)):
      ax[i].imshow(np.reshape(x[i], (28,28)), cmap='gray')
      ax[i].axis('off')
  else:
      plt.imshow(np.reshape(x, (28,28)), cmap='gray')
      plt.axis('off')
  plt.show()
```

In [ ]:
```
# example of using MNIST load and display  functions
(x_train, y_train), (x_test, y_test) = load_mnist()
display_mnist(x_train[:10],1,10)
print('Total size: train={}, test ={}'.format(len(x_train), len(x_test)))
```

```
Total size: train=60000, test =10000
```

1. Retrieval, Clustering, and NN Classification

In [ ]:
```
# Retrieval
import math
def get_nearest(X_query, X):
  ''' Return the index of the sample in X that is closest to X_query accordi
      to L2 distance '''
  # TO DO
  mindis = math.inf
  n = X.shape[0]
  res = -1
  for i in range(n):
    dis = np.linalg.norm(X[i]-X_query)
    if dis<mindis:
      mindis = dis
      res = i
  if res>=0:
    return res
  else:
    print("error happens!")


# print(x_train.shape)
# print(x_test.shape)
j = get_nearest(x_test[0], x_train)
print(j)
j = get_nearest(x_test[1], x_train)
print(j)
```

```
53843
28882
```

In [ ]:
```
# K-means
def kmeans(X, K, niter=10):
  '''
  Starting with the first K samples in X as cluster centers, iteratively ass
  point to the nearest cluster and compute the mean of each cluster.
  Input: X[i] is the ith sample, K is the number of clusters, niter is the r
  Output: K cluster centers
  '''
  # TO DO -- add code to display cluster centers at each iteration also
  centers = np.copy(X[:K])
  for i in range(niter):
    choice = np.array([get_nearest(x, centers) for x in X])
    for j in range(K):
      centers[j] = np.mean(X[choice==j],axis=0)
    if i==0 or i==9:
      print(f"iter={i+1}")
      display_mnist(centers,1,30)
  return centers
K=30
centers = kmeans(x_train[:1000], K)
```

```
iter=1
```

```
iter=10
```

In [ ]:
```
# 1-NN

# TO DO
train = x_train[:10000]
wrong = 0
for i in range(100):
  if y_test[i] != y_train[get_nearest(x_test[i], train)]:
    wrong += 1
print(f"incorrect rate: {wrong}%")
```

```
incorrect rate: 8%
```

2. Make it fast

In [ ]:
```
# install libraries you need for part 2
!apt install libomp-dev
!pip install faiss-cpu
import faiss
import time
```

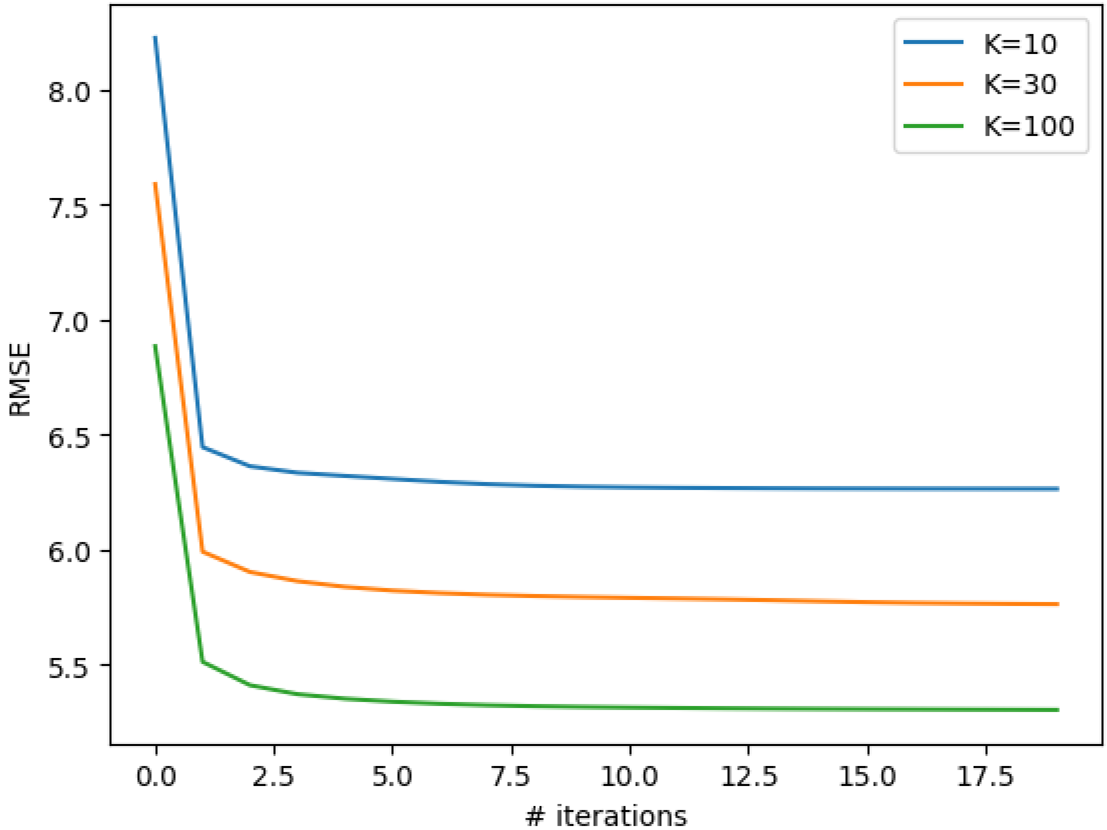
```
The operation couldn’t be completed. Unable to locate a Java Runtime that su
pports apt.
Please visit http://www.java.com for information on installing Java.

Requirement already satisfied: faiss-cpu in /Users/janghl/anaconda3/envs/am
l/lib/python3.10/site-packages (1.7.4)
```

In [ ]:
```
# retrieval

# TO DO (check that you're using FAISS correctly)

def fast_nearest(X_query, X, K=1, LSH=False):
  if not LSH:
    index = faiss.IndexFlatL2(x_train.shape[1])  # set for exact search
  else:
    dim = X.shape[1]
    index = faiss.IndexLSH(dim, dim)
  index.add(X) # add the data
  dist, idx = index.search(X_query, K) # returns index and sq err for each s
  return np.array(dist), np.array(idx)

dist, idx = fast_nearest(x_test, x_train)
print(idx[0])
print(idx[1])
```

```
[53843]
[28882]
```

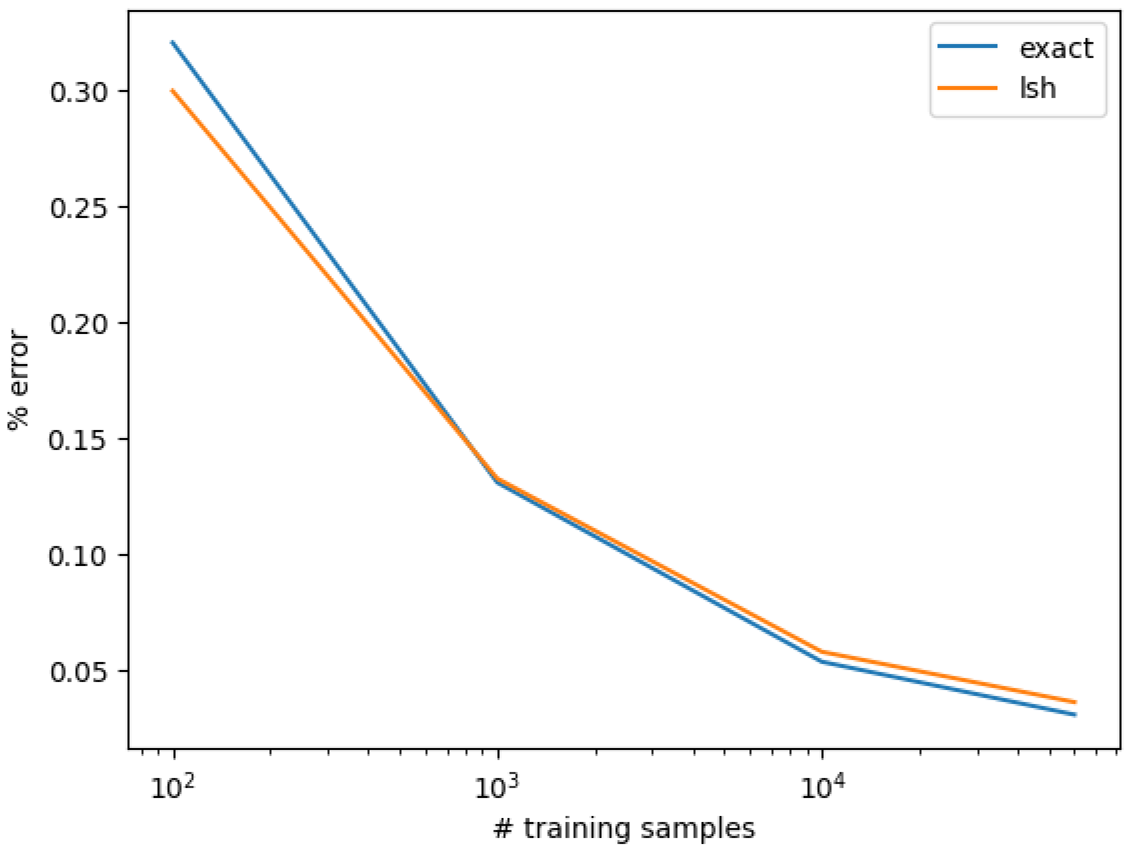
In [ ]:
```
# K-means

def kmeans_fast(X, K, niter=10):
```

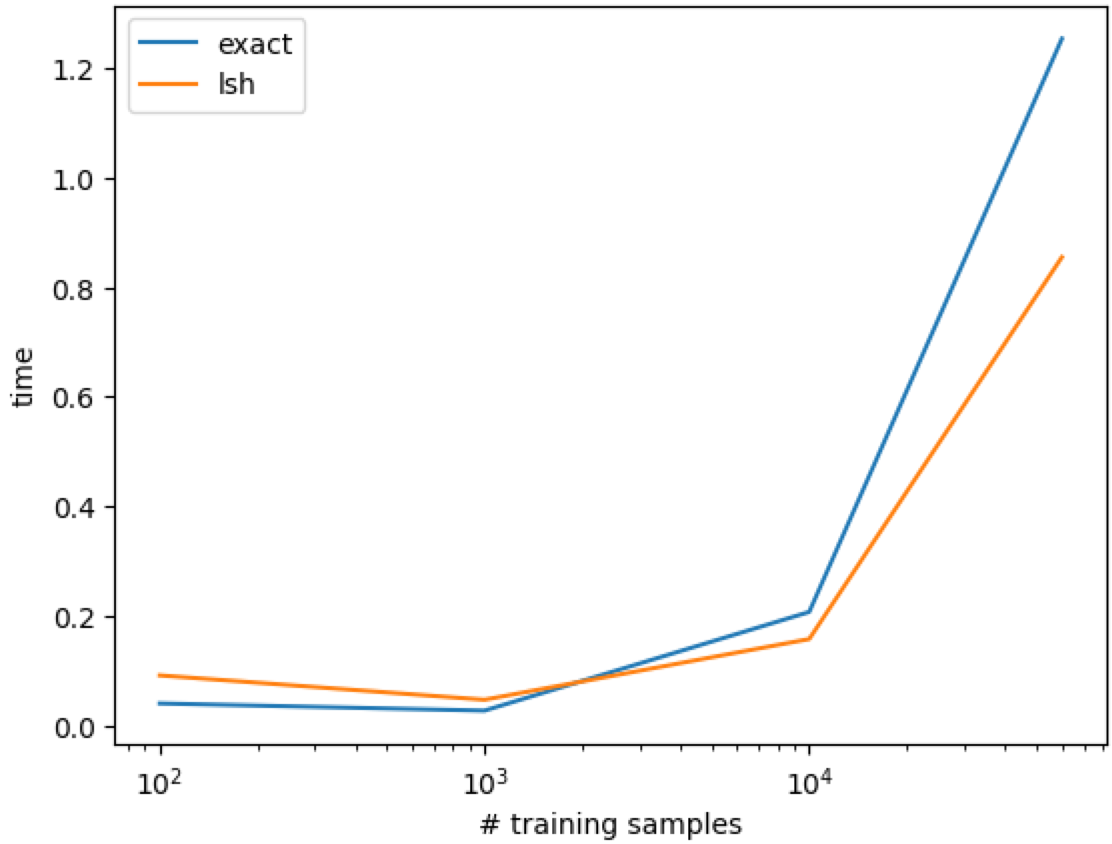
```
    '''
    Starting with the first K samples in X as cluster centers, iteratively ass
    point to the nearest cluster using faiss and compute the mean of each clus
    Input: X[i] is the ith sample, K is the number of clusters, niter is the r
    Output: K cluster centers
    '''

    # TO DO (you can base this on part 1, but use FAISS for search)
    # if you include display code, you need to re-organize the plotting code b
    rmse = []
    centers = np.copy(X[:K])
    for i in range(niter):
      dist, idx = fast_nearest(X, centers)
      choice = np.array(idx).reshape(idx.shape[0])
      rmse.append(np.sqrt(np.sum(dist)/dist.shape[0]))
      for j in range(K):
        centers[j] = np.mean(X[choice==j],axis=0)
    return centers, rmse

K=10
centers, rmse = kmeans_fast(x_train, K, niter=20)
plt.plot(np.arange(len(rmse)), rmse, label='K=10')

K=30
centers, rmse = kmeans_fast(x_train, K, niter=20)
plt.plot(np.arange(len(rmse)), rmse, label='K=30')

K=100
centers, rmse = kmeans_fast(x_train, K, niter=20)
plt.plot(np.arange(len(rmse)), rmse, label='K=100')
plt.legend(), plt.ylabel('RMSE'), plt.xlabel('# iterations')
plt.show()
```



In [ ]:
```
# 1-NN

nsample = [100, 1000, 10000, 60000]

# TO DO
def ONN(x_train, x_test, s, LSH):
  begin = time.time()
  train = x_train[:s]
  right = 0
  if LSH:
    dim = x_test.shape[1]
    index = faiss.IndexLSH(dim, dim)
  else:
    index = faiss.IndexFlatL2(x_test.shape[1])
  index.add(train)
  dist, idx = index.search(x_test,1) # returns index and sq err for each sam
  idx = np.array(idx).reshape(idx.shape[0])
  for i in range(len(x_test)):
    if y_test[i] == y_train[idx[i]]:
      right += 1
  acc = right/len(x_test)
  end = time.time()
  timing = end-begin
  return acc, timing

acc_exact = []
acc_lsh = []
timing_exact = []
timing_lsh = []
```

```
for s in nsample:
  acc_exact_elem, timing_exact_elem = ONN(x_train, x_test, s, LSH=False)
  acc_lsh_elem, timing_lsh_elem = ONN(x_train, x_test, s, LSH=True)
  acc_exact.append(1-acc_exact_elem)
  acc_lsh.append(1-acc_lsh_elem)
  timing_exact.append(timing_exact_elem)
  timing_lsh.append(timing_lsh_elem)


plt.semilogx(nsample, acc_exact, label='exact')
plt.semilogx(nsample, acc_lsh, label='lsh')
plt.legend(), plt.ylabel('% error'), plt.xlabel('# training samples')
plt.show()

plt.semilogx(nsample, timing_exact, label='exact')
plt.semilogx(nsample, timing_lsh, label='lsh')
plt.legend(), plt.ylabel('time'), plt.xlabel('# training samples')
plt.show()
```





In [ ]:
```
# Confusion matrix
from sklearn import metrics
# TO DO
index = faiss.IndexFlatL2(x_test.shape[1])
index.add(train)
dist, idx = index.search(x_test,1) # returns index and sq err for each sampl
idx = np.array(idx).reshape(idx.shape[0])
metrics.confusion_matrix(y_test, y_train[idx])
```

Out[ ]:
```
array([[ 971,    1,    1,    0,    0,    1,    4,    1,    1,    0],
       [   0, 1130,    1,    2,    0,    0,    2,    0,    0,    0],
       [  15,   14,  960,    9,    1,    1,    4,   23,    5,    0],
       [   1,    1,    4,  945,    1,   23,    3,   12,    9,   11],
       [   0,   11,    0,    0,  915,    0,    8,    6,    2,   40],
       [   8,    4,    0,   28,    3,  828,   10,    2,    2,    7],
       [   9,    3,    0,    0,    3,    4,  938,    0,    1,    0],
       [   0,   26,    8,    1,    4,    1,    0,  972,    1,   15],
       [   9,    5,    6,   32,    4,   21,    7,    8,  863,   19],
       [   6,    6,    3,    7,   20,    4,    1,   17,    4,  941]])
```

Part 3: Temperature Regression

Include all your code used for part 2 in this section.

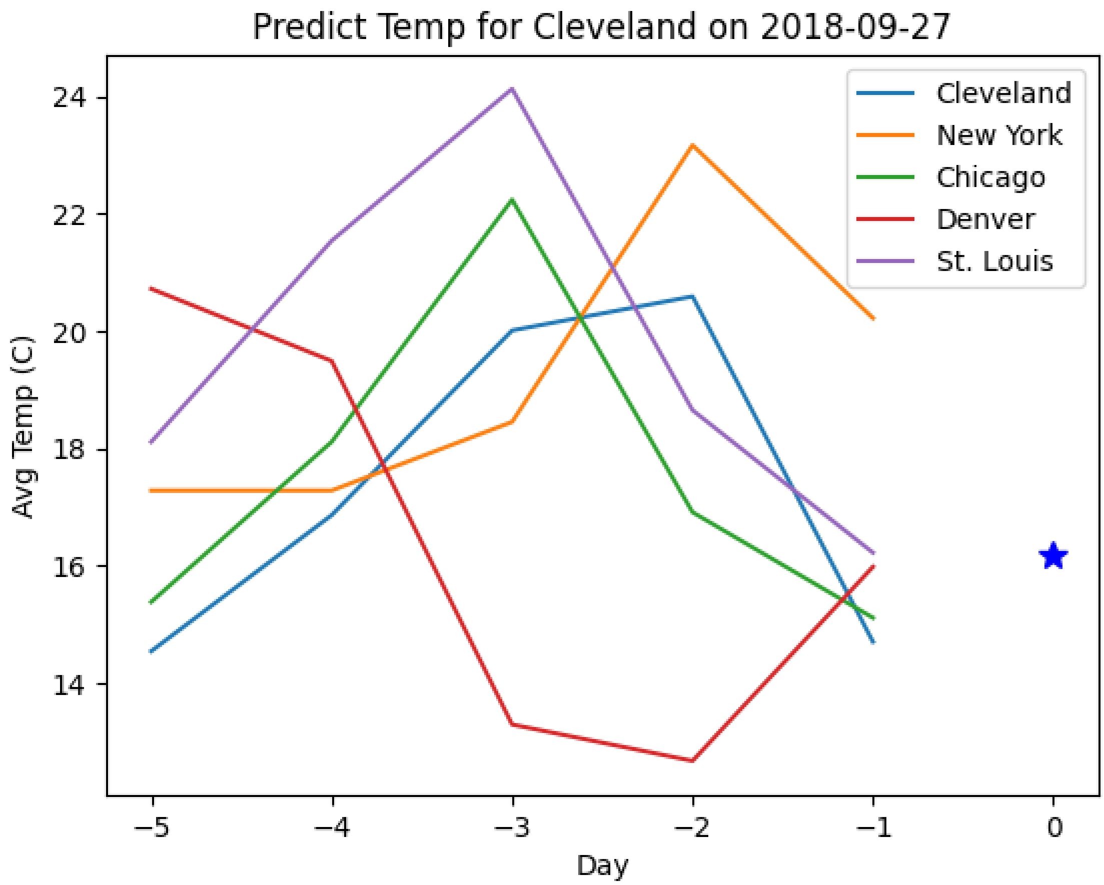
In [ ]:
```
import numpy as np
# from google.colab import drive
%matplotlib inline
```

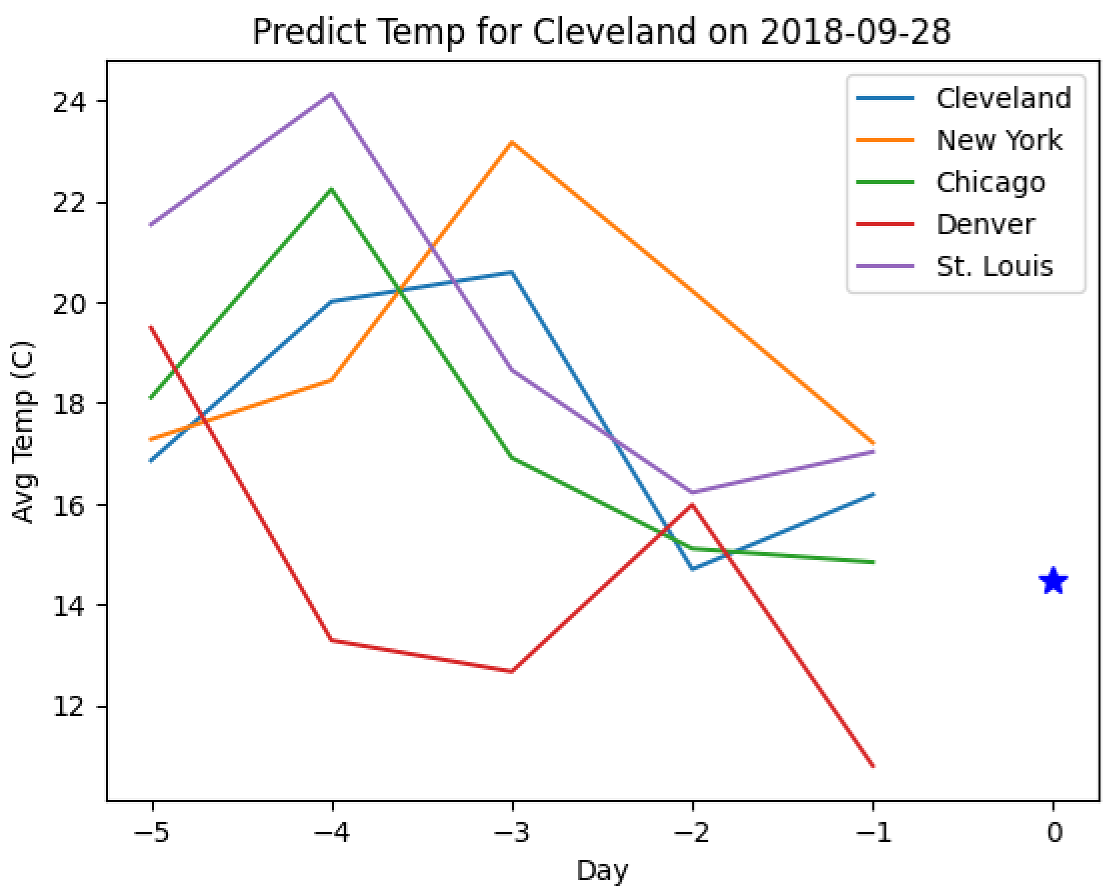
```python
from matplotlib import pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso

# load data (modify to match your data directory or comment)
def load_temp_data():
  # drive.mount('/content/drive')
  # datadir = "/content/drive/My Drive/CS441/24SP/hw1/"
  datadir = "./"
  T = np.load(datadir + 'temperature_data.npz')
  x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates_val, da
  T['x_train'], T['y_train'], T['x_val'], T['y_val'], T['x_test'], T['y_test
  return (x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates

# plot one data point for listed cities and target date
def plot_temps(x, y, cities, feature_to_city, feature_to_day, target_date):
  nc = len(cities)
  ndays = 5
  xplot = np.array([-5,-4,-3,-2,-1])
  yplot = np.zeros((nc,ndays))
  for f in np.arange(len(x)):
    for c in np.arange(nc):
      if cities[c]==feature_to_city[f]:
        yplot[feature_to_day[f]+ndays,c] = x[f]
  plt.plot(xplot,yplot)
  plt.legend(cities)
  plt.plot(0, y, 'b*', markersize=10)
  plt.title('Predict Temp for Cleveland on ' + target_date)
  plt.xlabel('Day')
  plt.ylabel('Avg Temp (C)')
  plt.show()
```

In [ ]:

```python
# load data
(x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates_val, dat
''' Data format:
      x_train, y_train: features and target value for each training sample (
      x_val, y_val: features and target value for each validation sample (us
      x_test, y_test: features and target value for each test sample (used t
      dates_xxx: date of the target value for the corresponding sample
      feature_to_city: maps from a feature number to the city
      feature_to_day: maps from a feature number to a day relative to the ta
      Note: 361 is the temperature of Cleveland on the previous day
'''
f = 361
print('Feature {}: city = {}, day= {}'.format(f,feature_to_city[f], feature_
baseline_rmse = np.sqrt(np.mean((y_val[1:]-y_val[:-1])**2)) # root mean squa
print('Baseline - prediction using previous day: RMSE={}'.format(baseline_rm

# plot first two x/y for val
plot_temps(x_val[0], y_val[0], ['Cleveland', 'New York', 'Chicago', 'Denver'
plot_temps(x_val[1], y_val[1], ['Cleveland', 'New York', 'Chicago', 'Denver'
```

```
Feature 361: city = Cleveland, day= -1
Baseline - prediction using previous day: RMSE=3.460601246750482
```


Predict Temp for Cleveland on 2018-09-27
Cleveland
New York
Chicago
Denver
St. Louis
Avg Temp (C)
24
22
20
18
16
14
−5
−4
−3
−2
−1
0
Day


Predict Temp for Cleveland on 2018-09-28
Cleveland
New York
Chicago
Denver
St. Louis
Avg Temp (C)
24
22
20
18
16
14
12
−5
−4
−3
−2
−1
0
Day

In [ ]:
```
# K-NN Regression
import copy

def regress_KNN(X_trn, y_trn, X_tst, K=1):
  '''
  Predict the target value for each data point in X_tst using a
  K-nearest neighbor regressor based on (X_trn, y_trn), with L2 distance.
  Input: X_trn[i] is the ith training data. y_trn[i] is the ith training lab
  Output: return y_pred, where y_pred[i] is the predicted ith test value
  '''
  # TO DO

  dist, idx = fast_nearest(x_test, x_train, K=K)
  y_pred = np.zeros(shape=X_tst.shape[0])
  for i in range(len(X_tst)):
    points = []
    for j in idx[i]:
      points.append(y_trn[j])
      y_pred[i] = np.mean(points)
  return y_pred


def normalize_features(x, y, fnum):
  ''' Normalize the features in x and y.
      For each data sample i:
        x2[i] = x[i]-x[i,fnum]
        y2[i] = y[i]-x[i,fnum]
  '''
  # TO DO
  x2 = np.zeros_like(x)
  y2 = np.zeros_like(y)
  xsub = np.array(np.copy(x[:,fnum]))
  for i in range(x.shape[1]):
    x2[:,i] = x[:,i] - xsub
  y2 = y - xsub
  return x2, y2



# KNN with original features

# TO DO
y_predict = regress_KNN(x_train, y_train, x_test, K=5)
# print(f"y_predict: {y_val.shape}\nx_test: {x_val.shape}")
print(f"rmse when K=5: {np.sqrt(np.mean((y_test-y_predict)**2))}")
# KNN with normalized features
fnum = 361 # previous day temp in Cleveland

# TO DO
x_train_new, y_train_new = normalize_features(x_train, y_train, fnum)
x_test_new, y_test_new = normalize_features(x_test, y_test, fnum)
y_predict_new = regress_KNN(x_train_new, y_train_new, x_test_new, K=5)
print(f"rmse when K=5: {np.sqrt(np.mean((y_test_new-y_predict_new)**2))}")
```

```
rmse when K=5: 3.249556245363484
rmse when K=5: 2.9770334300836487
```

Part 5: Stretch Goals

Include all your code used for part 5 in this section. You can copy-paste code from parts 1-3 if it is re-usable.

In [ ]:
```python
# Stretch: KNN classification (Select K)

# find the prediction for a set
def calc_pred(valid, dist, idx, y_train, K):
  pred = np.zeros(valid.shape[0])
  for i in range(valid.shape[0]):
    count = {}
    distance = {}
    for j in range(K):            #for each of K points, p is predicted numb
      p = y_train[idx[i][j]]
      d = dist[i][j]
      if p in distance:
        distance[p] += d
      else:
        distance[p] = d
      if p in count:
        count[p] += 1
      else:
        count[p] = 1
    maxnum = -math.inf      #find all candidates with highest votes
    candidate = []
    for c in count:
      if count[c]>maxnum:
        maxnum = count[c]
        candidate = [c]
      elif count[c]==maxnum:
        candidate.append(c)
    mindist = math.inf
    result = -1
    for can in candidate:
      if distance[can]<mindist:
        result = can
    pred[i] = result
  return pred


def KNN_classification(x_train, y_train, K):
  train = x_train[:50000]
  valid = x_train[50000:]
  answer = y_train[50000:]
  dist, idx = fast_nearest(valid, train, K=K)
  correct = 0
  pred = calc_pred(valid, dist, idx, y_train, K)
  for i in range(len(valid)):
    if pred[i]==answer[i]:
      correct += 1
```

```
  print(f"incorrect rate for K={K}: {100*(1-correct/len(valid))}%")


(x_train, y_train), (x_test, y_test) = load_mnist()
hyperparameter = [1, 3, 5, 11, 25]

for K in hyperparameter:
  KNN_classification(x_train, y_train, K)

# Perform K=3 on the test set
best_K = 3
dist, idx = fast_nearest(x_test, x_train, K=best_K)
correct = 0
pred = calc_pred(x_test, dist, idx, y_train, best_K)
for i in range(len(x_test)):
  if pred[i]==y_test[i]:
    correct += 1
print(f"incorrect rate for K={best_K}: {100*(1-correct/len(x_test))}%")
```

```
incorrect rate for K=1: 2.880000000000005%
incorrect rate for K=3: 2.8200000000000003%
incorrect rate for K=5: 2.949999999999997%
incorrect rate for K=11: 3.1499999999999972%
incorrect rate for K=25: 3.8799999999999946%
incorrect rate for K=3: 2.959999999999996%
```

In [ ]:

```
# Stretch: KNN regression (Select K)
(x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates_val, dat
# K-NN Regression
hyperparameter = [1, 3, 5, 11, 25]

for K in hyperparameter:
  # KNN with original features

  y_predict = regress_KNN(x_train, y_train, x_val, K=K)
  print(f"rmse when K={K}: {np.sqrt(np.mean((y_val-y_predict)**2))}")
  # KNN with normalized features
  fnum = 361 # previous day temp in Cleveland
  x_train_new, y_train_new = normalize_features(x_train, y_train, fnum)
  x_val_new, y_val_new = normalize_features(x_val, y_val, fnum)
  y_predict_new = regress_KNN(x_train_new, y_train_new, x_val_new, K=K)
  print(f"(normalized)rmse when K={K}: {np.sqrt(np.mean((y_val_new-y_predict
```

```
rmse when K=1: 5.994830467058024
(normalized)rmse when K=1: 4.962816257477145
rmse when K=3: 5.0719820622291065
(normalized)rmse when K=3: 4.081979762608333
rmse when K=5: 4.814445285619017
(normalized)rmse when K=5: 3.7840627858326146
rmse when K=11: 4.635825782051689
(normalized)rmse when K=11: 3.6293441508410265
rmse when K=25: 4.482832870675747
(normalized)rmse when K=25: 3.544085254919383
```

In [ ]:
```
# Best on test set
y_predict = regress_KNN(x_train, y_train, x_test, K=25)
print(f"rmse when K={K}: {np.sqrt(np.mean((y_test-y_predict)**2))}")
```

```
rmse when K=25: 3.041778206247129
```

In [ ]:
```
# Stretch: K-means (more iters vs redos)
def test(niter, nredo):
  kmeans = faiss.Kmeans(x_train.shape[1], 30, niter=niter, nredo=nredo, seed
  kmeans.train(x_train)
  dist, idx = kmeans.index.search(x_train, 1)
  rmse = np.sqrt(np.sum(dist) / x_train.shape[0])
  return rmse
(x_train, y_train), (x_test, y_test) = load_mnist()
first_trail_multiple = []
first_trail_single = []
second_trail_multiple = []
second_trail_single = []
for i in range(5):
  first_trail_multiple.append(test(10,5))
  first_trail_single.append(test(50,1))
  second_trail_multiple.append(test(4,5))
  second_trail_single.append(test(20,1))
print(f"for first_trail_multiple: mean={np.mean(first_trail_multiple)}, std=
print(f"for first_trail_single: mean={np.mean(first_trail_single)}, std={np.
print(f"for second_trail_multiple: mean={np.mean(second_trail_multiple)}, st
print(f"for second_trail_single: mean={np.mean(second_trail_single)}, std={r
```

```
for first_trail_multiple: mean=5.789932893969209, std=0.0
for first_trail_single: mean=5.769123994738427, std=0.0
for second_trail_multiple: mean=5.819028949632977, std=0.0
for second_trail_single: mean=5.775398976117003, std=0.0
```

In [ ]:
```
# from https://gist.github.com/jonathanagustin/b67b97ef12c53a8dec27b343dca4a
# For use in Colab.  For local, just use jupyter nbconvert directly

import os
# @title Convert Notebook to PDF. Save Notebook to given directory
NOTEBOOKS_DIR = "/content/drive/My Drive/CS441/24SP/hw1" # @param {type:"str
NOTEBOOK_NAME = "CS441_SP24_HW1_Solution.ipynb" # @param {type:"string"}
#------------------------------------------------------------------------
from google.colab import drive
drive.mount("/content/drive/", force_remount=True)
NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"
assert os.path.exists(NOTEBOOK_PATH), f"NOTEBOOK NOT FOUND: {NOTEBOOK_PATH}"
!apt install -y texlive-xetex texlive-fonts-recommended texlive-plain-generi
!jupyter nbconvert "$NOTEBOOK_PATH" --to pdf > /dev/null 2>&1
NOTEBOOK_PDF = NOTEBOOK_PATH.rsplit('.', 1)[0] + '.pdf'
assert os.path.exists(NOTEBOOK_PDF), f"ERROR MAKING PDF: {NOTEBOOK_PDF}"
print(f"PDF CREATED: {NOTEBOOK_PDF}")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[30], line 9
      7 NOTEBOOK_NAME = "CS441_SP24_HW1_Solution.ipynb" # @param {type:"stri
ng"}
      8 #------------------------------------------------------------------
-----------#
----> 9 from google.colab import drive
     10 drive.mount("/content/drive/", force_remount=True)
     11 NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"

ModuleNotFoundError: No module named 'google.colab'
```