Name:

Jeffrey Hui

Netid:

jhui8

CS 441 - HW1: Instance-based Methods

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

Total Points Available **[ ] / 145**

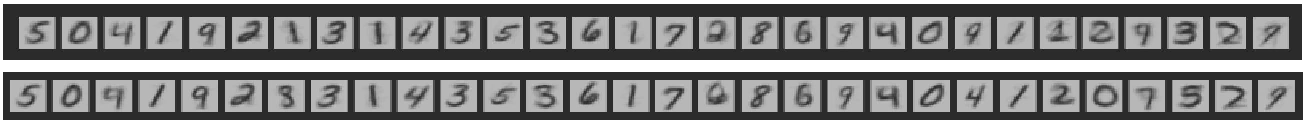
1. Retrieval, K-means, 1-NN on MNIST
    a. Retrieval [ ] / 5
    b. K-means [ ] / 15
    c. 1-NN [ ] / 10
2. Make it fast
    a. K-means plot [ ] / 15
    b. 1-NN error plots [ ] / 8
    c. 1-NN time plots [ ] / 7
    d. Most confused label [ ] / 5
3. Temperature Regression
    a. RMSE Tables [ ] / 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.

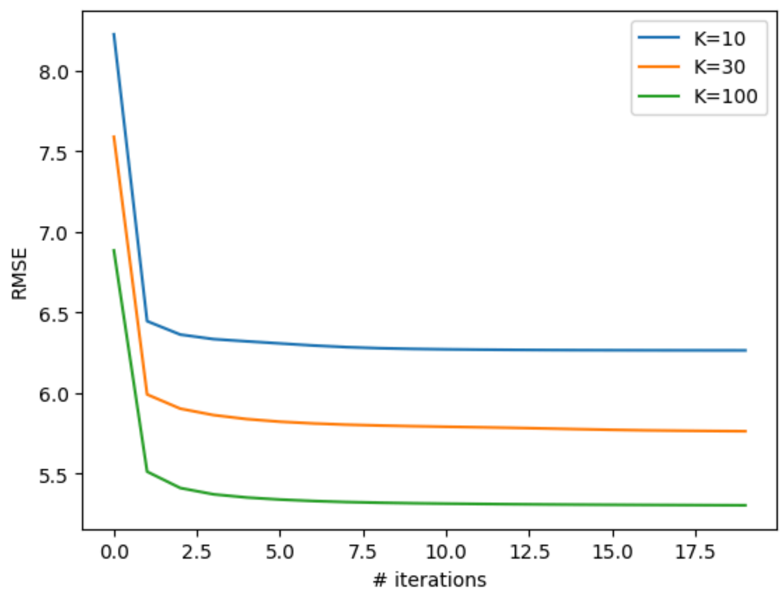


c. Error rate for first 100 test samples, using first 10,000 training samples (x.x)

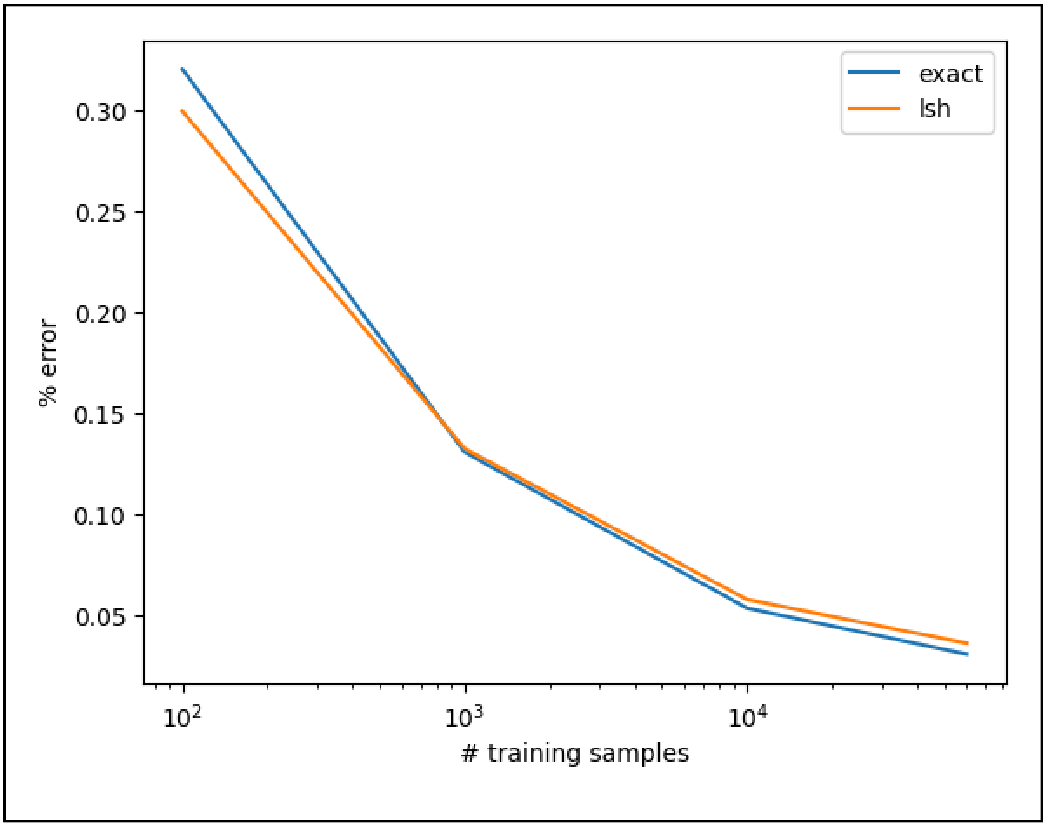
8.0

2. Make it fast

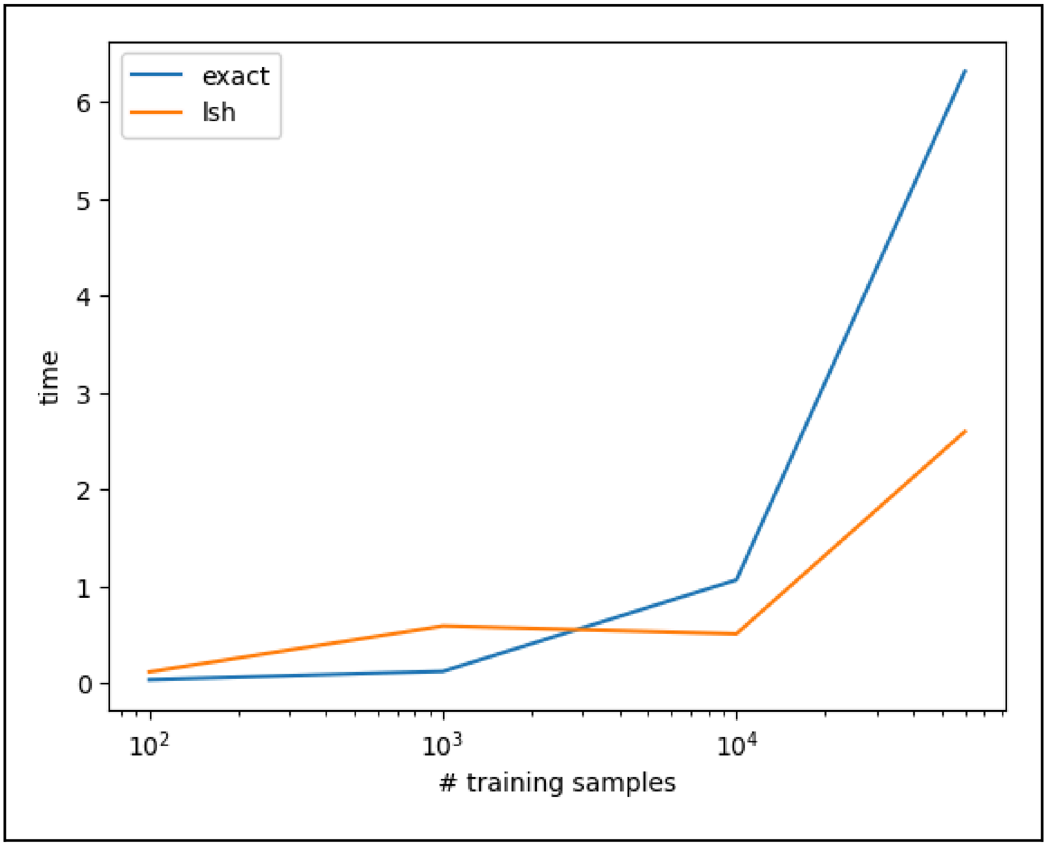
a. KMeans plot of RMSE vs iterations for K=10, 30, 100



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’?

7

3. Temperature Regression

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

	KNN (K=5)
Original Features	3.25
Normalized Features	2.93

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

b

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

a

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	2.80	2.82	3.08	3.82

Best K:

3

Test % error (x.xx)

2.95

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					
Normalized Features					

Best Setting (K, feature type):

Test RMSE (x.xx)

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	5.7912	0.0114
4 iterations, 5 restarts	5.8302	0.0115
50 iterations, 1 restart	5.7766	0.0029
10 iterations, 5 restarts	5.7894	0.0099

Acknowledgments / Attribution

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

None.

cs441-sp24-hw1-starter

February 6, 2024

0.1 CS441: Applied ML - HW 1

0.1.1 Parts 1-2: MNIST

Include all the code for generating MNIST results below

[1]:
```
# initialization code
import numpy as np
from keras.datasets import mnist
%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 data
  x_train = np.reshape(x_train, (len(x_train), 28*28))  # reformat to 768-d␣
↪vectors
  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()
```

```
WARNING:tensorflow:From D:\CODE\School\cs441\venv\Lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

[2]:
```
# example of using MNIST load, display, indices, and count 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

[3]:
```
# Retrieval

def get_nearest(X_query, X):
  ''' Return the index of the sample in X that is closest to X_query according
      to L2 distance '''
  return np.argmin([np.sum((X_query - x) ** 2) for x in X])

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

```
53843
28882
```

[4]:
```
# K-means
import copy
def kmeans(X, K, niter=10):
  '''
  Starting with the first K samples in X as cluster centers, iteratively assign each
  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 number of iterations
  Output: K cluster centers
```

```
  '''
  # TO DO -- add code to display cluster centers at each iteration also
  clusters = [[] for _ in range(K)]
  cluster_centers = copy.deepcopy(X[:K])

  for i in range(niter):

    # For each x_i, compute the closest cluster it belongs
    for x_i in X:
      nearest_center = get_nearest(x_i, cluster_centers)
      clusters[nearest_center].append(x_i)
    # Update centers
    for j in range(len(cluster_centers)):
      cluster_centers[j] = np.mean(clusters[j], axis=0)

    display_mnist(cluster_centers, 1, K)
    # reset clusters
    clusters = [[] for _ in range(K)]

  return cluster_centers

K=30
centers = kmeans(x_train[:1000], K)
```

[5]:
```python
# 1-NN
def one_nn(X_test, Y_test, X_train, Y_train):
  correct_labels = Y_test

  predictions = [get_nearest(x_t, X_train) for x_t in X_test]
  predictions = np.take(Y_train, predictions, axis=0)

  accuracy = (predictions == correct_labels).sum() / len(X_test)

  return 1 - accuracy


# TO DO
n_test = 100
n_train = 10000
error_rate = one_nn(x_test[:n_test], y_test[:n_test], x_train[:n_train], y_train[:n_train])
print('Error rate: {:0.5f}'.format(error_rate))
```

```
Error rate: 0.08000
```

2. Make it fast

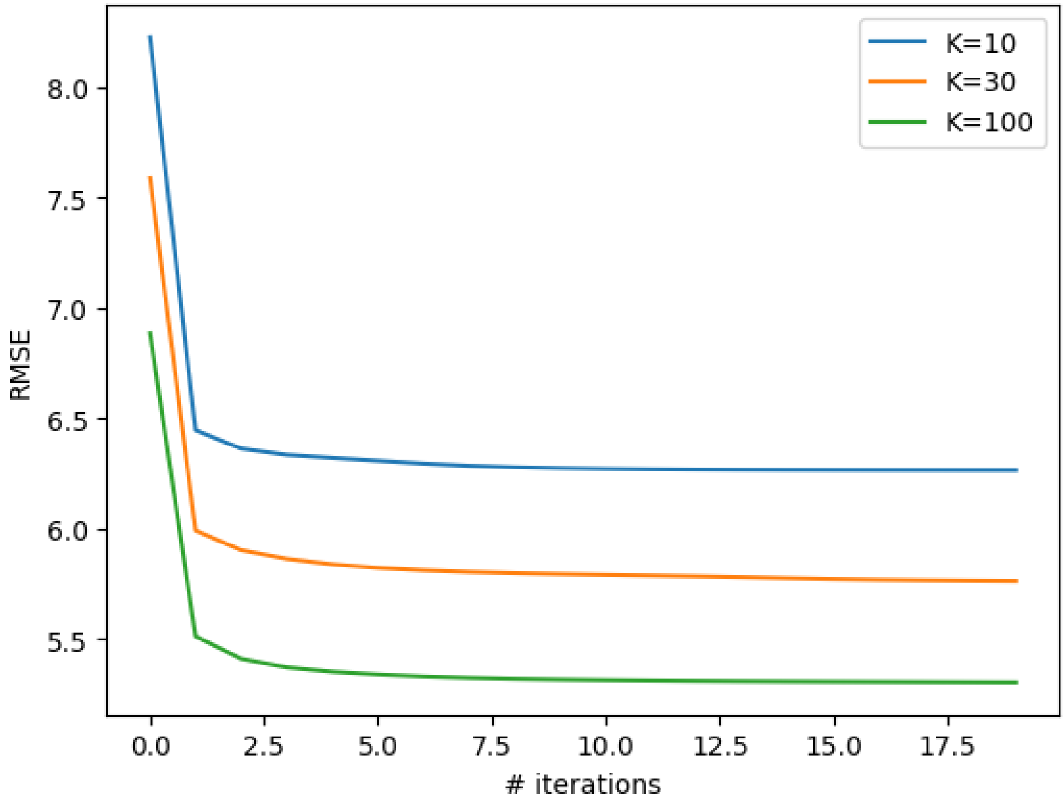
[6]:
```python
# install libraries you need for part 2
import faiss
import time
```

[7]:
```python
# retrieval

# TO DO (check that you're using FAISS correctly)
def get_nearest_faiss(X_query, X):
  index = faiss.IndexFlatL2(X.shape[1])
  index.add(X)
  dist, idx = index.search(X_query,1)

  return dist.item(), idx.item()

j = get_nearest_faiss(x_test[0].reshape(1,-1), x_train)
print(j)
j = get_nearest_faiss(x_test[1].reshape(1,-1), x_train)
print(j)
```

```
(7.039846420288086, 53843)
(20.798309326171875, 28882)
```

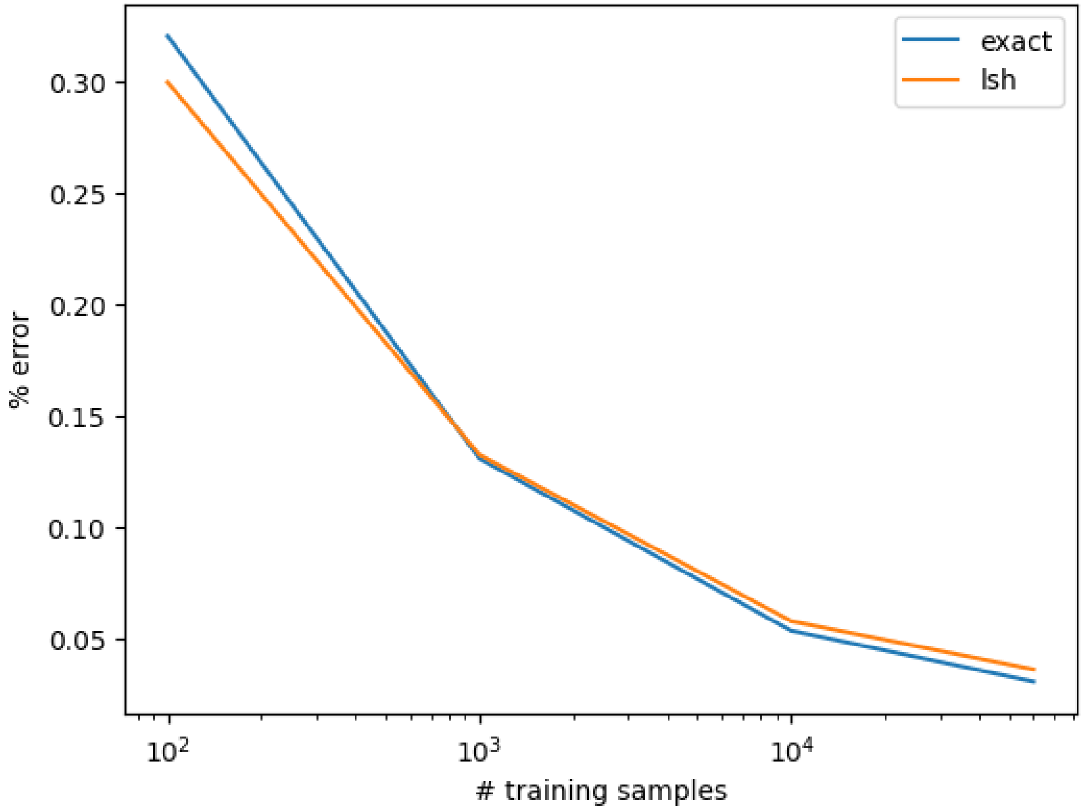
[8]:
```python
# K-means
from collections import defaultdict
def create_index(index_dataset):
  index = faiss.IndexFlatL2(index_dataset.shape[1])
  index.add(index_dataset)
  return index

def find_with_index(index: faiss.IndexFlatL2, query):
  dist, idx = index.search(query,1)
  return dist, idx

def kmeans_fast(X, K, niter=10):
  '''
  Starting with the first K samples in X as cluster centers, iteratively assign␣
↪each
  point to the nearest cluster using faiss and compute the mean of each cluster.
  Input: X[i] is the ith sample, K is the number of clusters, niter is the␣
↪number of iterations
  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 below

  clusters = defaultdict(list)
  distances = []
  cluster_centers = copy.deepcopy(X[:K])

  for i in range(niter):
```

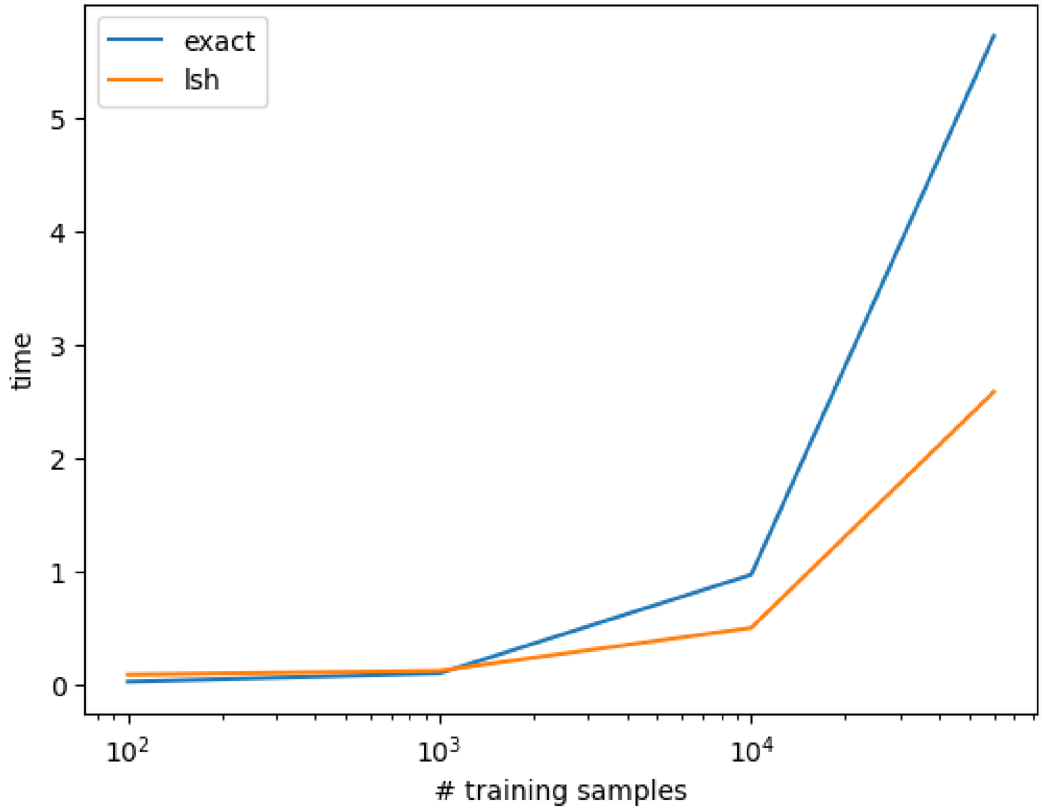
```
    faiss_index = create_index(cluster_centers)
    dist, nearest_center = find_with_index(faiss_index, X)
    dist, nearest_center = dist.flatten(), nearest_center.flatten()
    distances.append(np.sqrt(np.mean(dist)))

    # Populate clusters
    for img_idx, center_idx in enumerate(nearest_center):
      clusters[center_idx].append(X[img_idx])

    # Update centers
    for j in range(len(cluster_centers)):
      cluster_centers[j] = np.mean(clusters[j], axis=0)

    # reset
    clusters = defaultdict(list)

  return cluster_centers, distances

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()
```



[9]:
```
# 1-NN
from time import time
nsample = [100, 1000, 10000, 60000]

calc_acc = lambda actual, prediction: (prediction == actual).sum() /␣
 ↪len(prediction)
get_predictions = lambda indices: np.take(y_train, indices, axis=0)
acc = lambda actual, prediction: calc_acc(actual, get_predictions(prediction))

acc_exact = []
acc_lsh = []

timing_exact = []
timing_lsh = []

exact_predictions = None

# TO DO
for samp in nsample:
  # setup index for exact
  dim = x_train.shape[1]
```

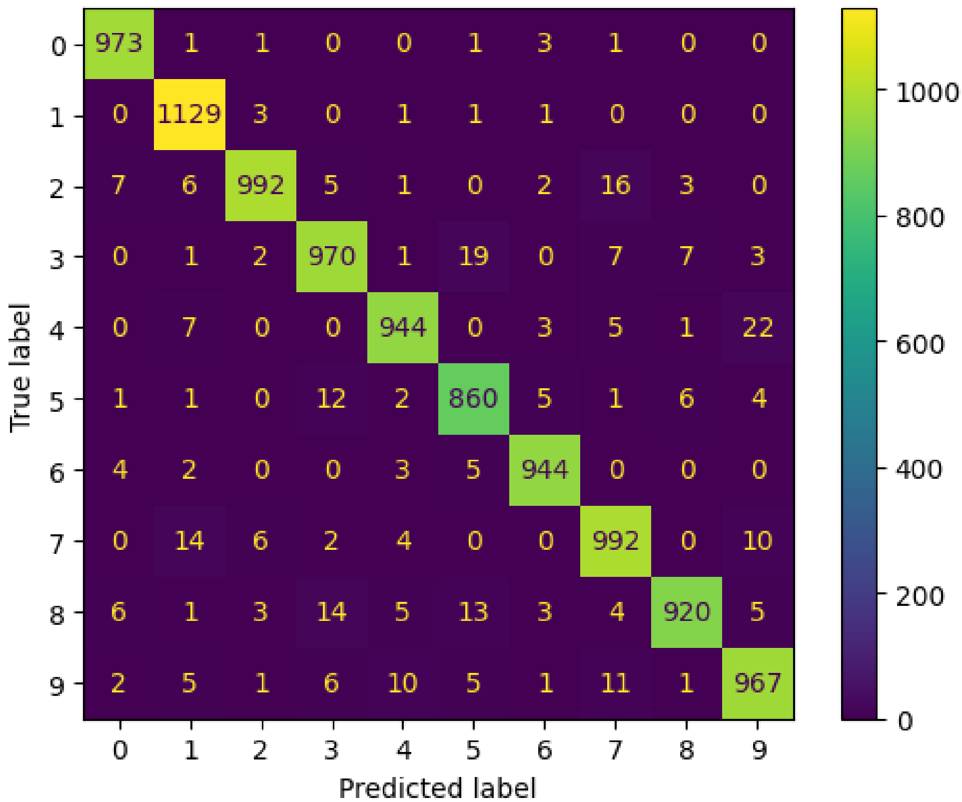
```
    exact_idx = faiss.IndexFlatL2(dim)
    exact_idx.add(x_train[:samp])

    # search for exact
    t1 = time()
    _, idx = exact_idx.search(x_test, 1) # full test set search
    t2 = time()
    timing_exact.append(t2 - t1)
    acc_exact.append(acc(y_test, idx.reshape(-1))) # take out the extra dimension using reshape
    exact_predictions = get_predictions(idx)

    # setup index for lsh
    lsh_idx = faiss.IndexLSH(dim, dim)
    lsh_idx.add(x_train[:samp])

    # search for lsh
    t1 = time()
    _, idx = lsh_idx.search(x_test, 1)
    t2 = time()
    timing_lsh.append(t2 - t1)
    acc_lsh.append(acc(y_test, idx.reshape(-1)))

acc_exact = np.array(acc_exact)
acc_lsh = np.array(acc_lsh)

plt.semilogx(nsample, 1-acc_exact, label='exact')
plt.semilogx(nsample, 1-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()
```


exact
lsh
0.30
0.25
0.20
0.15
0.10
0.05
% error
10^2
10^3
10^4
training samples



[10]:
```
# Confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, exact_predictions)
disp = ConfusionMatrixDisplay(cm)
disp.plot()

# TO DO
```

[10]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x278f2054cd0>
```



0.2 Part 3: Temperature Regression

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

[11]:
```
import numpy as np
%matplotlib inline
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():
  datadir = "./"
  T = np.load(datadir + 'temperature_data.npz')
  x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates_val,␣
 ↪dates_test, feature_to_city, feature_to_day = \
  T['x_train'], T['y_train'], T['x_val'], T['y_val'], T['x_test'], T['y_test'],␣
 ↪T['dates_train'], T['dates_val'], T['dates_test'], T['feature_to_city'],␣
 ↪T['feature_to_day']
  return (x_train, y_train, x_val, y_val, x_test, y_test, dates_train,␣
 ↪dates_val, dates_test, feature_to_city, feature_to_day)
```

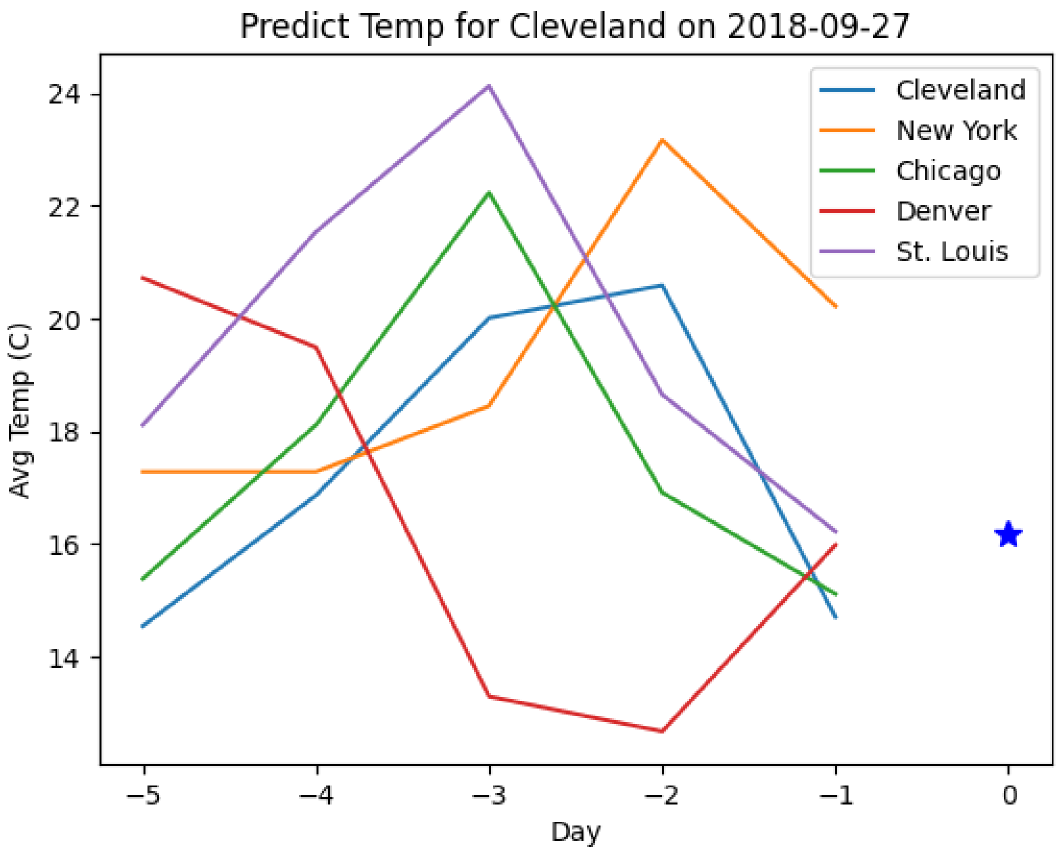
```
# plot one data point for listed cities and target date
def plot_temps(x, y, cities, feature_to_city, feature_to_day, target_date):
  nc = len(cities)
  ndays = 5
  xplot = np.array([-5,-4,-3,-2,-1])
  yplot = np.zeros((nc,ndays))
  for f in np.arange(len(x)):
    for c in np.arange(nc):
      if 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()
```

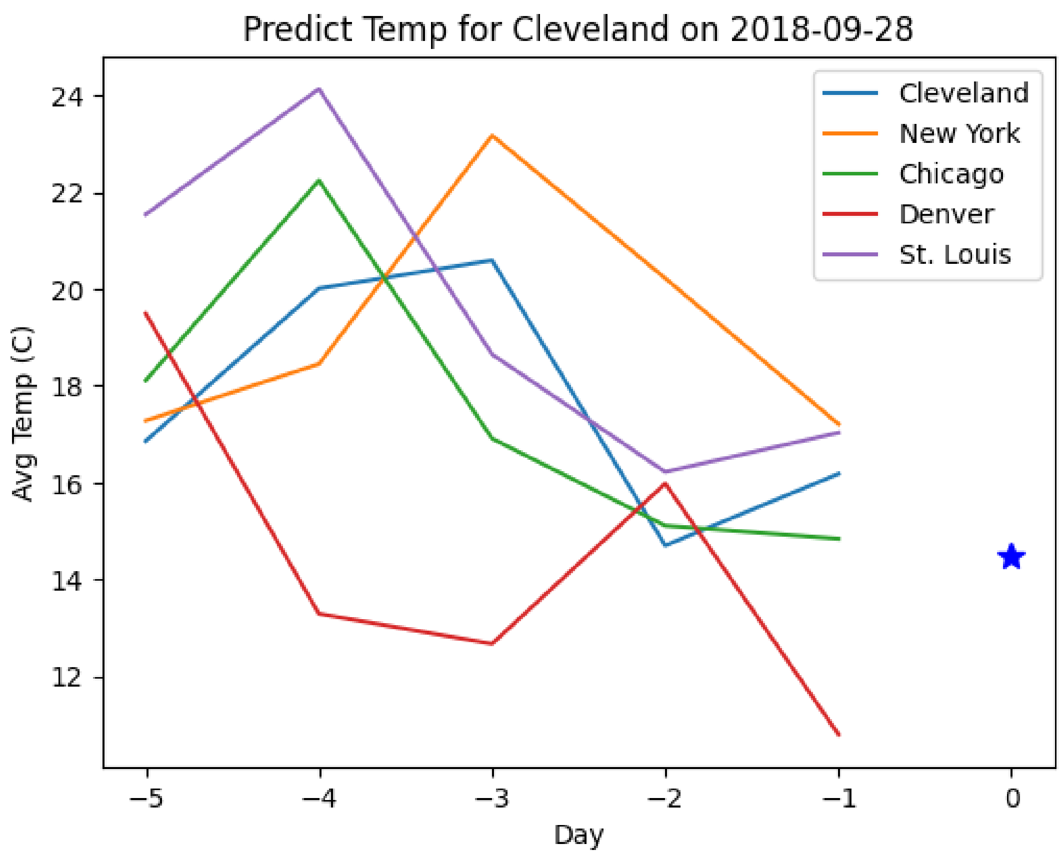
[12]:

```
# load data
(x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates_val,␣
 ↪dates_test, feature_to_city, feature_to_day) = load_temp_data()
''' Data format:
      x_train, y_train: features and target value for each training sample␣
 ↪(used to fit model)
      x_val, y_val: features and target value for each validation sample (used␣
 ↪to select hyperparameters, such as regularization and K)
      x_test, y_test: features and target value for each test sample (used to␣
 ↪evaluate final performance)
      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␣
 ↪target value, e.g. -2 means two days before
      Note: 361 is the temperature of Cleveland on the previous day
'''
f = 361
print('Feature {}: city = {}, day= {}'.format(f,feature_to_city[f],␣
 ↪feature_to_day[f]))
baseline_rmse = np.sqrt(np.mean((y_val[1:]-y_val[:-1])**2)) # root mean squared␣
 ↪error example
print('Baseline - prediction using previous day: RMSE={}'.format(baseline_rmse))

# plot first two x/y for val
plot_temps(x_val[0], y_val[0], ['Cleveland', 'New York', 'Chicago', 'Denver',␣
 ↪'St. Louis'], feature_to_city, feature_to_day, dates_val[0])
```

```
plot_temps(x_val[1], y_val[1], ['Cleveland', 'New York', 'Chicago', 'Denver',␣
 ↪'St. Louis'], feature_to_city, feature_to_day, dates_val[1])
```

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





[13]:
```python
# K-NN Regression

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 label.
    ↪K is the number of closest neighbors to use.
    Output: return y_pred, where y_pred[i] is the predicted ith test value
    '''
    # TO DO
    faiss_index = create_index(X_trn)
    dist, idx = faiss_index.search(X_tst, K)

    # get prediction
    predictions = np.take(y_trn, idx, axis=0)
    predictions = np.mean(predictions, axis=1)

    return predictions
```

```
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
    return (x - np.take(x, fnum, axis=1).reshape(-1, 1)), (y - np.take(x, fnum,␣
 ↪axis=1))



# KNN with original features
res = regress_KNN(x_train, y_train, x_test, 5)
print(np.sqrt(np.mean((res - y_test) ** 2)))

# TO DO

# KNN with normalized features
fnum = 361 # previous day temp in Cleveland

# TO DO
x_train_norm, y_train_norm = normalize_features(x_train, y_train, fnum)
x_test_norm, y_test_norm = normalize_features(x_test, y_test, fnum)
# KNN with normalized features
res = regress_KNN(x_train_norm, y_train_norm, x_test_norm, 5)
print(np.sqrt(np.mean((res - y_test_norm) ** 2)))
```

```
3.249556245363484
2.9324389176041588
```

0.3 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.

[14]:
```
# Stretch: KNN classification (Select K)
from sklearn.neighbors import KNeighborsClassifier

(x_train, y_train), (x_test, y_test) = load_mnist()

p5a_train_x, p5a_train_y = x_train[:50000], y_train[:50000]
p5a_val_x, p5a_val_y = x_train[50000:], y_train[50000:]

# for i in [1,3,5,11,25]:
#   neigh = KNeighborsClassifier(n_neighbors=i)
#   neigh.fit(p5a_train_x, p5a_train_y)
#   print(f'n = {i}, error rate = {1 - neigh.score(p5a_val_x, p5a_val_y)}')
```

```
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(x_train, y_train)
1 - neigh.score(x_test, y_test)
```

[14]: 0.02949999999999997

[15]:
```
# Stretch: KNN regression (Select K)
```

[16]:
```
# Stretch: K-means (more iters vs redos

params = [(20, 1), (4, 5), (50, 1), (10, 5)]

for (iteration, restart) in params:
  rmses = []
  for _ in range(5):
    fkmeans = faiss.Kmeans(x_train.shape[1], 30, niter=iteration,␣
  ↪nredo=restart, seed=int(time()))
    fkmeans.train(x_train)
    dist, idx = fkmeans.index.search(x_train,1)
    rmses.append(np.sqrt(np.sum(dist) / x_train.shape[0]))

  std = np.std(rmses)
  avg = np.mean(rmses)

  print(f'iteration: {iteration}, redos: {restart}, mean: {avg}, std: {std}')
```

```
iteration: 20, redos: 1, mean: 5.79427435672978, std: 0.013635576397397473
iteration: 4, redos: 5, mean: 5.824453619315952, std: 0.0014559432134116084
iteration: 50, redos: 1, mean: 5.780731385540949, std: 0.008345551020031898
iteration: 10, redos: 5, mean: 5.780008829446705, std: 0.003928297103671805
```