The best value of k is k = 3. The RMSE indicated above

========== Question 3.(o) ==========

We try not to scale each feature to have unit variance but having range [0, 1]That is, for each feature, we map value using function y = (x-a)/(b-a)where a is the min value of the feature and b is the max value of the feature

It does not helps much

========== Question 3.(p) ==========

The best model so far is kNN with k = 3, weighted neighbors and scaling

Done and submitted file submission.txt to gradescope.

========== Question 3.(q) ==========

The best naive classifier will assign all points to the most possible class.

The answer is 1/k.

========== Question 3.(s) ==========

The SVM may not work well, since a lot of points mingle with each other.

In other words, they are not linearly separate.

========== Question 3.(t) ==========

SVM Classification

Accuracy: 0.75

Pipeline(memory=None,

steps=[('svm', SVC(C=48.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='linear',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False))])

The accuracy indicated above.

========== Question 3.(u) ==========

SVM Classification modified by adding PCA step and Scaling step

Accuracy: 0.810810810811

Pipeline(memory=None,

steps=[('pca', PCA(copy=True, iterated\_power='auto', n\_components=7, random\_state=None,

svd\_solver='auto', tol=0.0, whiten=False)), ('scale', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('svm', SVC(C=0.02, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='linear',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False))])

The accuracy indicated above. It does improve.

========== Question 3.(v) ==========

SVM Classification with kernel of radial basis function

Accuracy: 0.689189189189

Pipeline(memory=None,

steps=[('svm', SVC(C=98.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False))])

The accuracy indicated above.

========== Question 3.(w) ==========

k Nearest Neighbors Classification

Accuracy: 0.763513513514

Pipeline(memory=None,

steps=[('knn', KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=4, p=2,

weights='distance'))])

The accuracy indicated above.

k Nearest Neighbors Classification with Scaling

Accuracy: 0.77027027027

Pipeline(memory=None,

steps=[('scale', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('knn', KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=4, p=2,

weights='distance'))])

The accuracy indicated above.

Scling helps a little bit, increasing the accuracy from 0.7635 to 0.7703.

========== Question 3.(x) ==========

At 110 responses, the feature numbers for Berkeley are: [33,30,15,57,51,95,44,55,36,57,22,54,31,47,36,52].

Predicted HDI: 0.461942596688

At 162 responses, the feature numbers for Berkeley are: [49,37,19,79,71,139,60,83,55,84,31,83,38,73,41,72].

Predicted HDI: 0.462173616823

At 229 responses, the feature numbers for Berkeley are: [68,46,26,116,98,198,83,115,78,118,39,116,58,99,64,89].

Predicted HDI: 0.461521885868

========== Question 3.(y) ==========

Regarding the sensor location problem, we can use kNN in the same way:Basically, we are given the distances from m sensors, we can treat them as a vector of featuresFor kNN, we don't learn the model but we learn the boundaries (generative model vs discrimimative modelthat is, given a test point, we determined its 'distance' from k training points, then we infer its locationImplementation:1. Create kNN model knn = KNeighborsRegressor()2. Train model knn.fit(X, y)3. Test model and caculate RMSE knn(X\_test)This is the basic model, we expect to tune parameter k, attempt scaling, attempt weighted neighbors, etc.

========== Question 3.(z) ==========

From this problems, I learned that data modelling is so painful and requires a lot of patience.

Basically, we have to try many model, and we have to search for wide ranges of different parametersbefore come up with an acceptable model.For the nature of the problem, it looks like if our data spread out with low correlations coefficient,the kNN method works better than ridge regression or lasso regression.Feedback for the problem author: The problem is very interesting and useful.

"""

The world\_values data set is available online at http://54.227.246.164/dataset/. In the data,

residents of almost all countries were asked to rank their top 6 'priorities'. Specifically,

they were asked "Which of these are most important for you and your family?"

This code and world-values.tex guides the student through the process of training several models

to predict the HDI (Human Development Index) rating of a country from the responses of its

citizens to the world values data. The new model they will try is k Nearest Neighbors (kNN).

The students should also try to understand \*why\* the kNN works well.

"""

from math import sqrt

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import GridSearchCV

from sklearn.neighbors import KNeighborsRegressor

from world\_values\_utils import import\_world\_values\_data

from world\_values\_utils import hdi\_classification

from world\_values\_utils import calculate\_correlations

from world\_values\_utils import plot\_pca

from world\_values\_utils import plot\_pca\_2

from world\_values\_pipelines import ridge\_regression\_pipeline

from world\_values\_pipelines import lasso\_regression\_pipeline

from world\_values\_pipelines import k\_nearest\_neighbors\_regression\_pipeline

from world\_values\_pipelines import k\_nearest\_neighbors\_regression\_scaled\_pipeline

from world\_values\_pipelines import svm\_classification\_pipeline

from world\_values\_pipelines import svm\_classification\_pca\_scale\_pipeline

from world\_values\_pipelines import k\_nearest\_neighbors\_classification\_pipeline

from world\_values\_pipelines import k\_nearest\_neighbors\_classification\_scale\_pipeline

from world\_values\_parameters import regression\_ridge\_parameters

from world\_values\_parameters import regression\_lasso\_parameters

from world\_values\_parameters import regression\_knn\_parameters

from world\_values\_parameters import regression\_knn\_weighted\_parameters

from world\_values\_parameters import classification\_svm\_parameters

from world\_values\_parameters import classification\_svm\_pca\_scale\_parameters

from world\_values\_parameters import classification\_svm\_rbf\_parameters

from world\_values\_parameters import classification\_knn\_parameters

def main():

print("========== Question 3.(a) ==========")

print('Done filling out the "Berkeley F2017 Values Survey"')

print()

print("Predicting HDI from World Values Survey")

print()

# Import Data #

print("Importing Training and Testing Data")

values\_train, hdi\_train, values\_test = import\_world\_values\_data()

# Center the HDI Values #

hdi\_scaler = StandardScaler(with\_std=False)

hdi\_shifted\_train = hdi\_scaler.fit\_transform(hdi\_train)

# Classification Data #

hdi\_class\_train = hdi\_train['2015'].apply(hdi\_classification)

# Data Information #

print('Training Data Count:', values\_train.shape[0])

print('Test Data Count:', values\_test.shape[0])

print()

# Calculate Correlations #

print("========== Question 3.(b)(c) ==========")

correlations = calculate\_correlations(values\_train, hdi\_train)

# PCA #

print("========== Question 3.(d) ==========")

plot\_pca(values\_train, hdi\_train, hdi\_class\_train)

print()

# Regression Grid Searches #

regression\_grid\_searches(training\_features=values\_train,

training\_labels=hdi\_shifted\_train)

print("========== Question 3.(o) ==========")

print("We try not to scale each feature to have unit variance but having range [0, 1]" +

"That is, for each feature, we map value using function y = (x-a)/(b-a)" +

"where a is the min value of the feature and b is the max value of the feature")

mapping = lambda x, a, b: (x-a)/(b-a)

mapped\_values\_train = scaler.fit\_transform(values\_train)

mapped\_values\_test = scaler.transform(values\_test)

knn\_map = KNeighborsRegressor(n\_neighbors=3, weights="distance")

knn\_map.fit(mapped\_values\_train, hdi\_train["2015"])

knn.predict(scaled\_values\_test)

print("It does not helps much")

print()

print("========== Question 3.(p) ==========")

print("The best model so far is kNN with k = 3, weighted neighbors and scaling")

scaler = StandardScaler()

scaled\_values\_train = scaler.fit\_transform(values\_train)

scaled\_values\_test = scaler.transform(values\_test)

knn = KNeighborsRegressor(n\_neighbors=3, weights="distance")

knn.fit(scaled\_values\_train, hdi\_train["2015"])

hdi\_test = knn.predict(scaled\_values\_test)

with open("submission.txt", "w") as f:

for i in hdi\_test:

f.write(str(i)[:6] + "\n")

print("Done and submitted file submission.txt to gradescope.")

print()

print("========== Question 3.(q) ==========")

print("The best naive classifier will assign all points to the most possible class.\n" +

"The answer is 1/k.")

print()

# PCA for Classification#

# print("========== Question 3.(r) ==========")

# plot\_pca\_2(values\_train, hdi\_train, hdi\_class\_train)

# print()

print("========== Question 3.(s) ==========")

print("The SVM may not work well, since a lot of points mingle with each other.\n" +

"In other words, they are not linearly separate.")

print()

# Classification Grid Searches #

classification\_grid\_searches(training\_features=values\_train,

training\_classes=hdi\_class\_train)

print("========== Question 3.(x) ==========")

print("At 110 responses, the feature numbers for Berkeley are: [33,30,15,57,51,95,44,55,36,57,22,54,31,47,36,52].")

berkeley\_110 = [33,30,15,57,51,95,44,55,36,57,22,54,31,47,36,52]

berkeley\_110 = [i / sum(berkeley\_110) for i in berkeley\_110]

scaled\_berkeley\_110 = scaler.transform([berkeley\_110])

print("Predicted HDI: " + str(knn.predict(scaled\_berkeley\_110)[0]))

print()

print("At 162 responses, the feature numbers for Berkeley are: [49,37,19,79,71,139,60,83,55,84,31,83,38,73,41,72].")

berkeley\_162 = [49,37,19,79,71,139,60,83,55,84,31,83,38,73,41,72]

berkeley\_162 = [i / sum(berkeley\_162) for i in berkeley\_162]

scaled\_berkeley\_162 = scaler.transform([berkeley\_162])

print("Predicted HDI: " + str(knn.predict(scaled\_berkeley\_162)[0]))

print()

print("At 229 responses, the feature numbers for Berkeley are: [68,46,26,116,98,198,83,115,78,118,39,116,58,99,64,89].")

berkeley\_229 = [68,46,26,116,98,198,83,115,78,118,39,116,58,99,64,89]

berkeley\_229 = [i / sum(berkeley\_229) for i in berkeley\_229]

scaled\_berkeley\_229 = scaler.transform([berkeley\_229])

print("Predicted HDI: " + str(knn.predict(scaled\_berkeley\_229)[0]))

print()

print("========== Question 3.(y) ==========")

print("Regarding the sensor location problem, we can use kNN in the same way:" +

"Basically, we are given the distances from m sensors, we can treat them as a vector of features" +

"For kNN, we don't learn the model but we learn the boundaries (generative model vs discrimimative model" +

"that is, given a test point, we determined its 'distance' from k training points, then we infer its location" +

"Implementation:" +

"1. Create kNN model knn = KNeighborsRegressor()" +

"2. Train model knn.fit(X, y)" +

"3. Test model and caculate RMSE knn(X\_test)" +

"This is the basic model, we expect to tune parameter k, attempt scaling, attempt weighted neighbors, etc.")

print()

print("========== Question 3.(z) ==========")

print("From this problems, I learned that data modelling is so painful and requires a lot of patience.\n" +

"Basically, we have to try many model, and we have to search for wide ranges of different parameters" +

"before come up with an acceptable model." +

"For the nature of the problem, it looks like if our data spread out with low correlations coefficient,"+

"the kNN method works better than ridge regression or lasso regression." +

"Feedback for the problem author: The problem is very interesting and useful.")

print()

def find\_neighbors(training\_features):

distance\_map = {}

usa = np.array(training\_features.iloc[45])

for i in range(training\_features.shape[0]):

country = np.array(training\_features.iloc[i])

distance\_map[i] = np.mean( (country - usa) \*\* 2 ) \*\* 0.5

usa\_neighbors = []

for \_ in range(8):

index = min(distance\_map, key=distance\_map.get)

distance\_map.pop(index)

usa\_neighbors.append(index)

print("Country indices: " + str(usa\_neighbors[1:]))

def \_rmse\_grid\_search(training\_features, training\_labels, pipeline, parameters, technique):

"""

Input:

training\_features: world\_values responses on the training set

training\_labels: HDI (human development index) on the training set

pipeline: regression model specific pipeline

parameters: regression model specific parameters

technique: regression model's name

Output:

Prints best RMSE and best estimator

Prints feature weights for Ridge and Lasso Regression

Plots RMSE vs k for k Nearest Neighbors Regression

"""

grid = GridSearchCV(estimator=pipeline,

param\_grid=parameters,

scoring='neg\_mean\_squared\_error')

grid.fit(training\_features,

training\_labels)

print("RMSE:", sqrt(-grid.best\_score\_))

print(grid.best\_estimator\_)

# Check Ridge or Lasso Regression

if hasattr(grid.best\_estimator\_.named\_steps[technique], 'coef\_'):

print(grid.best\_estimator\_.named\_steps[technique].coef\_)

else:

# Plot RMSE vs k for k Nearest Neighbors Regression

plt.plot(grid.cv\_results\_['param\_knn\_\_n\_neighbors'],

(-grid.cv\_results\_['mean\_test\_score'])\*\*0.5)

plt.xlabel('k')

plt.ylabel('RMSE')

plt.title('RMSE versus k in kNN')

plt.show()

print()

def regression\_grid\_searches(training\_features, training\_labels):

"""

Input:

training\_features: world\_values responses on the training set

training\_labels: HDI (human development index) on the training set

Output:

Prints best RMSE, best estimator, feature weights for Ridge and Lasso Regression

Prints best RMSE, best estimator, and plots RMSE vs k for k Nearest Neighbors Regression

"""

print("========== Question 3.(e) ==========")

print("Ridge Regression")

\_rmse\_grid\_search(training\_features, training\_labels,

ridge\_regression\_pipeline, regression\_ridge\_parameters, 'ridge')

print("I changed the range of hyper-parameter ridge\_\_alpha to obtain the finer result. That is\n" +

"ridge\_\_alpha has np.arange(0.001, 1.0, 0.001) instead of np.arange(0.01, 1.0, 0.01) given\n" +

"The best RMSE indicated above.")

print()

print("========== Question 3.(f) ==========")

print("Lasso Regression")

\_rmse\_grid\_search(training\_features, training\_labels,

lasso\_regression\_pipeline, regression\_lasso\_parameters, 'lasso')

print("I changed the range of hyper-parameter lasso\_\_alpha to obtain the finer result. That is\n" +

"lasso\_\_alpha has np.arange(0.00001, 0.01, 0.00001) instead of np.arange(0.0001, 0.01, 0.0001) given\n" +

"The best RMSE indicated above.")

print()

print("========== Question 3.(g) ==========")

print("The Lasso Regression does give more 0 weights.\n" +

"That indicates some features do not really matter in this method.")

print()

print("========== Question 3.(h) ==========")

print("To deal with continuous outputs, we can weight the neigbors instead of treating them uniformly.\n" +

"Say, each neighbor is weighted by its inverse distance, and we use the average of k-neareast-neighbor\n" +

"weights to predict the output.")

print()

print("========== Question 3.(i) ==========")

print("The 7 nearest neighbors of the USA:")

find\_neighbors(training\_features)

print("Countries: Ireland, United Kingdom, Belgium, Finland, Malta, Austria, France")

print()

print("========== Question 3.(j) ==========")

print("k Nearest Neighbors Regression")

\_rmse\_grid\_search(training\_features, training\_labels,

k\_nearest\_neighbors\_regression\_pipeline,

regression\_knn\_parameters, 'knn')

print("The best value of k is k = 12. The RMSE indicated above")

print()

print("========== Question 3.(k) ==========")

print("When we increase k, the model goes from overfitting to fitting then underfitting. That is increasing k leads\n" +

"to the increase of bias and decrease of variance because the model becomes less flexible to accommodate ambiguous\n" +

"points. Since more points are taken into account when considering a test point, the model works more consistent\n" +

"(with lower variance), but less accurate (with higher bias) if the nature of training data is the mingling of data points.")

print()

print("========== Question 3.(l) ==========")

print("k Nearest Neighbors Regression with weighted neighbor distances")

\_rmse\_grid\_search(training\_features, training\_labels,

k\_nearest\_neighbors\_regression\_pipeline,

regression\_knn\_weighted\_parameters, 'knn')

print("The best value of k is k = 14. The RMSE indicated above")

print()

print("========== Question 3.(m) ==========")

scaler = StandardScaler()

scaled\_training\_features = scaler.fit\_transform(training\_features)

scaled\_training\_features = pd.DataFrame(scaled\_training\_features)

find\_neighbors(scaled\_training\_features)

print("Countries: Ireland, United Kingdom, Finland, Belgium, Malta, France, Austria")

print()

print("Compared to (i), the neighbors just change order a little bit.")

print()

print("========== Question 3.(n) ==========")

print("k Nearest Neighbors Regression with weighted neighbor distances and scaling the features")

\_rmse\_grid\_search(training\_features, training\_labels,

k\_nearest\_neighbors\_regression\_scaled\_pipeline,

regression\_knn\_weighted\_parameters, 'knn')

print("The best value of k is k = 3. The RMSE indicated above")

print()

print("========== Question 3.(o) ==========")

print("The best model so far is kNN with k = 3, weighted neighbors and scaling")

scaler = StandardScaler()

scaled\_values\_train = scaler.fit\_transform(values\_train)

scaled\_values\_test = scaler.transform(values\_test)

knn = KNeighborsRegressor(n\_neighbors=3, weights="distance")

knn.fit(scaled\_values\_train, hdi\_train["2015"])

hdi\_test = knn.predict(scaled\_values\_test)

with open("submission.txt", "w") as f:

for i in hdi\_test:

f.write(str(i)[:6] + "\n")

print("Done and submitted file submission.txt to gradescope.")

print()

def \_accuracy\_grid\_search(training\_features, training\_classes, pipeline, parameters):

"""

Input:

training\_features: world\_values responses on the training set

training\_labels: HDI (human development index) on the training set

pipeline: classification model specific pipeline

parameters: classification model specific parameters

Output:

Prints best accuracy and best estimator of classification model

"""

grid = GridSearchCV(estimator=pipeline,

param\_grid=parameters,

scoring='accuracy')

grid.fit(training\_features, training\_classes)

print("Accuracy:", grid.best\_score\_)

print(grid.best\_estimator\_)

print()

def classification\_grid\_searches(training\_features, training\_classes):

"""

Input:

training\_features: world\_values responses on the training set

training\_labels: HDI (human development index) on the training set

Output:

Prints best accuracy and best estimator for SVM and k Nearest Neighbors Classification

"""

print("========== Question 3.(t) ==========")

print("SVM Classification")

\_accuracy\_grid\_search(training\_features, training\_classes,

svm\_classification\_pipeline,

classification\_svm\_parameters)

print("The accuracy indicated above.")

print()

print("========== Question 3.(u) ==========")

print("SVM Classification modified by adding PCA step and Scaling step")

\_accuracy\_grid\_search(training\_features, training\_classes,

svm\_classification\_pca\_scale\_pipeline,

classification\_svm\_pca\_scale\_parameters)

print("The accuracy indicated above. It does improve.")

print()

print("========== Question 3.(v) ==========")

print("SVM Classification with kernel of radial basis function")

\_accuracy\_grid\_search(training\_features, training\_classes,

svm\_classification\_pipeline,

classification\_svm\_rbf\_parameters)

print("The accuracy indicated above.")

print()

print("========== Question 3.(w) ==========")

print("k Nearest Neighbors Classification")

\_accuracy\_grid\_search(training\_features, training\_classes,

k\_nearest\_neighbors\_classification\_pipeline,

classification\_knn\_parameters)

print("The accuracy indicated above.")

print()

print("k Nearest Neighbors Classification with Scaling")

\_accuracy\_grid\_search(training\_features, training\_classes,

k\_nearest\_neighbors\_classification\_scale\_pipeline,

classification\_knn\_parameters)

print("The accuracy indicated above.")

print("Scling helps a little bit, increasing the accuracy from 0.7635 to 0.7703.")

print()

if \_\_name\_\_ == '\_\_main\_\_':

main()