CS 189: Introduction to Machine Learning

Fall 2017

•

Homework 11

•

Solutions by

Jinhong Du

3033483677

Question 1

(a)

Jinhong Du
jaydu@berkeley.edu

(b)

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up. Jinhong Du

Question 2

(a)

Suppose that $f(x) = \sum\limits_{m=1}^{M} \alpha_m k(x, y_m), g(x) = \sum\limits_{s=1}^{S} \beta_s k(x, x_s), h(x) = \sum\limits_{t=1}^{T} \gamma_t k(x, z_t),$

$$\begin{aligned}
< f, g >_H &= \sum_{m=1}^{M} \sum_{s=1}^{S} \alpha_m \beta_s k(y_m, x_s) \\
&= \sum_{s=1}^{S} \sum_{m=1}^{M} \beta_s \alpha_m k(x_s, y_m) \\
&=< g, f >_H \\
< af, g >_H &= \sum_{m=1}^{M} \sum_{s=1}^{S} (a\alpha_m) \beta_s k(y_m, x_s) \\
&= a \sum_{m=1}^{M} \sum_{s=1}^{S} \alpha_m \beta_s k(y_m, x_s) \\
&= a < f, g >_H
\end{aligned}$$

Suppose that $\alpha_i = \gamma_{i-M}$ and $x_i = z_{i-M}$ $(i = M + 1, \cdots , M + T)$.

$$\begin{aligned}
< f + h, g >_H &= \sum_{m=1}^{M+T} \sum_{s=1}^{S} \alpha_m \beta_s k(y_s, x_m) \\
&= \sum_{m=1}^{M} \sum_{s=1}^{S} \alpha_m \beta_s k(y_m, x_s) + \sum_{s=M+1}^{M+T} \sum_{s=1}^{S} \alpha_m \beta_s k(y_s, x_m) \\
&= \sum_{m=1}^{M} \sum_{s=1}^{S} \alpha_m \beta_s k(y_m, x_s) + \sum_{t=1}^{T} \sum_{s=1}^{S} \gamma_t \beta_s k(y_s, z_t) \\
&=< f, g >_H + < h, g >_H \\
< f, f >_H &= \sum_{m=1}^{M} \sum_{s=1}^{M} \alpha_m \alpha_s k(y_m, x_s) \\
&= \alpha^T K \alpha \geqslant 0
\end{aligned}$$

and

$$\begin{aligned}
\alpha^T K \alpha = 0 \quad &\iff \quad \alpha = 0 \\
&\iff \quad f \equiv 0
\end{aligned}$$

where $\alpha = \begin{pmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_M \end{pmatrix}^T$ since K is positive definite.
$\therefore$ the dened inner product is valid.
The norm of f is

$$\|f\|_H = \sqrt{< f, f >_H} = \sqrt{\alpha^T K \alpha}$$

(b)

Let $f_1(t) = k(x,t) = k(t,x),\ f_2(t) = k(y,t) = k(t,y)$, then $f_1, f_2 \in H$.
$\because$

$$
\begin{aligned}
< k(x,\cdot), k(y,\cdot) >_H &=< f_1(t), f_2(t) >_H \\
&= \sum_{m=1}^{1}\sum_{s=1}^{1} 1\cdot 1\cdot k(x,y) \\
&= k(x,y)
\end{aligned}
$$

$\therefore$ the defined inner product has the reproducing property

$$
\begin{aligned}
< k(\cdot,x_i), f >_H &= \sum_{s=1}^{1}\sum_{m=1}^{M} 1\cdot \alpha_m k(x_i,y_m) \\
&= \sum_{m=1}^{M} \alpha_m k(x_i,y_m) \\
&= f(x_i)
\end{aligned}
$$

(c)

Suppose that $M = \{\sum\limits_{n=1}^{N} \alpha_n k(x,x_n) : \alpha_i \in \mathbb{R}\}$ and $f = m + g$ s.t. for some $m \in M$ and some g such that $< m', g >= 0$ for all $m' \in M$.
$\because$ $k(x,x_n) \in M,\ g \in M^{\perp}$
$\therefore$

$$
< k(x,x_n), g >= 0
$$

and

$$
< m, g >= 0
$$

$\therefore$

$$
\begin{aligned}
f(x_i) &=< k(\cdot,x_i), f >_H \\
&=< k(\cdot,x_i), m+g >_H \\
&=< k(\cdot,x_i), m >_H + < k(\cdot,x_i), g >_H \\
&=< k(\cdot,x_i), m >_H
\end{aligned}
$$

$\because$

$$
\begin{aligned}
\|f\|_H^2 &=< f, f > \\
&=< m+g, m+g > \\
&=< m, m > + < g, g > \\
&= \|m\|_H^2 + \|g\|_H^2
\end{aligned}
$$

$\therefore$

$$
\begin{aligned}
\min_{f\in H} \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda\|f\|_H^2 &= \min_{f\in H} \frac{1}{N}\sum_{i=1}^{N} L(y_i, m(x_i)) + \lambda\|m\|_H^2 + \lambda\|g\|_H^2 \\
&\geqslant \min_{f\in H} \frac{1}{N}\sum_{i=1}^{N} L(y_i, m(x_i)) + \lambda\|m\|_H^2
\end{aligned}
$$

Solution (cont.)

i.e. the minimizing solution to the problem is attained when $f \in M$, i.e. $g \equiv 0$.

$\therefore$ the minimizing solution to the problem has the form

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i)$$

(d)

From (c), the minimizing solution to the problem

$$\min_{f \in H} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda \|f\|_H^2$$

has the form

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i)$$

and

$$\|f_H\|_H^2 = \alpha^T K \alpha$$

Therefore, for SVM

$$\frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda \|f\|_H^2 = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \alpha^T K \alpha$$

To minimize kernel SVM through $f \in H$ is equivalent to minimize it through $\alpha \in \mathbb{R}^d$, i.e.

$$\min_{f \in H} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda \|f\|_H^2 = \min_{\alpha \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \alpha^T K \alpha$$

(e)

From (c), the minimizing solution to the problem

$$\min_{f \in H} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda \|f\|_H^2$$

has the form

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i)$$

and

$$\|f_H\|_H^2 = \alpha^T K \alpha$$

Solution (cont.)

Therefore, for ridge regession

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N}L(y_i,f(x_i))+\lambda\|f\|_H^2 &= \frac{1}{N}\sum_{i=1}^{N}\left\|y_i-\sum_{j=1}^{N}\alpha_j k(x_i,x_j)\right\|_2^2+\lambda\alpha^T K\alpha \\ &= \frac{1}{N}\sum_{i=1}^{N}\left[y_i-\sum_{j=1}^{N}\alpha_j k(x_i,x_j)\right]^2+\lambda\alpha^T K\alpha \\ &= \frac{1}{N}\|Y-K\alpha\|_2^2+\lambda\alpha^T K\alpha\end{aligned}$$

To minimize kernel SVM through $f\in H$ is equivalent to minimize it through $\alpha\in\mathbb{R}^d$, i.e.

$$\min_{f\in H}\frac{1}{N}\sum_{i=1}^{N}L(y_i,f(x_i))+\lambda\|f\|_H^2=\min_{\alpha\in\mathbb{R}^d}\frac{1}{N}\|Y-K\alpha\|_2^2+\lambda\alpha^T K\alpha$$

$\because$ $K=K^T$ is positive definite

$\therefore$ Let

$$\begin{aligned}\frac{\partial}{\partial\alpha}\left(\frac{1}{N}\|Y-K\alpha\|_2^2+\lambda\alpha^T K\alpha\right) &= \frac{\partial}{\partial\alpha}\left[\frac{1}{N}(Y-K\alpha)^T(Y-K\alpha)+\lambda\alpha^T K\alpha\right] \\ &= \frac{\partial}{\partial\alpha}\left[\frac{1}{N}(Y^TY-2Y^TK\alpha+\alpha^TK^TK\alpha)+\lambda\alpha^T K\alpha\right] \\ &= \frac{1}{N}(2K^TK\alpha-2K^TY)+\lambda(K+K^T)\alpha \\ &= \frac{1}{N}(2KK\alpha-2KY)+2\lambda K\alpha \\ &= 0\end{aligned}$$

we have

$$\alpha=(K+\lambda NI_N)^{-1}K^{-1}KY=(K+\lambda NI_N)^{-1}Y$$

(f)

$\because$

$$\begin{aligned}k(x_i,x_j) &= (1+x_i^Tx_j)^2 \\ &= (1+x_i^{(1)}x_j^{(1)}+x_i^{(2)}x_j^{(2)})^2 \\ &= 1+x_i^{(1)2}x_j^{(1)2}+x_i^{(2)2}x_j^{(2)2}+2+x_i^{(1)}x_j^{(1)}+2x_i^{(2)}x_j^{(2)}+2x_i^{(1)}x_j^{(1)}x_i^{(2)}x_j^{(2)} \\ &=<\begin{pmatrix}1 & \sqrt{2}x_i^{(1)} & \sqrt{2}x_i^{(2)} & \sqrt{2}x_i^{(1)}x_i^{(2)} & x_i^{(1)2} & x_i^{(2)2}\end{pmatrix}^T, \\ &\quad\ \begin{pmatrix}1 & \sqrt{2}x_j^{(1)} & \sqrt{2}x_j^{(2)} & \sqrt{2}x_j^{(1)}x_j^{(2)} & x_j^{(1)2} & x_j^{(2)2}\end{pmatrix}^T> \\ &=<\Phi(x_i),\Phi(x_j)> \\ &= \Phi(x_i)^T\Phi(x_j)\end{aligned}$$

$\therefore$ define the polynomial features of sample ponits as

$$\psi(x)=\begin{pmatrix}1 & x^{(1)} & x^{(2)} & x^{(1)}x^{(2)} & x^{(1)2} & x^{(2)2}\end{pmatrix}$$

Solution (cont.)

we have

$$\begin{pmatrix} \Phi(x_1)^T \\ \Phi(x_2)^T \\ \vdots \\ \Phi(x_N)^T \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \phi(x_1)^T \\ \phi(x_2)^T \\ \vdots \\ \phi(x_N)^T \end{pmatrix}$$

$$\Phi = \phi D$$

$\therefore$

$$\begin{aligned} K &= \Phi\Phi^T \\ &= \phi D^2 \phi^T \\ &= X' X'^T \\ X &= \phi \end{aligned}$$

we have

$$K = X' X'^T$$

Given new data $x = \begin{pmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(N)} \end{pmatrix}$, we first calculate the polynomial features of x as $\Phi(x)$, then we have

$$\begin{aligned} \hat{y} &= \begin{pmatrix} k(x, x_1) & k(x, x_2) & \cdots & k(x, x_N) \end{pmatrix} \alpha \\ &= \begin{pmatrix} \Phi(x)^T\Phi(x_1) & \Phi(x)^T\Phi(x_2) & \cdots & \Phi(x)^T\Phi(x_N) \end{pmatrix} (K + \lambda N I_N)^{-1} Y \\ &= \Phi(x)^T X'^T (X' X'^T + \lambda N I_N)^{-1} Y \\ &= \Phi(x)^T (\lambda N I_6 + X'^T X')^{-1} X'^T Y \\ &= \phi(x)^T D (\lambda N I_6 + D X^T X D)^{-1} D X^T Y \\ &= \phi(x)^T (\lambda N D^{-2} + X^T X)^{-1} X^T Y \end{aligned}$$

Compare it with the prediction of Tikhonov regularization

$$w = \phi(x)^T (X^T X + \Gamma^T \Gamma)^{-1} X^T Y$$

we have

$$\Gamma^T \Gamma = \lambda N D^{-2}$$

So when $d = 2$, for second-order polynomial regression with kernel,

$$\begin{aligned} \Gamma &= \sqrt{\lambda N} D^{-1} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{\frac{\lambda N}{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{\frac{\lambda N}{2}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{\frac{\lambda N}{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

(g)

Let $C = \binom{p+d}{d}$ denote the number of features.

(1) Train

To find w,

$O(C^2N)$ to multiply X^T by X

$O(CN)$ to multiply X^T by Y

$O(N^2)$ to multiply Γ^T by Γ

$O(C^3)$ to compute the LU (or Cholesky) factorization of $X^TX + \Gamma^T\Gamma$ and use that to compute the product $(X^TX + \Gamma^T\Gamma)^{-1}(X^TY)$

Totally, $O(CN(C+N) + C^3)$.

To find α,

$O(dN^2 + p)$ to compute all the p-degree polynomial kernel $k(x_i, x_j) = (c + x_i^T x_j)^p$

$O(N^3)$ to compute the LU (or Cholesky) factorization of $K + \lambda N I_N$ and use that to compute the product $(K + \lambda N I_N)^{-1}(X^TY)$

Totally, $O(dN^2 + N^3)$ (we can consider p is less than N)

(2) Predict, given new data vector x,

For w,

$O(C)$ to compute polynomial features $\Phi(x)$;

$O(C^2)$ to multiply x by w

Totally, $O(C^2)$.

For α,

$O(dN + p)$ to compute all $k(x, x_i)$

$O(N)$ to multiply $k(x, x_i)$ with α_i for all i.

Totally, $O(dN)$ (we can consider p is less than N).

Question 3

(a)

Have filled out the survey.

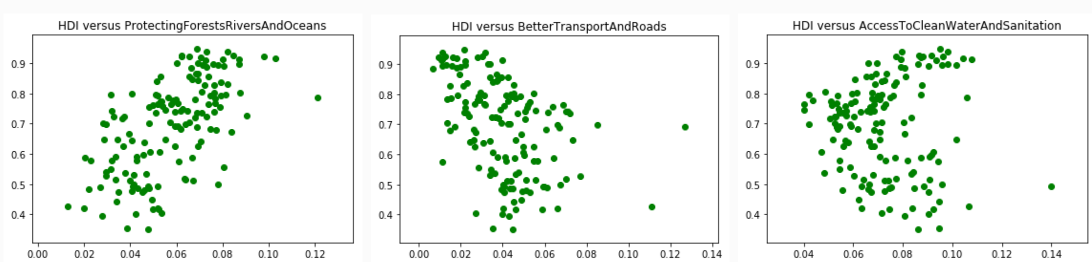
(b)

‘Protecting forests rivers and oceans’ is the most positively correlated with HDI.
‘Better transport and roads’ is the most negatively correlated with HDI.
‘Access to clean water and sanitation’ is the least correlated with HDI (closest to 0).

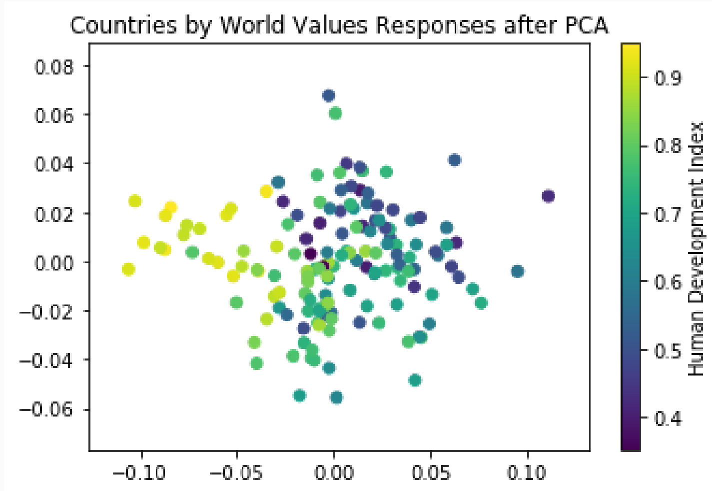
```
Action taken on climate change 0.473312891543
Better transport and roads -0.439633638622
Support for people who can't work -0.336213236721
Access to clean water and sanitation -0.018169084456
Better healthcare -0.422012359959
A good education -0.303978889772
A responsive government we can trust 0.329445314984
Phone and internet access -0.351604712158
Reliable energy at home -0.285423563836
Affordable and nutritious food 0.195193300786
Protecting forests rivers and oceans 0.613458756271
Protection against crime and violence 0.14331869918
Political freedoms 0.238099006821
Freedom from discrimination and persecution 0.432932375445
Equality between men and women 0.276496043498
Better job opportunities -0.39734452674
[0.4733, -0.43959999999999999, -0.3362, -0.018200000000000001, -0.42199999999999999, -0.30399999999999999,
0.32940000000000003, -0.35160000000000002, -0.28539999999999999, 0.19520000000000001, 0.61350000000000005,
0.14330000000000001, 0.23810000000000001, 0.43290000000000001, 0.27650000000000002, -0.39729999999999999]
```

(c)

For the most positively correlated feature, the label and feature seem to have linear relationship $y = kx + b$ where $k > 0$.
For the most negatively correlated feature, the label and feature seem to have linear relationship $y = kx + b$ where $k < 0$.
For the least correlated feature, the label and feature seem to have no linear relationship.



(d)



(e)

The best RMSE is 0.12303337350607801 for Ridge Regression with parameter 0.02.

```
Ridge Regression
RMSE: 0.12303337350607801
Pipeline(steps=[('ridge', Ridge(alpha=0.02, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001))])
[[ 0.80823467 -0.74985758 -0.17800015 -1.28408103 -0.66293176 -0.82203172
   0.73733884 -0.92891581 -0.82049672  0.39614952  2.0708291  -0.06718981
   0.48310656  0.72671425  0.42921192 -0.13808023]]
```

(f)

The best RMSE is 0.12602242808947522 for Lasso Regression with parameter 0.0002.

```
Lasso Regression
RMSE: 0.12602242808947522
Pipeline(steps=[('lasso', Lasso(alpha=0.00020000000000000001, copy_X=True, fit_intercept=True,
   max_iter=1000, normalize=False, positive=False, precompute=False,
   random_state=None, selection='cyclic', tol=0.0001, warm_start=False))])
[ 0.1590192  -0.72844929 -0.         -0.85945074 -0.66274144 -0.02556703
  0.33904781 -0.29897158 -0.          0.          3.48536375  0.          0.
  0.87057995  0.32897045 -0.        ]
```

(g)

Yes, Lasso Regression gives 6 zero weights and it is more than Ridge Regression.

(h)

1.Compute the Euclidean or Mahalanobis distance from the query example to the labeled examples.
2.Order the labeled examples by increasing distance.
3.Find a heuristically optimal number k of nearest neighbors, based on RMSE. This is done using cross validation.
4.Calculate an inverse distance weighted average with the k-nearest multivariate neighbors.
Use the training label to train a regression model between the label and the inverse distance weights.

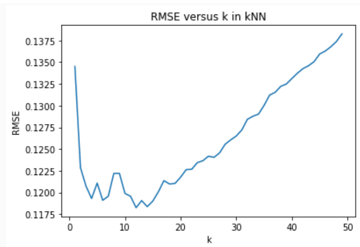
(i)

```
90             Ireland
61      United Kingdom
37             Belgium
108            Finland
69               Malta
132            Austria
110             France
```

(j)

The best RMSE is 0.1182458946077689 for KNN with number of neighbors equals 12.

```
k Nearest Neighbors Regression
RMSE: 0.1182458946077689
Pipeline(steps=[('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=12, p=2,
          weights='uniform'))])
```



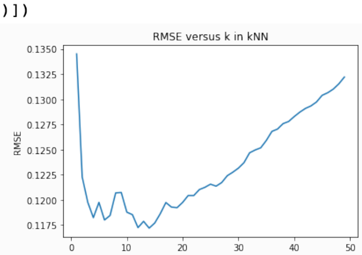
(k)

When k is small, the model tends to find as small neighbors of each point as possible, which may be overfitting. So in such cases, the bias will decrease and the variance will increase as k goes to 0.
When k is big, the model tends to find as many neighbors of each point as possible, which may be underfitting. So in such cases, the bias will increase and the variance will decrease as k goes to ∞ (when the sample size is big enough).

(l)

The best RMSE is 0.1171925270311745 for KNN with number of neighbors equals 14.

```
k Nearest Neighbors Regression
RMSE: 0.1171925270311745
Pipeline(steps=[('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=14, p=2,
          weights='distance'))])
```



(m)

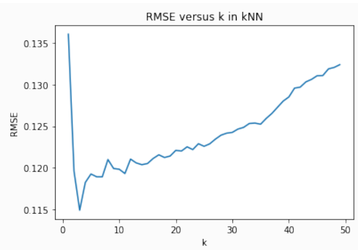
After scaling the features, the nearest neighbors of the USA are very different from (i), which shows that KNN is very sensitive to the scale of the features.

```
107                Qatar
22                Belize
26            Mauritania
94              Slovakia
120            Lithuania
72     Brunei Darussalam
139               Kuwait
```

(n)

The best RMSE is 0.11488547357936414 for scaled KNN with number of neighbors equals 3.

```
k Nearest Neighbors Regression
RMSE: 0.11488547357936414
Pipeline(steps=[('scale', StandardScaler(copy=True, with_mean=True, with_std=True)), ('knn', KNeighborsRegressor(algo
rithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=3, p=2,
          weights='distance'))])
```



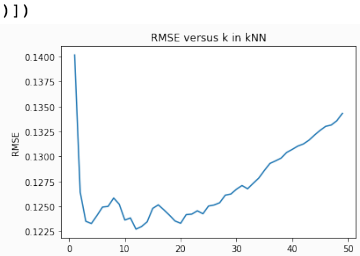
(o)

Here, I use Max-min Standarlization, for each column (feature)

$$z_i = \frac{x_i - \min x_i}{\max x_i - \min x_i}$$

The best RMSE is 0.12270887251692039 for non-uniformly scaled KNN with number of neighbors equals 12. It can prevent overfitting but not improve much.

```
k Nearest Neighbors Regression
RMSE: 0.12270887251692039
Pipeline(steps=[('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=12, p=2,
          weights='distance'))])
```



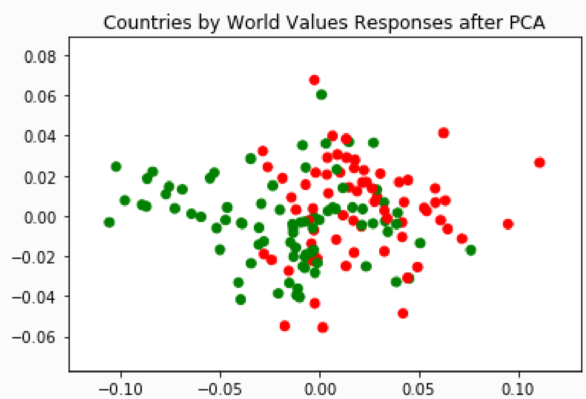
(p)

```
[ 0.52235714  0.6965      0.64778571  0.66564286  0.82007143  0.60928571
  0.63157143  0.61457143  0.7915      0.79892857  0.68364286  0.73614286
  0.73614286  0.69707143  0.61257143  0.71628571  0.73942857  0.6175
  0.66192857  0.66714286  0.59778571  0.7575      0.56157143  0.8275
  0.72828571  0.719       0.57492857  0.91114286  0.6335      0.715
  0.71678571  0.86078571  0.71521429  0.90678571  0.91335714  0.77642857
  0.525       0.78714286]
```

(q)

To use naive classifier to deal with k-class classification, we use k different naive classifiers, the ith of them classifiers all points into the ith class. naive classifiers. So each point can be classified correctly with probability $\frac{1}{k}$ and therefore the accuracy we guaranteed to get with the best naive classifier is $\frac{1}{k}$.

(r)



(s)

I think the linear SVM cannot classify these two class well since they are mixed up. The data seems not to be linear separable.

(t)

```
SVM Classification
Accuracy: 0.75
Pipeline(steps=[('svm', SVC(C=48.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])
```

(u)

The accuracy has been approved up to 0.77027027027.

Solution (cont.)

```
SVM Classification
Accuracy: 0.77027027027
Pipeline(steps=[('pca', PCA(copy=True, iterated_power='auto', n_components=8, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)), ('scale', StandardScaler(copy=True, with_mean=True, with_std=True)),
 ('svm', SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])
```

(v)

The accuracy is 0.689189189189.

```
SVM Classification
Accuracy: 0.689189189189
Pipeline(steps=[('svm', SVC(C=98.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])
```

(w)

The accuracies of raw KNN and scaled KNN are 0.763513513514 and 0.77027027027 respectively. Scaling can improve the the prediction of KNN.

```
k Nearest Neighbors Classification
Accuracy: 0.763513513514
Pipeline(steps=[('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])
```

```
k Nearest Neighbors Classification
Accuracy: 0.77027027027
Pipeline(steps=[('scale', StandardScaler(copy=True, with_mean=True, with_std=True)), ('knn', KNeighborsClassifier(alg
orithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])
```

(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]. The predicted HDI is 0.54592857.

(y)

Treat distances between the object and the sensors as x and the object locations as y, train the kNN regression model since the distance reflects how far the object is from each sensor. Then we can use this model to predict object location given the newly observed distances.

(z)

Scaling or dimension reduction sometimes may help to improve some Machine Learning model but at other times it may be worse. So we need to try more proibility to find out the best model.
Feedback: I think this problem is more clearer than the previous and easy to follow with, which help me kown what is the relationship between all parts instead of mixing them up.

Question 4

Question How to combine SVM with KNN?
Solution A naive version of the SVM-KNN is: for a query

(1) compute distances of the query to all training examples and pick the nearest K neighbors;

(2) if the K neighbors have all the same labels, the query is labeled and exit; else, compute the pairwise distances between the K neighbors;

(3) convert the distance matrix to a kernel matrix and apply multiclass SVM;

(4) use the resulting classifier to label the query.

The naive version of SVM-KNN is slow mainly because it has to compute the distances of the query to all training examples. So we can both compute a crude distance (e.g. L^2 distance) to prune the list of neighbors before the more costly accurate distance computation and cache the pairwise distance matrix in step 2 to speed up the algorithm. The first improvement may be called 'shortlisting'.

(1) Find a collection of K_{sl} neighbors using a crude distance function (e.g. L^2);

(2) Compute the accurate distance function (e.g. tangent distance) on the K_{sl} samples and pick the K nearest neighbors;

(3) Compute (or read from cache if possible) the pairwise accurate distance of the union of the K neighbors and the query;

(4) Convert the pairwise distance matrix into a kernel matrix using the kernel trick;

(5) Apply DAGSVM on the kernel matrix and label the query using the resulting classifier.

HW 11

November 10, 2017

1 Question 3

```
In [1]: from sklearn.pipeline import Pipeline
        from sklearn.preprocessing import StandardScaler
        from sklearn.decomposition import PCA
        from sklearn.preprocessing import PolynomialFeatures
        from sklearn.neural_network import MLPRegressor
        from sklearn.linear_model import Ridge
        from sklearn.linear_model import Lasso
        from sklearn.svm import SVC
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.neighbors import KNeighborsRegressor


        ridge_regression_pipeline = Pipeline(
                [
                    # Apply scaling to Ridge Regression
                    # ('scale', StandardScaler()),
                    ('ridge', Ridge())
                ]
            )

        lasso_regression_pipeline = Pipeline(
                [
                    # Apply scaling to Lasso Regression
                    # ('scale', StandardScaler()),
                    ('lasso', Lasso())
                ]
            )

        k_nearest_neighbors_regression_pipeline = Pipeline(
                [
                    # Apply scaling to k Nearest Neighbors Regression
                    # ('scale', StandardScaler()),
                    ('knn', KNeighborsRegressor())
                ]
            )
```

```
svm_classification_pipeline = Pipeline(
        [
            # Apply PCA to SVM Classification
            # ('pca', PCA()),
            # Apply scaling to SVM Classification
            # ('scale', StandardScaler()),
            ('svm', SVC())
        ]
    )


k_nearest_neighbors_classification_pipeline = Pipeline(
        [
            # Apply scaling to k Nearest Neighbors Classification
            # ('scale', StandardScaler()),
            ('knn', KNeighborsClassifier())
        ]
    )
```

In [2]:
```
import numpy as np

regression_ridge_parameters = {
    'ridge__alpha': np.arange(0.01, 1.0, 0.01)
}

regression_lasso_parameters = {
    'lasso__alpha': np.arange(0.0001, 0.01, 0.0001)
}

regression_knn_parameters = {
    'knn__n_neighbors': np.arange(1, 50),

    # Apply uniform weighting vs k for k Nearest Neighbors Regression
    'knn__weights': ['uniform']

    # Apply distance weighting vs k for k Nearest Neighbors Regression
    # 'knn__weights': ['distance']
}

classification_svm_parameters = {
    # Use linear kernel for SVM Classification
    'svm__kernel': ['linear'],

    # Use rbf kernel for SVM Classification
    # 'svm__kernel': ['rbf'],

    # Original hyperparameters
```

```
    'svm__C': np.arange(1.0, 100.0, 1.0),

    # Original hyperparameters scaled by 1/100
    # 'svm__C': np.arange(0.01, 1.0, 0.01),

    # Hyperparameter search over all possible dimensions for PCA reduction
    # 'pca__n_components': np.arange(1, 17),

    #'svm_gamma': np.arange(0.001, 0.1, 0.001)
}

classification_knn_parameters = {
    'knn__n_neighbors': np.arange(1, 50),

    # Apply distance weighting vs k for k Nearest Neighbors Classification
    'knn__weights': ['distance']
}
```

```
In [3]: import pandas as pd
        from math import sqrt
        from sklearn.decomposition import PCA
        from sklearn.metrics import mean_squared_error
        import matplotlib.pyplot as plt
        import numpy as np


        def import_world_values_data():
            """
            Reads the world values data into data frames.

            Returns:
                values_train: world_values responses on the training set
                hdi_train: HDI (human development index) on the training set
                values_test: world_values responses on the testing set
            """
            values_train = pd.read_csv('world-values-train2.csv')
            values_train = values_train.drop(['Country'], axis=1)
            values_test = pd.read_csv('world-values-test.csv')
            values_test = values_test.drop(['Country'], axis=1)
            hdi_train = pd.read_csv('world-values-hdi-train2.csv')
            hdi_train = hdi_train.drop(['Country'], axis=1)
            return values_train, hdi_train, values_test


        def plot_hdi_vs_feature(training_features, training_labels, feature, color, title):
            """
            Input:
            training_features: world_values responses on the training set
```

```
    training_labels: HDI (human development index) on the training set
    feature: name of one selected feature from training_features
    color: color to plot selected feature
    title: title of plot to display

    Output:
    Displays plot of HDI vs one selected feature.
    """
    plt.scatter(training_features[feature],
    training_labels['2015'],
    c=color)
    plt.title(title)
    plt.show()


def calculate_correlations(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 correlations between HDI and each feature, separately.
        Displays plot of HDI vs one selected feature.
    """
    # Calculate correlations between HDI and each feature
    correlations = []
    for column in training_features.columns:
        print(column, training_features[column].corr(training_labels['2015']))
        correlations.append(round(training_features[column].corr(training_labels['2015']
    print(correlations)
    print()

    # Identify three features
    plot_hdi_vs_feature(training_features, training_labels, 'Protecting forests rivers a
                        'green', 'HDI versus ProtectingForestsRiversAndOceans')
    plot_hdi_vs_feature(training_features, training_labels, 'Better transport and roads'
                        'green', 'HDI versus BetterTransportAndRoads')
    plot_hdi_vs_feature(training_features, training_labels, 'Access to clean water and s
                        'green', 'HDI versus AccessToCleanWaterAndSanitation')

def plot_pca(training_features,
             training_labels,
             training_classes):
    """
    Input:
        training_features: world_values responses on the training set
```

```
            training_labels: HDI (human development index) on the training set
            training_classes: HDI class, determined by hdi_classification(), on the training

        Output:
            Displays plot of first two PCA dimensions vs HDI
            Displays plot of first two PCA dimensions vs HDI, colored by class
        """
        # Run PCA on training_features
        pca = PCA()
        transformed_features = pca.fit_transform(training_features)

        # Plot countries by first two PCA dimensions
        plt.scatter(transformed_features[:, 0],     # Select first column
                    transformed_features[:, 1],     # Select second column
                    c=training_labels)
        plt.colorbar(label='Human Development Index')
        plt.title('Countries by World Values Responses after PCA')
        plt.show()

        # Plot countries by first two PCA dimensions, color by class
        # training_colors = training_classes.apply(lambda x: 'green' if x else 'red')
        # plt.scatter(transformed_features[:, 0],     # Select first column
        #             transformed_features[:, 1],     # Select second column
        #             c=training_colors)
        # plt.title('Countries by World Values Responses after PCA')
        # plt.show()


    def hdi_classification(hdi):
        """
        Input:
            hdi: HDI (human development index) value

        Output:
            high HDI vs low HDI class identification
        """
        if 1.0 > hdi >= 0.7:
            return 1.0
        elif 0.7 > hdi >= 0.30:
            return 0.0
        else:
            raise ValueError('Invalid HDI')
```

In [4]:
```
"""
The world_values data set is available online at http://54.227.246.164/dataset/. In the
    residents of almost all countries were asked to rank their top 6 'priorities'. Speci
    they were asked "Which of these are most important for you and your family?"
```

```python
This code and world-values.tex guides the student through the process of training severa
    to predict the HDI (Human Development Index) rating of a country from the responses
    citizens to the world values data. The new model they will try is k Nearest Neighbor
    The students should also try to understand *why* the kNN works well.
"""

from math import sqrt
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

def _rmse_grid_search(training_features, training_labels, pipeline, parameters, techniqu
    """
    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 Re
    """

    print("Ridge Regression")
    _rmse_grid_search(training_features, training_labels,
                ridge_regression_pipeline, regression_ridge_parameters, 'ridge')

    print("Lasso Regression")
    _rmse_grid_search(training_features, training_labels,
                lasso_regression_pipeline, regression_lasso_parameters, 'lasso')

    print("k Nearest Neighbors Regression")
    _rmse_grid_search(training_features, training_labels,
                k_nearest_neighbors_regression_pipeline,
                regression_knn_parameters, 'knn')


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 Classifi
    """
    print("SVM Classification")
    _accuracy_grid_search(training_features, training_classes,
                          svm_classification_pipeline,
                          classification_svm_parameters)

    print("k Nearest Neighbors Classification")
    _accuracy_grid_search(training_features, training_classes,
                          k_nearest_neighbors_classification_pipeline,
                          classification_knn_parameters)
```

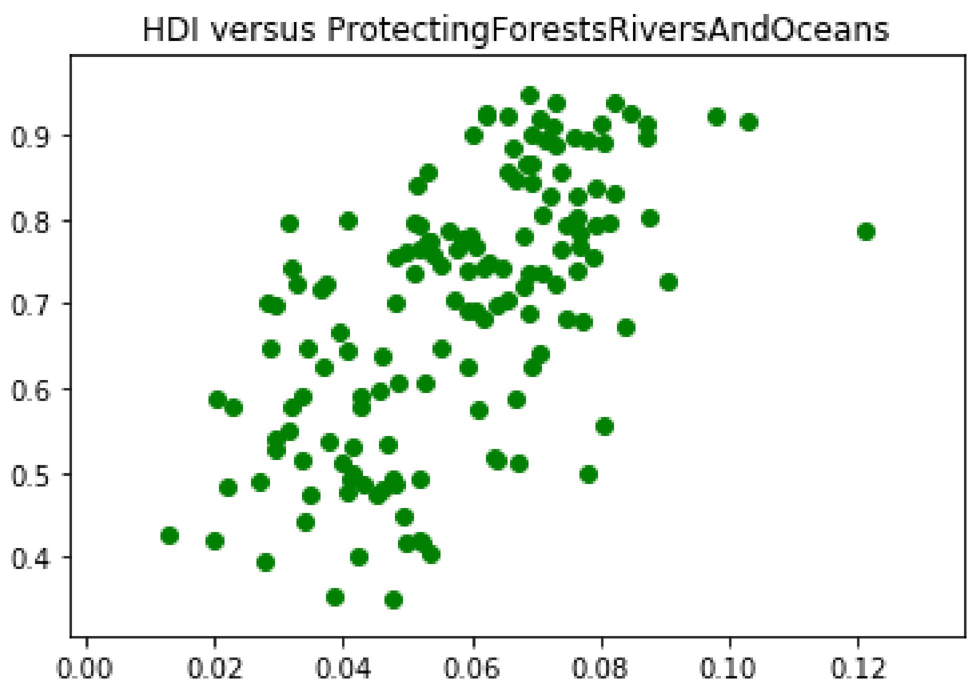
In [5]:
```
print("Predicting HDI from World Values Survey")
print()

# 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()
```

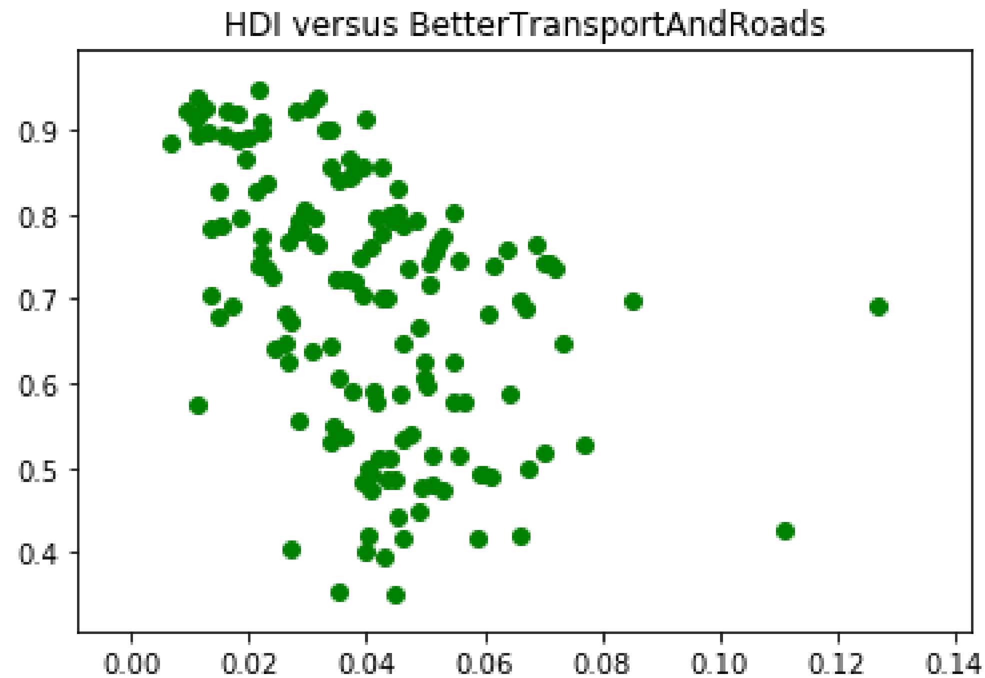
```
Predicting HDI from World Values Survey

Importing Training and Testing Data
Training Data Count: 148
Test Data Count: 38
```

1.1 (a) (b) (c)

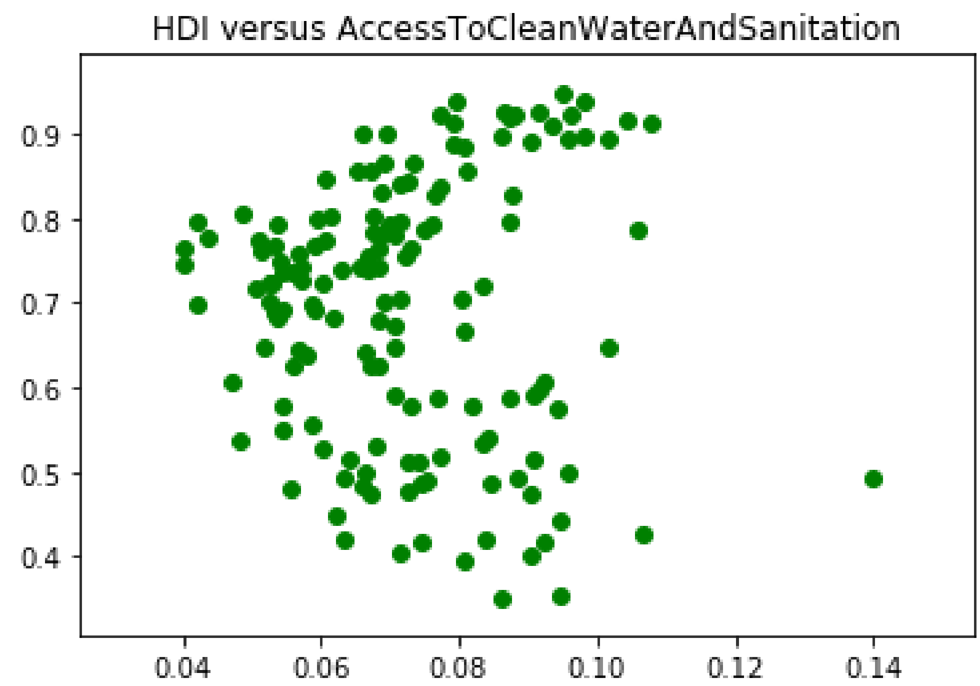
In [4]:
```
# Calculate Correlations #
calculate_correlations(values_train, hdi_train)
```

```
Action taken on climate change 0.473312891543
Better transport and roads -0.439633638622
Support for people who can't work -0.336213236721
Access to clean water and sanitation -0.018169084456
Better healthcare -0.422012359959
A good education -0.303978889772
A responsive government we can trust 0.329445314984
Phone and internet access -0.351604712158
Reliable energy at home -0.285423563836
Affordable and nutritious food 0.195193300786
Protecting forests rivers and oceans 0.613458756271
Protection against crime and violence 0.14331869918
Political freedoms 0.238099006821
Freedom from discrimination and persecution 0.432932375445
Equality between men and women 0.276496043498
Better job opportunities -0.39734452674
[0.4733, -0.43959999999999999, -0.3362, -0.018200000000000001, -0.42199999999999999, -0.30399999
```





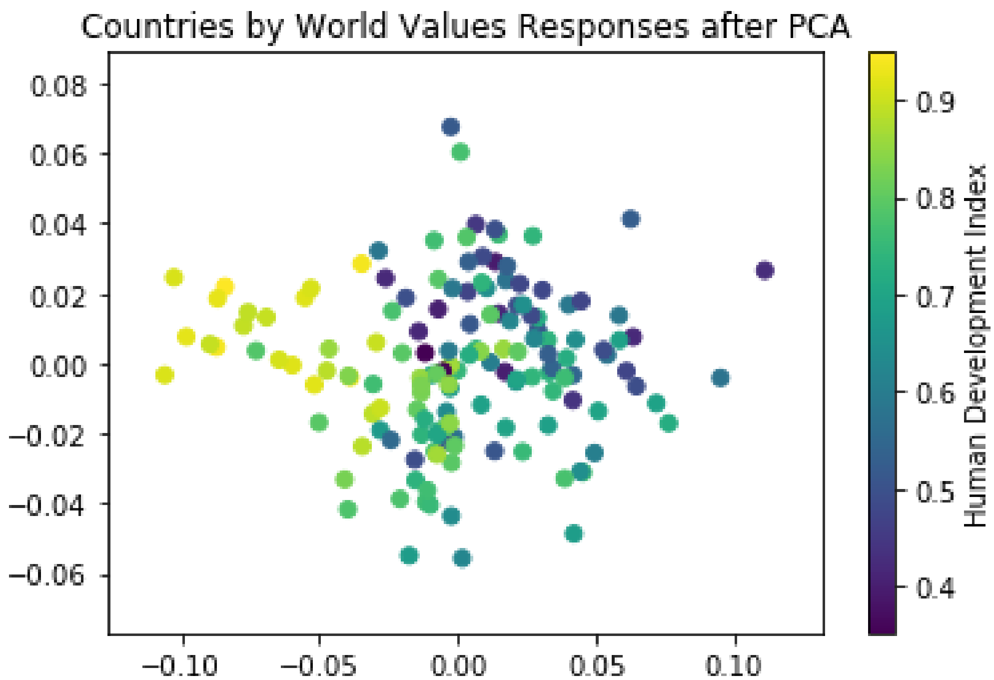
HDI versus BetterTransportAndRoads
0.9
0.8
0.7
0.6
0.5
0.4
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14



HDI versus AccessToCleanWaterAndSanitation
0.9
0.8
0.7
0.6
0.5
0.4
0.04
0.06
0.08
0.10
0.12
0.14

1.2 (d)

```
In [5]: # PCA #
        plot_pca(values_train, hdi_train, hdi_class_train)
```

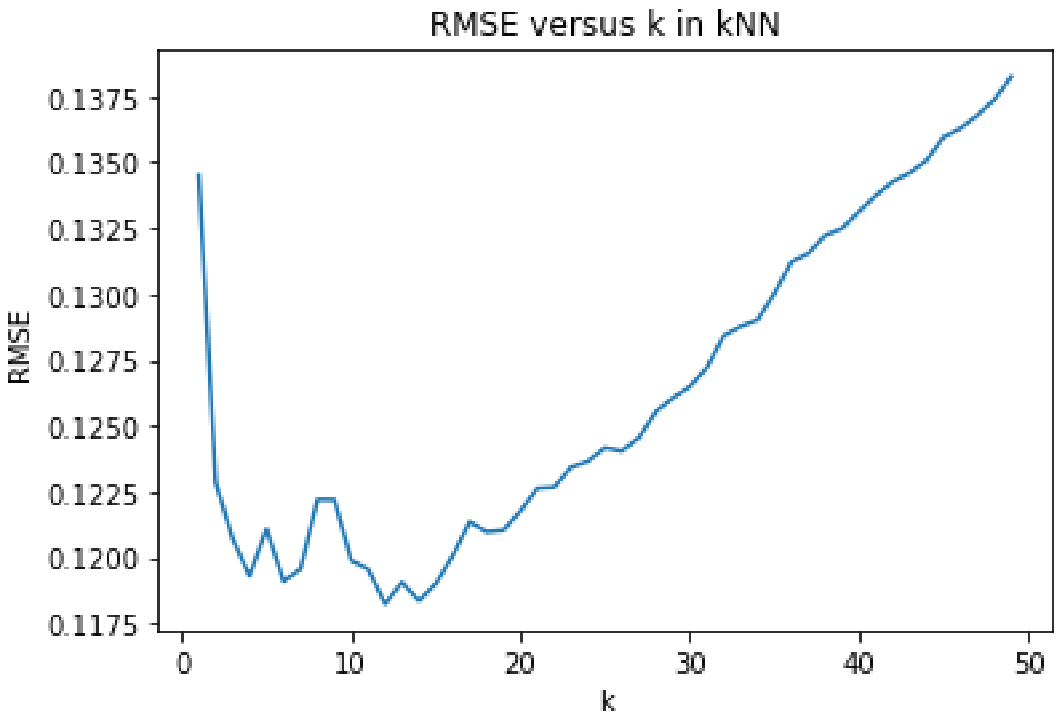


1.3 (e) (f) (g) (j)

```
In [8]: # Regression Grid Searches #
        regression_grid_searches(training_features=values_train,
                                 training_labels=hdi_shifted_train)
```

```
Ridge Regression
RMSE: 0.12303337350607801
Pipeline(steps=[('ridge', Ridge(alpha=0.02, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001))])
[[ 0.80823467 -0.74985758 -0.17800015 -1.28408103 -0.66293176 -0.82203172
   0.73733884 -0.92891581 -0.82049672  0.39614952  2.0708291  -0.06718981
   0.48310656  0.72671425  0.42921192 -0.13808023]]

Lasso Regression
RMSE: 0.12602242808947522
Pipeline(steps=[('lasso', Lasso(alpha=0.00020000000000000001, copy_X=True, fit_intercept=True,
   max_iter=1000, normalize=False, positive=False, precompute=False,
   random_state=None, selection='cyclic', tol=0.0001, warm_start=False))])
[ 0.1590192  -0.72844929 -0.          -0.85945074 -0.66274144 -0.02556703
```

```
  0.33904781 -0.29897158 -0.          0.          3.48536375  0.          0.
  0.87057995  0.32897045 -0.        ]

k Nearest Neighbors Regression
RMSE: 0.1182458946077689
Pipeline(steps=[('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=12, p=2,
          weights='uniform'))])
```



1.4 (h) (i)

In [113]:
```
neigh = KNeighborsRegressor(n_neighbors=7)
neigh.fit(values_train, hdi_shifted_train)
distance, index = neigh.kneighbors(values_train[45:46],8)
index = index[0,1:]
values_train.loc[index,:]
```

Out[113]:
```
     Action taken on climate change  Better transport and roads  \
90                         0.062152                    0.016203
61                         0.062594                    0.021893
```

37	0.080216	0.022225
108	0.067466	0.011244
69	0.064262	0.034098
132	0.071992	0.015786
110	0.079462	0.012823

	Support for people who can't work	Access to clean water and sanitation \
90	0.029873	0.087975
61	0.028984	0.093560
37	0.030295	0.086029
108	0.026837	0.101799
69	0.032787	0.081311
132	0.030854	0.095671
110	0.023346	0.098294

	Better healthcare	A good education \
90	0.087215	0.112025
61	0.073555	0.104335
37	0.068727	0.115161
108	0.062669	0.113643
69	0.070164	0.106230
132	0.065654	0.119349
110	0.064133	0.116129

	A responsive government we can trust	Phone and internet access \
90	0.094937	0.022152
61	0.089939	0.021651
37	0.083020	0.025303
108	0.087106	0.024438
69	0.095082	0.027541
132	0.080244	0.022483
110	0.079085	0.017561

	Reliable energy at home	Affordable and nutritious food \
90	0.019747	0.083924
61	0.030224	0.090759
37	0.027764	0.076934
108	0.026537	0.084108
69	0.037377	0.081967
132	0.024276	0.074025
110	0.023071	0.085265

	Protecting forests rivers and oceans \
90	0.062152
61	0.072450
37	0.075976
108	0.071364
69	0.065574

```
132                                0.078211
110                                0.087153

     Protection against crime and violence  Political freedoms  \
90                                0.072911            0.045316
61                                0.079449            0.044499
37                                0.065445            0.056760
108                               0.072114            0.055172
69                                0.078689            0.034098
132                               0.065056            0.063860
110                               0.069866            0.053816

     Freedom from discrimination and persecution  \
90                                      0.075316
61                                      0.076983
37                                      0.069890
108                                     0.071664
69                                      0.067541
132                                     0.081081
110                                     0.075274

     Equality between men and women  Better job opportunities
90                         0.070759                  0.057342
61                         0.061839                  0.047286
37                         0.070574                  0.045681
108                        0.080060                  0.043778
69                         0.066230                  0.057049
132                        0.068166                  0.043291
110                        0.076733                  0.037989
```

In [114]:
```python
values_train_with_country = pd.read_csv('world-values-train2.csv')
values_train_with_country.loc[index,'Country']
```

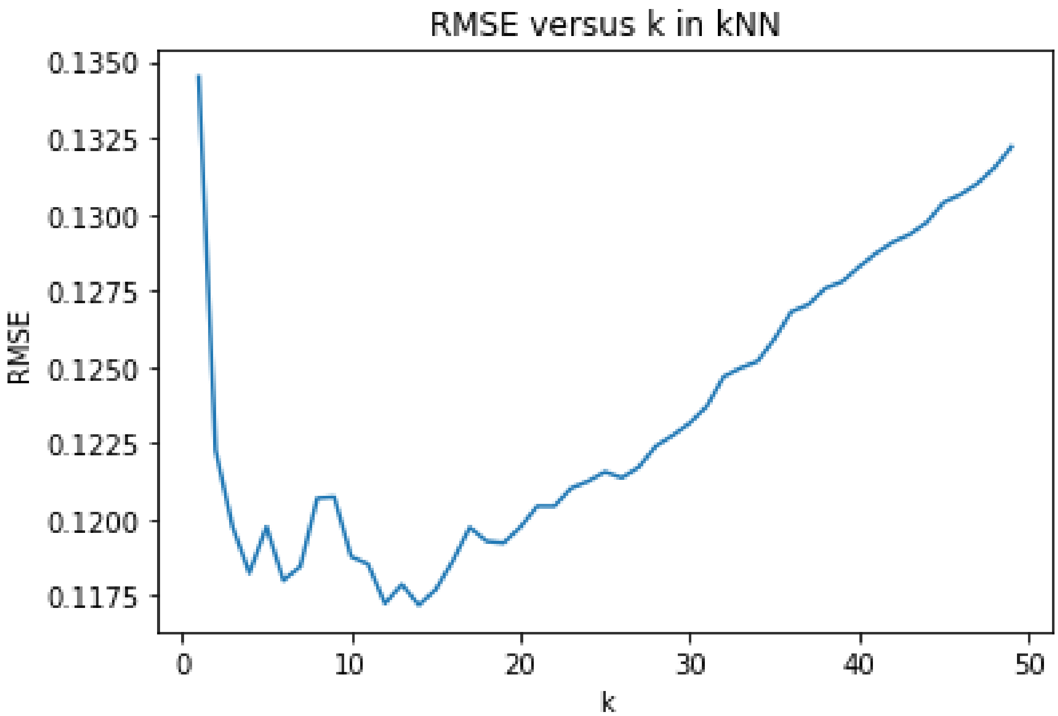
Out[114]:
```
90            Ireland
61     United Kingdom
37            Belgium
108           Finland
69              Malta
132           Austria
110            France
Name: Country, dtype: object
```

1.5 (l)

In [41]:
```python
regression_knn_parameters = {
    'knn__n_neighbors': np.arange(1, 50),

    # Apply uniform weighting vs k for k Nearest Neighbors Regression
```

```python
            #'knn__weights': ['uniform']

            # Apply distance weighting vs k for k Nearest Neighbors Regression
            'knn__weights': ['distance']
        }

        # Regression Grid Searches #
        regression_grid_searches(training_features=values_train,
                                 training_labels=hdi_shifted_train)
```

```
Ridge Regression
RMSE: 0.12303337350607801
Pipeline(steps=[('ridge', Ridge(alpha=0.02, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001))])
[[ 0.80823467 -0.74985758 -0.17800015 -1.28408103 -0.66293176 -0.82203172
   0.73733884 -0.92891581 -0.82049672  0.39614952  2.0708291  -0.06718981
   0.48310656  0.72671425  0.42921192 -0.13808023]]

Lasso Regression
RMSE: 0.12602242808947522
Pipeline(steps=[('lasso', Lasso(alpha=0.00020000000000000001, copy_X=True, fit_intercept=True,
   max_iter=1000, normalize=False, positive=False, precompute=False,
   random_state=None, selection='cyclic', tol=0.0001, warm_start=False))])
[ 0.1590192  -0.72844929 -0.         -0.85945074 -0.66274144 -0.02556703
  0.33904781 -0.29897158 -0.          0.          3.48536375  0.          0.
  0.87057995  0.32897045 -0.        ]

k Nearest Neighbors Regression
RMSE: 0.1171925270311745
Pipeline(steps=[('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=14, p=2,
          weights='distance'))])
```



1.6 (m)

```
In [115]: scaler = StandardScaler().fit(values_train)
          neigh = KNeighborsRegressor(n_neighbors=7)
          neigh.fit(scaler.transform(values_train), hdi_shifted_train)
          distance, index = neigh.kneighbors(values_train[45:46],8)
          index = index[0,1:]
          values_train_with_country.loc[index,'Country']
```

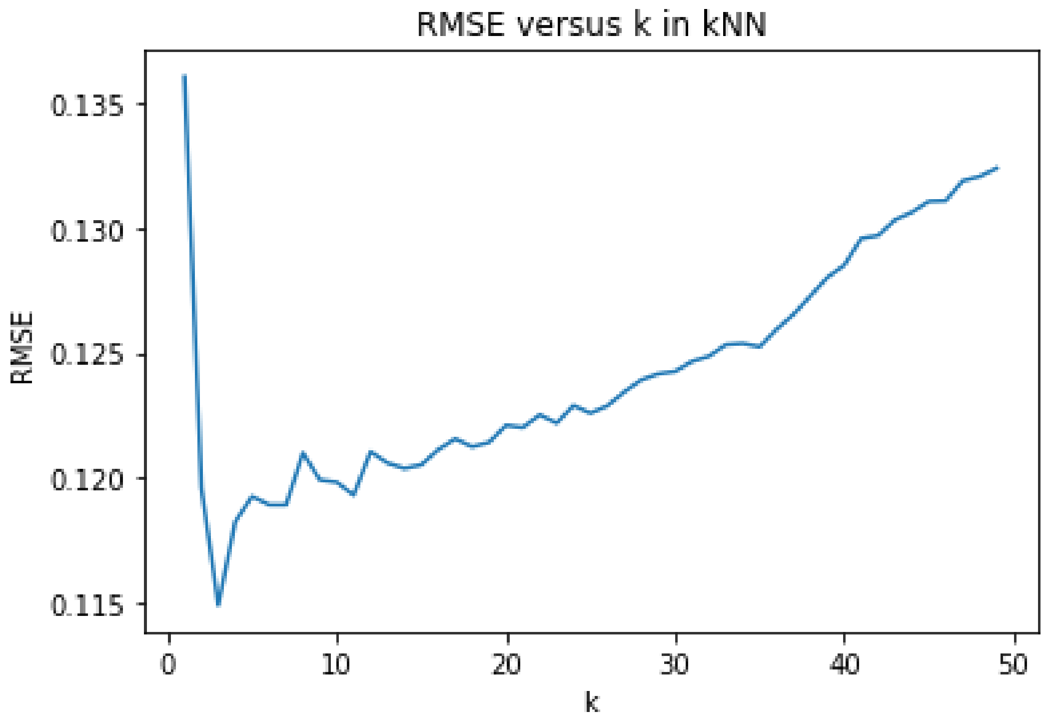
```
Out[115]: 107                 Qatar
          22                 Belize
          26             Mauritania
          94               Slovakia
          120             Lithuania
          72      Brunei Darussalam
          139                Kuwait
          Name: Country, dtype: object
```

1.7 (n)

```
In [45]: k_nearest_neighbors_regression_pipeline = Pipeline(
                 [
```

```
                    # Apply scaling to k Nearest Neighbors Regression
                    ('scale', StandardScaler()),
                    ('knn', KNeighborsRegressor())
                ]
            )
        # Regression Grid Searches #
        regression_grid_searches(training_features=values_train,
                                 training_labels=hdi_shifted_train)
```

```
Ridge Regression
RMSE: 0.12303337350607801
Pipeline(steps=[('ridge', Ridge(alpha=0.02, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001))])
[[ 0.80823467 -0.74985758 -0.17800015 -1.28408103 -0.66293176 -0.82203172
   0.73733884 -0.92891581 -0.82049672  0.39614952  2.0708291  -0.06718981
   0.48310656  0.72671425  0.42921192 -0.13808023]]

Lasso Regression
RMSE: 0.12602242808947522
Pipeline(steps=[('lasso', Lasso(alpha=0.00020000000000000001, copy_X=True, fit_intercept=True,
   max_iter=1000, normalize=False, positive=False, precompute=False,
   random_state=None, selection='cyclic', tol=0.0001, warm_start=False))])
[ 0.1590192  -0.72844929 -0.         -0.85945074 -0.66274144 -0.02556703
  0.33904781 -0.29897158 -0.          0.          3.48536375  0.          0.
  0.87057995  0.32897045 -0.        ]

k Nearest Neighbors Regression
RMSE: 0.11488547357936414
Pipeline(steps=[('scale', StandardScaler(copy=True, with_mean=True, with_std=True)), ('knn', KNe
          metric_params=None, n_jobs=1, n_neighbors=3, p=2,
          weights='distance'))])
```

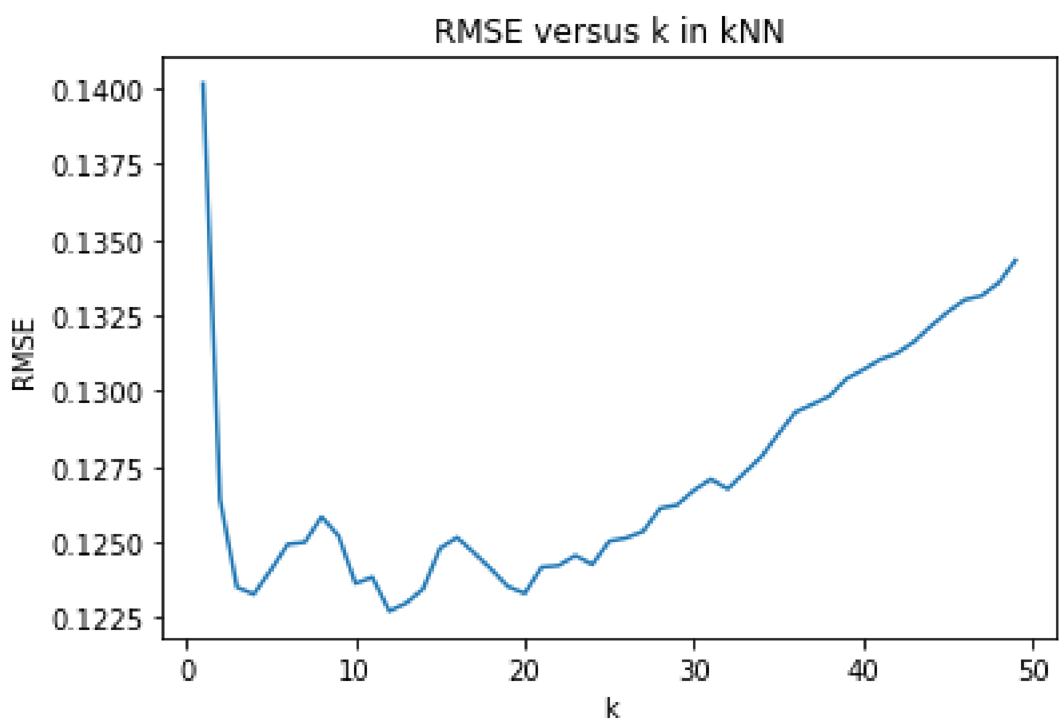


1.8 (o)

```
In [83]: values_train2 = values_train
         values_train2 = values_train2.apply(lambda x: (x - np.min(x)) / (np.max(x) - np.min(x))
         k_nearest_neighbors_regression_pipeline = Pipeline(
                 [
                     # Apply scaling to k Nearest Neighbors Regression
                     ('knn', KNeighborsRegressor())
                 ]
             )
         # Regression Grid Searches #
         regression_grid_searches(training_features=values_train2,
                                  training_labels=hdi_shifted_train)
```

```
Ridge Regression
RMSE: 0.1248148517352611
Pipeline(steps=[('ridge', Ridge(alpha=0.98999999999999999, copy_X=True, fit_intercept=True,
   max_iter=None, normalize=False, random_state=None, solver='auto',
   tol=0.001))])
[[ 0.08157067 -0.06169456  0.00430821 -0.16741075 -0.07579661 -0.1067142
   0.07202668 -0.08438885 -0.09631576  0.05612502  0.24389642 -0.03175892
```

```
   0.04745389  0.06829189  0.02722769  0.00200069]]

Lasso Regression
RMSE: 0.12709100104661503
Pipeline(steps=[('lasso', Lasso(alpha=0.0016000000000000001, copy_X=True, fit_intercept=True,
   max_iter=1000, normalize=False, positive=False, precompute=False,
   random_state=None, selection='cyclic', tol=0.0001, warm_start=False))])
[ 0.04110702 -0.05171624 -0.         -0.11769648 -0.09141354 -0.03434073
  0.02794239 -0.05645667 -0.06874863  0.          0.33412596 -0.
  0.03886577  0.05499597  0.         -0.        ]

k Nearest Neighbors Regression
RMSE: 0.12270887251692039
Pipeline(steps=[('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=12, p=2,
          weights='distance'))])
```



1.9 (p)

```
In [116]: neigh = KNeighborsRegressor(n_neighbors=14)
          neigh.fit(values_train, hdi_shifted_train)
```

```
         predict = neigh.predict(values_test)
```

```
In [117]: predict
```

```
Out[117]: array([[-0.17012934],
                  [ 0.00401351],
                  [-0.04470077],
                  [-0.02684363],
                  [ 0.12758494],
                  [-0.08320077],
                  [-0.06091506],
                  [-0.07791506],
                  [ 0.09901351],
                  [ 0.10644208],
                  [-0.00884363],
                  [ 0.04365637],
                  [ 0.04365637],
                  [ 0.00458494],
                  [-0.07991506],
                  [ 0.02379923],
                  [ 0.04694208],
                  [-0.07498649],
                  [-0.03055792],
                  [-0.02534363],
                  [-0.09470077],
                  [ 0.06501351],
                  [-0.13091506],
                  [ 0.13501351],
                  [ 0.03579923],
                  [ 0.02651351],
                  [-0.11755792],
                  [ 0.21865637],
                  [-0.05898649],
                  [ 0.02251351],
                  [ 0.02429923],
                  [ 0.16829923],
                  [ 0.0227278 ],
                  [ 0.21429923],
                  [ 0.22087066],
                  [ 0.08394208],
                  [-0.16748649],
                  [ 0.09465637]])
```

```
In [130]: pred = hdi_scaler.inverse_transform(predict)
          print(pred[:,0])
```

```
[ 0.52235714  0.6965      0.64778571  0.66564286  0.82007143  0.60928571
  0.63157143  0.61457143  0.7915      0.79892857  0.68364286  0.73614286
  0.73614286  0.69707143  0.61257143  0.71628571  0.73942857  0.6175
```

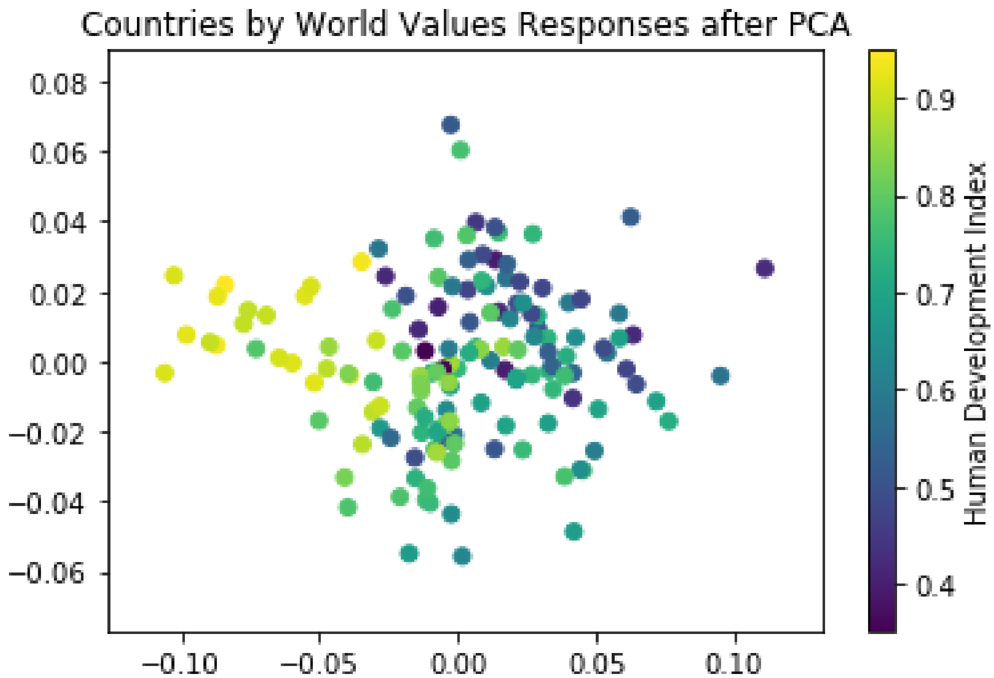
```
0.66192857  0.66714286  0.59778571  0.7575      0.56157143  0.8275
0.72828571  0.719       0.57492857  0.91114286  0.6335      0.715
0.71678571  0.86078571  0.71521429  0.90678571  0.91335714  0.77642857
0.525       0.78714286]
```

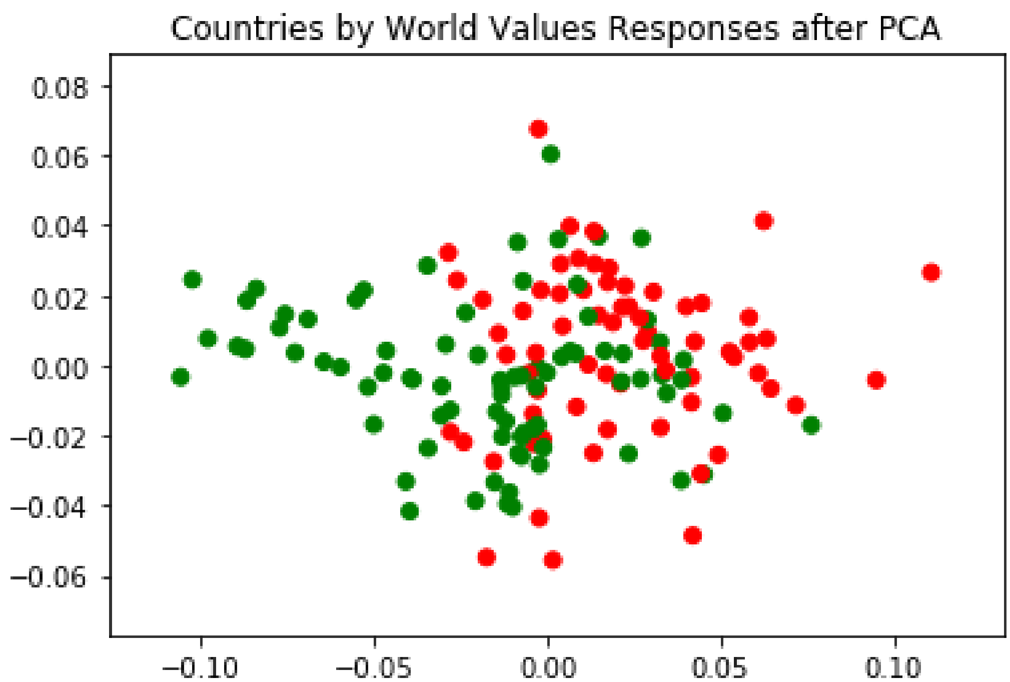
```
In [125]: with open('submission.txt','w') as f:
              for i in range(len(pred)):
                  f.write(str(np.round(pred[i,0],4)))
                  f.write('\n')
```

1.10 (r)

```
In [93]: def plot_pca(training_features,
                      training_labels,
                      training_classes):
             """
             Input:
                 training_features: world_values responses on the training set
                 training_labels: HDI (human development index) on the training set
                 training_classes: HDI class, determined by hdi_classification(), on the trainin

             Output:
                 Displays plot of first two PCA dimensions vs HDI
                 Displays plot of first two PCA dimensions vs HDI, colored by class
             """
             # Run PCA on training_features
             pca = PCA()
             transformed_features = pca.fit_transform(training_features)

             # Plot countries by first two PCA dimensions
             plt.scatter(transformed_features[:, 0],     # Select first column
                         transformed_features[:, 1],     # Select second column
                         c=training_labels)
             plt.colorbar(label='Human Development Index')
             plt.title('Countries by World Values Responses after PCA')
             plt.show()

             # Plot countries by first two PCA dimensions, color by class
             training_colors = training_classes.apply(lambda x: 'green' if x else 'red')
             plt.scatter(transformed_features[:, 0],     # Select first column
                         transformed_features[:, 1],     # Select second column
                         c=training_colors)
             plt.title('Countries by World Values Responses after PCA')
             plt.show()
```

```
In [94]: plot_pca(values_train, hdi_train, hdi_class_train)
```


Countries by World Values Responses after PCA
Human Development Index


Countries by World Values Responses after PCA

1.11 (t)

```
In [95]: # Classification Grid Searches #
         classification_grid_searches(training_features=values_train,
                                      training_classes=hdi_class_train)

SVM Classification
Accuracy: 0.75
Pipeline(steps=[('svm', SVC(C=48.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

k Nearest Neighbors Classification
Accuracy: 0.763513513514
Pipeline(steps=[('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])
```

1.12 (u)

```
In [97]: svm_classification_pipeline = Pipeline(
                 [
                     # Apply PCA to SVM Classification
                     ('pca', PCA()),
                     # Apply scaling to SVM Classification
                     ('scale', StandardScaler()),
                     ('svm', SVC())
                 ]
             )

         classification_svm_parameters = {
             # Use linear kernel for SVM Classification
             'svm__kernel': ['linear'],

             # Use rbf kernel for SVM Classification
             # 'svm__kernel': ['rbf'],

             # Original hyperparameters
             'svm__C': np.arange(1.0, 100.0, 1.0),

             # Original hyperparameters scaled by 1/100
             # 'svm__C': np.arange(0.01, 1.0, 0.01),

             # Hyperparameter search over all possible dimensions for PCA reduction
             'pca__n_components': np.arange(1, 17),
```

```
        #'svm_gamma': np.arange(0.001, 0.1, 0.001)
    }
```

```
In [98]: classification_grid_searches(training_features=values_train,
                                      training_classes=hdi_class_train)
```

```
SVM Classification
Accuracy: 0.77027027027
Pipeline(steps=[('pca', PCA(copy=True, iterated_power='auto', n_components=8, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)), ('scale', StandardScaler(copy=True, with_mean=True
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

k Nearest Neighbors Classification
Accuracy: 0.763513513514
Pipeline(steps=[('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])
```

1.13 (v)

```
In [99]: svm_classification_pipeline = Pipeline(
                 [
                     # Apply PCA to SVM Classification
                     #('pca', PCA()),
                     # Apply scaling to SVM Classification
                     #('scale', StandardScaler()),
                     ('svm', SVC())
                 ]
             )

         classification_svm_parameters = {
             # Use linear kernel for SVM Classification
             #'svm__kernel': ['linear'],

             # Use rbf kernel for SVM Classification
             'svm__kernel': ['rbf'],

             # Original hyperparameters
             'svm__C': np.arange(1.0, 100.0, 1.0),

             # Original hyperparameters scaled by 1/100
             # 'svm__C': np.arange(0.01, 1.0, 0.01),

             # Hyperparameter search over all possible dimensions for PCA reduction
```

```
            #'pca__n_components': np.arange(1, 17),

            #'svm_gamma': np.arange(0.001, 0.1, 0.001)
        }
```

```
In [100]: classification_grid_searches(training_features=values_train,
                                        training_classes=hdi_class_train)
```

```
SVM Classification
Accuracy: 0.689189189189
Pipeline(steps=[('svm', SVC(C=98.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

k Nearest Neighbors Classification
Accuracy: 0.763513513514
Pipeline(steps=[('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])
```

1.14 (w)

```
In [101]: classification_grid_searches(training_features=values_train,
                                        training_classes=hdi_class_train)
```

```
SVM Classification
Accuracy: 0.689189189189
Pipeline(steps=[('svm', SVC(C=98.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

k Nearest Neighbors Classification
Accuracy: 0.763513513514
Pipeline(steps=[('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])
```

```
In [102]: k_nearest_neighbors_classification_pipeline = Pipeline(
                  [
                      # Apply scaling to k Nearest Neighbors Classification
                      ('scale', StandardScaler()),
                      ('knn', KNeighborsClassifier())
                  ]
```

```
            )
        classification_knn_parameters = {
            'knn__n_neighbors': np.arange(1, 50),

            # Apply distance weighting vs k for k Nearest Neighbors Classification
            'knn__weights': ['distance']
        }
```

In [103]:
```
classification_grid_searches(training_features=values_train,
                              training_classes=hdi_class_train)
```

```
SVM Classification
Accuracy: 0.689189189189
Pipeline(steps=[('svm', SVC(C=98.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

k Nearest Neighbors Classification
Accuracy: 0.77027027027
Pipeline(steps=[('scale', StandardScaler(copy=True, with_mean=True, with_std=True)), ('knn', KNe
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])
```

1.15 (x)

In [7]:
```
x = np.array([33,30,15,57,51,95,44,55,36,57,22,54,31,47,36,52])/110
```

In [9]:
```
neigh = KNeighborsRegressor(n_neighbors=14)
neigh.fit(values_train, hdi_shifted_train)
predict = neigh.predict(x.reshape(1,-1))
```

In [11]:
```
pred = hdi_scaler.inverse_transform(predict)
pred
```

Out[11]:
```
array([[ 0.54592857]])
```