HW11 CS 189

1.
Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

I worked on this homework with Ehimare Okoyomon, Prashanth Ganeth, and Daniel Mockaitis. We worked by getting together throughout the week and communicating on facebook.

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}

Nicholas Lorio, 26089160

Question 5 Own Question.

When does nearest neighbors fail?

Nearest neighbors strongly depends on the distance between points. As the number of dimensions increases, the distances we use will be less representative (this is the curse of dimensionality). As the dimension increase, the training examples needed to cover the portion of the space becomes larger exponentially. Unless you modify the data, KNN is very bad for high dimensional data.

2.a. $\min \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} \phi_i \varepsilon_i$ s.t. $y_i(w^T x_i - b) - (1-\varepsilon_i) \geq 0 \ \forall i$

$\varepsilon \geq 0 \ \forall i$

• $L_{hinge}(y, w^T x + b) = \max(0, 1 - y_i(w^T x_i - b))$

regularized as: $\min_{w,b} \frac{1}{n}\sum_{i=1}^{n} \max(1 - y_i(w^T x_i - b), 0) + \lambda\|w\|^2$

• $(y_i(w^T x_i - b) - 1)(-1) \leq \varepsilon_i$, $\varepsilon_i \geq 1 - y_i(w^T x_i - b)$

$\varepsilon_i \geq 0$

$\forall w$ $\quad \varepsilon_i \geq \max(1 - y_i(w^T x_i - b), 0)$

$$\left.\begin{array}{l}\min_{w,b,\varepsilon_i} \frac{1}{2}\|w\|_2^2 + C\sum \phi_i \varepsilon_i \\ \text{s.t. } \varepsilon_i \geq \max(1 - y_i(w^T x_i - b), 0)\end{array}\right\} = \min_{w,b} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} \phi_i \max(1 - y_i(w^T x_i - b), 0)$$

as required

• $Dual(\alpha, \beta) = \min L(\vec{w}, b, \vec{\alpha}, \vec{\beta}, \vec{\varepsilon})$

2.b. $\mathcal{L}(\vec{w}, b, \vec{\alpha}, \vec{\beta}, \vec{\varepsilon}) = \frac{1}{2}\|w\|_2^2 + C\sum \phi_i \varepsilon_i - \sum \alpha_i(y_i(w^T x_i - b) - (1 - \varepsilon_i))$

$- \sum \beta_i \varepsilon_i$

$= \frac{1}{2}\|w\|^2 - \sum \alpha_i y_i (\underbrace{w^T x_i}_{x_i^T w} - b) + \sum \alpha_i + \sum (C\phi_i - \alpha_i - \beta_i)\varepsilon_i$

def cond. $\min \mathcal{L}(\ )$

$\delta\mathcal{L}/\delta w = 0$, $w^T - \sum \alpha_i y_i x_i^T = 0$, $w^{T*} = \sum \alpha_i y_i x_i^T$

$\delta\mathcal{L}/\delta b = 0$, $\sum \alpha_i^* y_i^* = 0$

$\delta\mathcal{L}/\delta \varepsilon_i = 0$, $C\phi_i - \alpha_i^* - \beta_i^* = 0$

$\varepsilon_i \geq 0$ primal feasibility, $\alpha_i \geq 0$, $\beta_i \geq 0$ dual feasibility. All Lagrange multipliers are non zero, hence complementary slackness satisfied.

$\mathcal{L}(\ ) = \frac{1}{2}\|\vec{w}^*\|^2 - \sum \alpha_i y_i x_i^T w^* + \sum \alpha^* y_i^* b^* + \sum \alpha_i^* + \sum (C\phi_i - \alpha_i^* - \beta_i^*)\varepsilon_i$

$= \frac{1}{2}\|w^*\|^2 - w^{T*} w^* + \sum \alpha_i^*$

$$= -\frac{1}{2}\|w^*\|^2 + \sum_{i=1}^{\hat{}} \alpha_i^* = -\frac{1}{2}\left(\sum_{i=1}^{\hat{}}\sum_{j=1}^{\hat{}} \alpha_i^* y_i x_i^T x_j y_j \alpha_j^*\right) + \sum \alpha_i$$

let $G = \left[y_i \langle x_i, x_j \rangle y_j \right] = diag(y)\, X X^T\, diag(y)$

$$= -\frac{1}{2}\alpha^{*T} G \alpha^* + \sum_{i=1}^{\hat{}} \alpha_i = -\frac{1}{2}\alpha^{*T} G \alpha^* + \alpha^T \cdot \vec{1}$$

$$= \min_{w,b,\xi} \mathcal{L}(\vec{w}, \vec{\xi}, b, \vec{\alpha}, \vec{\beta}) = \mathcal{L}(\alpha^*, x^*, y^*) = \text{Dual Function}$$

$$\left[\max_{\alpha} \; \alpha^T \vec{1} - \frac{1}{2}\alpha^T G \alpha \quad s.t \quad \sum \alpha_i y_i = 0, \;\; \alpha_i \geq 0, \;\; \alpha_i \leq C\phi_i \right]$$

$C\phi_i = \alpha_i - \beta_i = 0$, $\beta_i \geq 0, \alpha_i \geq 0$

$\beta_i = C\phi_i - \alpha_i \Rightarrow \alpha_i \leq C\phi_i$

c. $G = \left[y_i k(x_i, x_j) y_j \right]$ w/ $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$ as the kernel function

- allows opt efficient when ∃ lots of features
- allows comp one large mult per drug example instead of one weight per feature.

$w^* x_{test} - b^* = \sum_{i=1}^{\hat{}} \alpha_i^* y_i x_i^T x_{test} - b^* \Rightarrow \sum_{i=1}^{\hat{}} \alpha_i^* y_i k(x_i, x_{test}) - b^*$

-> the solution function above can be generalized as shown.

-> we kernelize it similarly to how we kernelize SVM in general.

2.C. $G = \left[y_i \; k(x_i, x_j) \; y_j \right]$ w/ $k(x_i, x_j)$ as the kernel function

- Solution Function: $h(x) = w^* x_{test} - b^*$

$$= \sum_{i=1}^{\hat{}} \alpha_i^* y_i x_i^T x_{test} - b^*$$

- Generalize $$= \sum_{i=1}^{\hat{}} \alpha_i^* y_i \, k(x_i, x_{test}) - b^*$$

-> Same as how we would generate regular SVM.

- Role of ϕ_i : $0 \alpha_i$, Lagrange multiplier for the primal is bounded above by the size of $\phi_i \cdot C$.

-> if ϕ_i is small than the primal places less weight on point i being correctly classified to correct side of the decision boundary. (the opposite can be said for ϕ_i large = More weight will be associated to point i.)
- the weight determines how much of an effect their classification will have for the minimization

-> if ϕ_i small => Dual is constrained the Maximizing α of the function to get correct classification on smaller interval $0 \leq \alpha_i \leq C \cdot \phi_i$ (same idea as in primal 3 less precedent to correctly maximize the point i)
-> if ϕ_i large => can have more optimized classification/ Larger/ More Maximized.

4.a. $E_{S\sim D^n}[L_D(w(S)) - L_S(w(S))]$

$= E_{S\sim D^n}\left[E_{z\sim D}[l(w(S), z)] - \frac{1}{n}\sum_{i=1}^{n} l(w(S), z_i) \right]$

$= E_{S\sim D^n}\left[E_{z\sim D}[l(w(S), z)] - E_{i\sim U(n)}[l(w(S), z_i)] \right]$

by canceling w/ (we switch last part w/ one of the terms) $E_{z\sim D}[l(w(S), z')] = E_{i\sim U(n), z\sim D}[l(w(S^{(i)}(z'), z_i)]$

$E_{S\sim D^n}\left[E_{i\sim U(n), z\sim D}[l(w(S^{(i)}(z'), z_i)] - E_{i\sim U(n), z\sim D}[l(w(S), z_i)] \right]$

$= E_{S\sim D^n, i\sim U(n), z\sim D}[l(w(S^{(i)}(z'), z_i) - l(w(S), z_i)]$

as required, (14) is true

then if $E_{S\sim D^n, z\sim D, i\sim U(n)}[l(w(S^{(i)}(z')), z_i) - l(w(S), z_i)] \leq \varepsilon_n$

$\Rightarrow E_{S\sim D^n}[L_D(w(S)) - L_S(w(S))] \leq \varepsilon_n$

$\Rightarrow E_{S\sim D^n}[L_D(w(S))] - E_{S\sim D^n}[L_S(w(S))] \leq \varepsilon_n$

$\Rightarrow E_{S\sim D^n}[L_D(w(S))] \leq E_{S\sim D^n}[L_S(w(S))] + \varepsilon_n$

as required (13) is true

4.b. $l(\mu(S^i(x')), x_i) - l(\mu(S), x_i) = \frac{1}{2}(\mu(S^i(x')) - x_i)^2 - \frac{1}{2}(\mu(S) - x_i)^2$

Let $A = \frac{1}{\lambda}\sum_{j=1}^{\lambda} x_j \quad B = x_i \quad C = x'$

$= \frac{1}{2}\left(\frac{1}{\lambda}\sum_{j=1}^{\lambda} x_j - x_i + x'\right) - x_i)^2 - \frac{1}{2}\left(\frac{1}{\lambda}\sum_{j=1}^{\lambda} x_j - x_i\right)^2$

$= \frac{1}{2}\left(A - \frac{1}{\lambda}B + \frac{1}{\lambda}C - B\right)^2 - \frac{1}{2}(A-B)^2$

$= \frac{1}{2}\left(A - \left(\frac{1}{\lambda}+1\right)B + \frac{1}{\lambda}C\right)^2 - \frac{1}{2}(A-B)^2$

$= \frac{1}{2}\left(A^2 - 2A\left(\frac{1}{\lambda}+1\right)B + 2A\frac{1}{\lambda}C - 2BC\left(\frac{1}{\lambda}+1\right)\left(\frac{1}{\lambda}\right) + \left(\frac{1}{\lambda}+1\right)^2 B^2 + \frac{1}{\lambda^2}C^2\right)$

$- 1/2\,(A-B)^2$

$= \frac{1-\lambda}{\lambda}AB + \frac{1}{\lambda}AC - \frac{\lambda+1}{\lambda^2}BC + \frac{(\lambda+1)^2}{\lambda^2}B^2 + \frac{1}{\lambda^2}C^2 + 2AB - B^2$

$AB = \frac{1}{\lambda}\sum x_j \cdot x_i \in \frac{1}{\lambda}\lambda\Omega^2 = \Omega^2 \qquad B^2 = x_i^2 \in \Omega^2$

$AC = \frac{1}{\lambda}\sum x_j \cdot x' \in \frac{1}{\lambda}\lambda\Omega^2 = \Omega^2 \qquad C^2 = x'^2 \in \Omega^2$

$BC = \frac{1}{\lambda}\sum x_j \cdot x' \in \frac{1}{\lambda}\lambda\Omega^2 = \Omega^2$

$\in \frac{1-\lambda}{\lambda}\Omega^2 + \frac{1}{\lambda}\Omega^2 - \frac{\lambda+1}{\lambda^2}\Omega^2 - \frac{(\lambda+1)^2}{\lambda^2}\Omega^2 + \frac{1}{\lambda^2}\Omega^2 + 2\Omega^2 - \Omega^2$

$= \left(\frac{1-\lambda}{\lambda} + \frac{1}{\lambda} - \frac{(\lambda+1)}{\lambda^2} - \frac{(\lambda+1)^2}{\lambda^2} + \frac{1}{\lambda^2} + 1\right)\Omega^2$

$= \left(\frac{1}{\lambda}(2) - \frac{1}{\lambda^2}(\lambda^2 + 3\lambda + 1) + 1\right)\Omega^2$

$= \left(\frac{2}{\lambda} - 1 - \frac{3}{\lambda} - \frac{1}{\lambda^2} + 1\right)\Omega^2 = \left(-\frac{1}{\lambda} - \frac{1}{\lambda^2}\right)\Omega^2 = \left(\frac{-\lambda^2-1}{\lambda^2}\right)\Omega^2$

4.b. cont. $\left(\frac{-\lambda^2-1}{\lambda}\right) R^2$ w/ $C = \frac{-\lambda^2-1}{\lambda}$

By (14) $\underset{\substack{i \sim w(\lambda) \\ z \sim D}}{E_{S \sim D^n}} \left[l(\mu(S^{(i)}(x^i)), x_i) - l(\mu(S), x_i) \right] \leq \frac{C R^2}{\lambda}$

$\Rightarrow E_{S \sim D^n} \left[L_D(\mu(S)) - L_S(\mu(S)) \right] \leq \frac{C R^2}{\lambda}$

4.c. $w(S) = \arg\min_w L_S(w) + \lambda \|w\|^2 = \arg\min_w \frac{1}{n} \sum_{i=1}^{n} l(w, z_i) + \lambda \|w\|^2$

$= \arg\min_w \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} (w^T x_i - y_i)^2 + \lambda \|w\|^2$

$\| \underbrace{\nabla l(v, z_i) - \nabla l(w, z_i)}_{G} \|$

$\nabla l(v, z_i) = \nabla \frac{1}{2} (v^T x_i - y_i)^2 = (v^T x_i - y_i) x_i$

$\nabla l(w, z_i) = \nabla \frac{1}{2} (w^T x_i - y_i)^2 = (w^T x_i - y_i) x_i$

$G = (v^T x_i - y_i) x_i - (w^T x_i - y_i) x_i = (v^T x_i - w^T x_i) x_i$

$\| \text{"} \| = \sqrt{[(v^T x_i - w^T x_i) x_i]^T (v^T x_i - w^T x_i) x_i}$

$= \sqrt{[x_i^T (x_i^T v - x_i^T w)][(v^T x_i - w^T x_i) x_i]}$

$= \sqrt{(x_i^T)^2 (v-w)(v^T - w^T) x_i^2} = x_i^T \sqrt{(v-w)(v-w)^T} x_i = \|x_i\|_2^2 \|v-w\|$

$= \|x_i\|^2 \|v-w\| \leq \max_i \|x_i\|^2 \|v-w\| = \beta \|v-w\|$ as reqd

4.D. as $\ell(w, z_i)$ is β-smooth by (16)

we have $E_{\substack{S\sim D^n \\ i\sim \mathrm{unif}(n) \\ z'\sim D}}\left[\ell(w(S^{(i)}(z')), z_i) - \ell(w(S), z_i)\right] \le \frac{c\beta}{\lambda n}$

by (14) $E_{S\sim D^n}\left[L_D(w(S)) - L_S(w(S))\right] \le \frac{c\beta}{\lambda n}$

$\Rightarrow E_{S\sim D^n}\left[L_D(w(S))\right] - E_{S\sim D^n}\left[L_S(w(S))\right] \le \frac{c\beta}{\lambda n}$

$\Rightarrow E_{S\sim D^n}\left[L_D(w(S))\right] \le E_{S\sim D^n}\left[L_S(w(S))\right] + \frac{c\beta}{\lambda n}$

4.C $\underline{\ell(w(S^i(z'), z_i) - \ell(w(S), z_i)} + \underline{\ell(w(S), z') - \ell(w(S^i(z')), z')}$

$= \frac{1}{n}\ell(w(S^{(i)}(z')), z_i) - \frac{1}{n}\ell(w(S), z_i) + \frac{1}{n}\ell(w(S), z') - \frac{1}{n}\ell(w(S^i(z')), z')$

$= \frac{1}{n}\ell(w(S^{(i)}(z')), z') + \frac{1}{n}\ell(w(S^{(i)}(z')), z_i) - \left[-\frac{1}{n}\ell(w(S), z') + \frac{1}{n}\ell(w(S), z_i)\right]$

Note: $F_S(w) - F_{S^i(z')}(w) = -\frac{1}{n}\ell(w, z') + \frac{1}{n}\ell(w, z_i)$

$= F_S(w(S^{(i)}(z'))) - F_{S^{(i)}(z')}(w(S^{(i)}(z'))) - \left[F_S(w(S)) - F_{S^i(z')}(w(S))\right]$

$= \underbrace{F_S(w(S^{(i)}(z'))) - F_S(w(S))}_{w(S) \text{ is minimizer of } F_S,} + \underbrace{F_{S^i(z')}(w(S)) - F_{S^{(i)}(z')}(w(S^{(i)}(z')))}_{w(S^{(i)}(z')) \text{ is minimizer of } F_{S^{(i)}(z')}}$

so $\ge \lambda\|w(S^{(i)}(z')) - w(S)\|_2^2$ | so $\ge \lambda\|w(S) - w(S^i(z'))\|_2^2$

4.e. cont.

$$\geq \lambda\|w(s^{(i)}(z')) - w(s)\|_2^2 + \lambda\|w(s) - w(s^{(i)}(z'))\|_2^2$$

$$= 2\lambda\|w(s^{(i)}(z')) - w(s)\|_2^2 \geq \lambda\|w(s^{(i)}(z')) - w(s)\|_2^2$$

so $\lambda\|w(s^{(i)}(z')) - w(s)\|_2^2 \leq$ [$\underline{\ell(w(s^{(i)}(z'), z_i) - \ell(w(s), z_i)}$ $\wedge$; $\underline{\ell(w(\tilde{s}), z') - \ell(w(s^{(i)}(z')), z')}$ $\wedge$]

Show eqn (18), using (17)

$$\rightarrow \lambda\|w(s^{(i)}(z')) - w(s)\|_2^2 \leq \underline{\rho\|w(s^{(i)}(z')) - w(s)\|} + \underline{\rho\|\hat{w}(s^{(i)}(z')) - w(s)\|}$$

$$= 2\rho\|w(s^{(i)}(z')) - w(s)\| \;/\wedge$$

so $\|w(s^{(i)}(z')) - w(s)\|_2^2 \leq \underline{2\rho\|w(s^{(i)}(z')) - w(s)\|}$ $\wedge$

$$\Rightarrow \|w(s^{(i)}(z')) - w(s)\| \leq 2\rho/\lambda \Rightarrow \rho\|w(s^{(i)}(z')) - w(s)\| \leq \frac{2\rho^2}{\lambda}$$

and by (17) $\ell(w(s^{(i)}(z')), z_i) - \ell(w(s), z_i) \leq \rho\|w(s^{(i)}(z')) - w(s)\|$

$$\Rightarrow \ell(w(s^{(i)}(z')), z_i) - \ell(w(s), z_i) \leq \frac{2\rho^2}{\lambda} \quad \square$$

4. $f: \mathbb{R} \to \mathbb{R}$ let $f(x) = \max(0, x)$

ρ Lipschitz: $|f(x) - f(y)| \leq \rho |x - y|$

let $\rho = 1$ & exam the falling cases for x, y

case 1

$x > 0, y > 0$ $\quad |\max(0,x) - \max(0,y)| \leq 1 \cdot |x - y|$

$|x - y| \leq |x - y|$

case 2

$x \leq 0, y > 0$ $\quad |\max(0,x) - \max(0,y)| \leq |x - y|$

$|0 - y| \leq |x - y|, \quad y \leq |x - y|$

case 3

$x > 0, y \leq 0$ $\quad |\max(0,x) - \max(0,y)| \leq |x - y|$

$|x - 0| \leq |x - y|, \quad x \leq |x - y|$

case 4.

$x \leq 0, y \leq 0$ $\quad |\max(0,x) - \max(0,y)| \leq |x - y|$

$|0 - 0| = 0 \leq |x - y|$

- Show { $g_i(w) = 1 - y_i w^T x_i$ is ρ Lipschitz w/ $\rho = \|x_i\|$ }

$\{ |g_i(w_1) - g_i(w_2)| \leq \|x_i\| |w_1 - w_2| \}$

$|1 - y_i w_1^T x_i - 1 + y_i w_2^T x_i| = |-y_i (w_1^T - w_2^T) x_i|$

$|-y_i (w_1^T - w_2^T) x_i| = |y_i (w_1^T - w_2^T) x_i|$

as $y_i z \leq z \Rightarrow |y_i (w_1^T - w_2^T) x_i| \leq |(w_1^T - w_2^T) x_i|$

$= |w_1^T - w_2^T| \, ||x_i||$

$|w_1^T - w_2^T| \, ||x_i|| \leq ||x_i||_2 |w_1 - w_2|$ as required.

Want show: composition of an L-lipschitz function w/ a M-lipschitz

$|F(x) - F(y)| \leq L |x - y|$ $\quad$ $|g(x) - g(y)| \leq M |x - y|$

then $F \circ G \rightarrow$

$|F(G(x)) - F(G(y))| \leq M |F(G(x)) - F(G(y))| \leq M \cdot L |F(G(x)) - F(G(y))|$

thus $F(G(w))$ is $||x_i||$ lipschitz

this proves that $l(w, z_i)$ is ρ-lipschitz

w/ $\rho = \max_i ||x_i||$ as our composition is a max function.

e.g. Since $\ell(w, z_i)$ is ρ Lipschitz

$$\ell(w(s^{(i)}(z')), z_i) - \ell(w(s), z_i) \le \rho \|w(s^i(z')) - w(s)\|$$

$$E[\,''\,] \le E[\,''\,]$$

By (14), $E[L_D(w(s)) - L_S(w(s))] \le \rho \|w(s^{(i)}(z')) - w(s)\|$

$$\Rightarrow E_{S\sim\mathcal{D}^n}[L_D(w(s))] \le E_{S\sim\mathcal{D}^n}[L_S(w(s))] + \underbrace{\rho \|w(s^i(z')) - w(s)\|}_{\varepsilon_n}$$

In [1]:
```
%matplotlib inline
```

In [2]:
```
"""
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

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

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_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 svm_classification_pipeline
from world_values_pipelines import k_nearest_neighbors_classification_pipeline
from world_values_pipelines import tree_classification_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 classification_svm_parameters
from world_values_parameters import classification_knn_parameters
from world_values_parameters import classification_tree_parameters


def main():
    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()

    # Part b and c: Calculate Correlations #
    # calculate_correlations(values_train, hdi_train)
```

```
Predicting HDI from World Values Survey

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

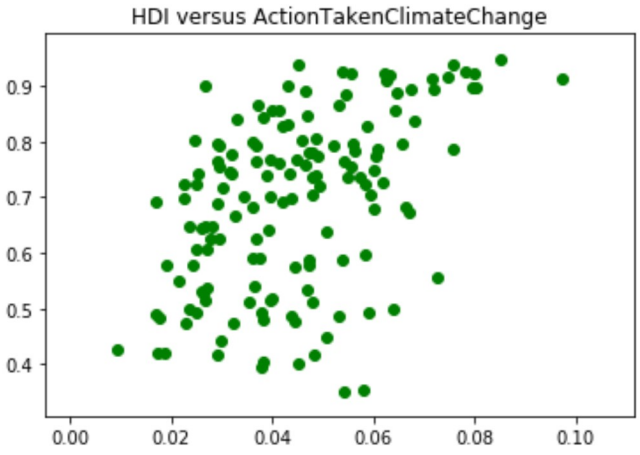
In [3]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Binarizer
from sklearn.decomposition import PCA
from sklearn.preprocessing import PolynomialFeatures
from sklearn.neural_network 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
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

3. Nearest Neighbors for Regression, from A to Z

b)

In [4]:
```
print("Predicting HDI from World Values Survey")
print()
# Import Data #
print("Importing Training and Testing Data")
print()
values_train, hdi_train, values_test = import_world_values_data()
# Part b and c: Calculate Correlations #
calculate_correlations(values_train, hdi_train)
```

```
Predicting HDI from World Values Survey

Importing Training and Testing Data

Action taken on climate change 0.4733128915429902
Better transport and roads -0.4396336386224581
Support for people who can't work -0.3362132367214908
Access to clean water and sanitation -0.018169084455954738
Better healthcare -0.4220123599593431
A good education -0.30397888977158366
A responsive government we can trust 0.3294453149841781
Phone and internet access -0.3516047121577415
Reliable energy at home -0.28542356383597445
Affordable and nutritious food 0.19519330078603306
Protecting forests rivers and oceans 0.6134587562712407
Protection against crime and violence 0.14331869917957568
Political freedoms 0.23809900682145016
Freedom from discrimination and persecution 0.4329323754445626
Equality between men and women 0.2764960434975283
Better job opportunities -0.39734452673964066
[0.4733, -0.4396, -0.3362, -0.0182, -0.422, -0.304, 0.3294, -0.3516, -0.2854, 0.
1952, 0.6135, 0.1433, 0.2381, 0.4329, 0.2765, -0.3973]
```

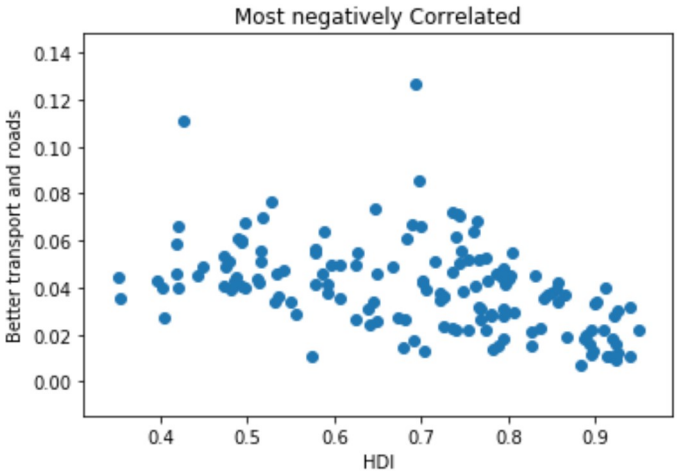


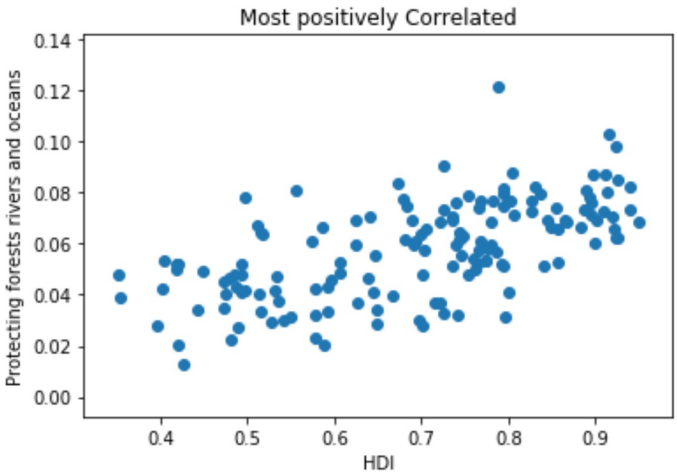
In [5]:
```
import numpy as np
import pandas as pd

correlations = [0.4733, -0.4396, -0.3362, -0.0182, -0.422, -0.304, 0.3294, -0.3516,
-0.2854, 0.1952, 0.6135, 0.1433, 0.2381, 0.4329, 0.2765, -0.3973]

print("Most negatively correlated with HDI: ")
print(np.min(correlations))
print("Better transport and roads -0.4396336386224581, index: 1")
print()

print("Most positively corrleated with HDI: ")
print(np.max(correlations))
print("Protecting forests rivers and oceans 0.6134587562712407, index: 10")
print()

print("Least correlated with HDI: ")
least = correlations[0]
for i in correlations:
    if np.abs(i - 0) < np.abs(least - 0):
        least = i
print(least)
print("Access to clean water and sanitation -0.018169084455954738, index: 3")
```

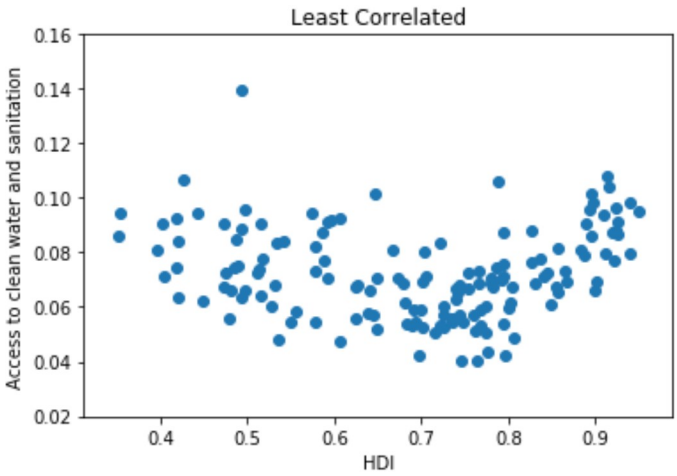
```
Most negatively correlated with HDI:
-0.4396
Better transport and roads -0.4396336386224581, index: 1

Most positively corrleated with HDI:
0.6135
Protecting forests rivers and oceans 0.6134587562712407, index: 10

Least correlated with HDI:
-0.0182
Access to clean water and sanitation -0.018169084455954738, index: 3
```

c)

In [6]:
```
hdi_col = hdi_train['2015']

negative_values = values_train['Better transport and roads']

plt.scatter(hdi_col, negative_values)
plt.title("Most negatively Correlated")
plt.xlabel("HDI")
plt.ylabel("Better transport and roads")
plt.show()

positive_values = values_train['Protecting forests rivers and oceans']

plt.scatter(hdi_col, positive_values)
plt.title("Most positively Correlated")
plt.xlabel("HDI")
plt.ylabel("Protecting forests rivers and oceans")
plt.show()

least_values = values_train['Access to clean water and sanitation']

plt.scatter(hdi_col, least_values)
plt.title("Least Correlated")
plt.xlabel("HDI")
plt.ylabel("Access to clean water and sanitation")
plt.show()
```


Most negatively Correlated
Better transport and roads
HDI
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.4
0.5
0.6
0.7
0.8
0.9

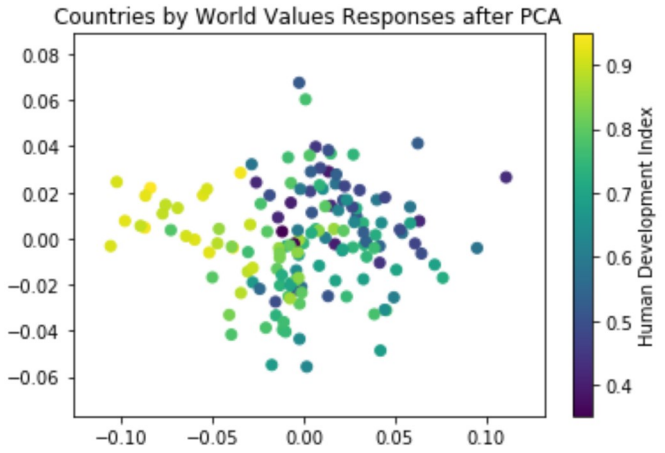

Most positively Correlated
Protecting forests rivers and oceans
HDI
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.4
0.5
0.6
0.7
0.8
0.9

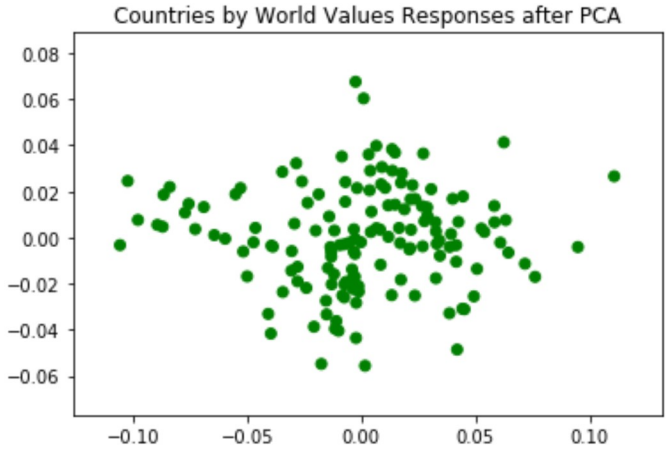

Least Correlated
Access to clean water and sanitation
HDI
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.4
0.5
0.6
0.7
0.8
0.9

d)

In [7]:
```
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']

#Part d, r: PCA #
plot_pca(values_train, hdi_train, hdi_class_train)
```

```
Predicting HDI from World Values Survey

Importing Training and Testing Data
      2015
0    0.774
1    0.420
2    0.736
3    0.754
4    0.420
5    0.924
6    0.726
7    0.592
8    0.692
9    0.448
10   0.527
11   0.899
12   0.418
13   0.418
14   0.579
15   0.921
16   0.683
17   0.781
18   0.748
19   0.775
20   0.402
21   0.579
22   0.706
23   0.638
24   0.949
25   0.925
26   0.513
27   0.739
28   0.742
29   0.767
..     ...
118  0.516
119  0.802
120  0.848
121  0.716
122  0.735
123  0.866
124  0.765
125  0.487
126  0.740
127  0.648
128  0.912
129  0.840
130  0.767
131  0.721
132  0.893
133  0.792
134  0.512
135  0.627
136  0.649
137  0.836
138  0.597
139  0.800
140  0.701
141  0.592
142  0.762
143  0.531
144  0.776
145  0.493
146  0.482
```

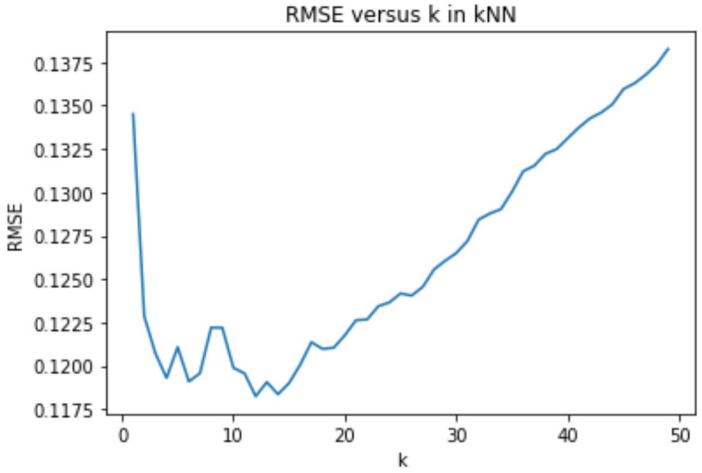

Countries by World Values Responses after PCA
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.10
−0.05
0.00
0.05
0.10
0.9
0.8
0.7
0.6
0.5
0.4
Human Development Index


Countries by World Values Responses after PCA
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.10
−0.05
0.00
0.05
0.10

e, f, g)

In [8]:
```
# Regression Grid Searches for Part e, f, and g
regression_grid_searches(training_features=values_train,training_labels=hdi_train)
```

```
Ridge Regression
RMSE: 0.12303337350607803
Pipeline(memory=None,
     steps=[('ridge', Ridge(alpha=0.02, copy_X=True, fit_intercept=True, max_ite
r=None,
   normalize=False, random_state=None, solver='auto', tol=0.001))])
Coefficients
[[ 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: 0.12602242808947525
Pipeline(memory=None,
     steps=[('lasso', Lasso(alpha=0.0002, copy_X=True, fit_intercept=True, max_i
ter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False))])
Coefficients
[ 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: 0.11824589460776892
Pipeline(memory=None,
     steps=[('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='
minkowski',
          metric_params=None, n_jobs=1, n_neighbors=12, p=2,
          weights='uniform'))])
```



g)

Ridge Regression RMSE: 0.12303337350607803

[[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: 0.12602242808947525

[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.]

Lasso regression gives more 0 weights. The above respective sets of coefficients show this.

h)

- How would you adapt the k nearest neighbors algorithm for a regression problem?

Let us use a weighted average of the k nearest neighbors, weighted by the inverse of their distances.

Algorithm:

- Compute euclidian distance from i to the labeled data points.
- Order the labeled data points by increasing distance
- Use cross validation to find optimal number of k nearest neighbors using RMSE.
- Calculate inverse distance weighted average with the k-nearest multivariate neighbors.

Or

"Take average of the k closest regression values"

i)

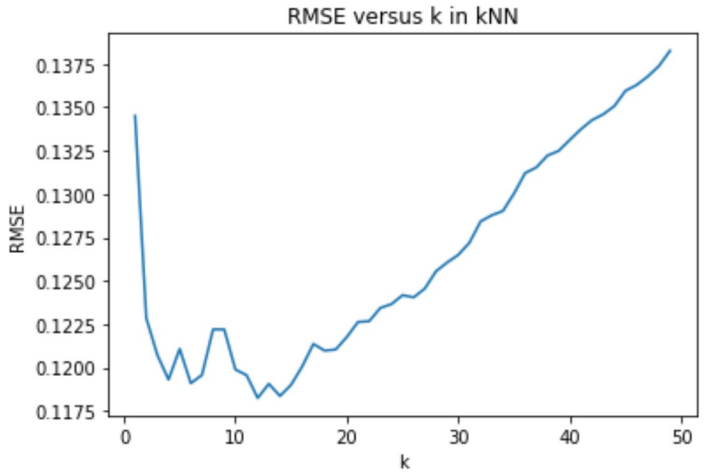
In [9]:
```
from sklearn.neighbors import NearestNeighbors

countries_train = pd.read_csv('world-values-train2.csv')
countries_train = countries_train['Country']

# Nearest neighbors to the US. n_neighbors is 8 because the first entry is the US.
nbrs = NearestNeighbors(n_neighbors=8).fit(values_train)
distances, indices = nbrs.kneighbors([values_train.iloc[45]])

print('These are the 7 nearest neighbors of the USA in order.')
print()
for i in indices:
    print(countries_train[i])
```

```
These are the 7 nearest neighbors of the USA in order.

45      United States
90            Ireland
61     United Kingdom
37            Belgium
108           Finland
69              Malta
132           Austria
110            France
Name: Country, dtype: object
```

j)

In [10]:
```
print("k Nearest Neighbors Regression")
grid = _rmse_grid_search(values_train, hdi_train,
            k_nearest_neighbors_regression_pipeline,
            regression_knn_parameters, 'knn')
```

```
k Nearest Neighbors Regression
RMSE: 0.11824589460776892
Pipeline(memory=None,
     steps=[('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='
minkowski',
          metric_params=None, n_jobs=1, n_neighbors=12, p=2,
          weights='uniform'))])
```



Best value of k: 12

RMSE at this point: 0.11824589460776892

3.k

- Explain your plot in (j) in terms of bias and variance.

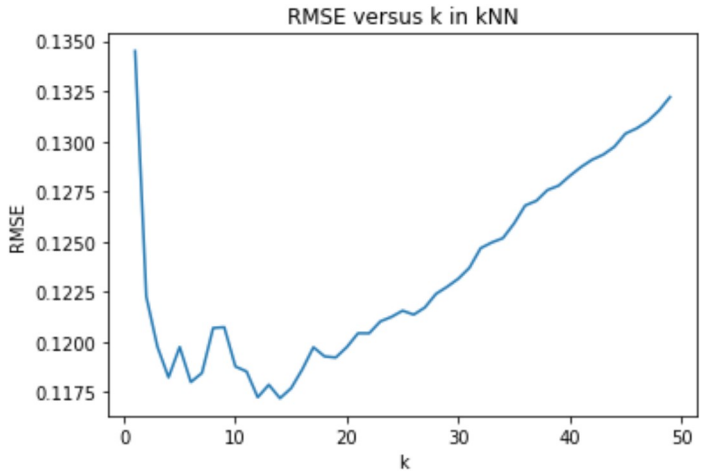
In general, a large K value is more precise as it reduces the overall noise; however, the compromise is that the distinct boundaries within the feature space are blurred. 10 is typically the best k in practice, however cross validation should be used to find the optimal.

When we increase k the bias decreases to a certain point (around 20). We get closer to the true model as we get to this point, but then the error starts to increase because are algorithm is considering too many data points. This increases bias for our model, but more specifically variance error is increasing because our algorithm has increasingly higher sensitivity to small fluctuations in the dataset. Our algorithm begins to model this random noise and we don't get the correct output.

l)

In [11]:
```
regression_knn_parameters = {
    # 'pca__n_components': np.arange(1, 17),

    'knn__n_neighbors': np.arange(1, 50),

    # Apply uniform weighting vs k for k Nearest Neighbors Regression (Part a-k)
    #'knn__weights': ['uniform']

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

print("k Nearest Neighbors Regression")
grid = _rmse_grid_search(values_train, hdi_train,
            k_nearest_neighbors_regression_pipeline,
            regression_knn_parameters, 'knn')
```

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



Best value of k: 14

RMSE at this point: 0.1171925270311745

m)

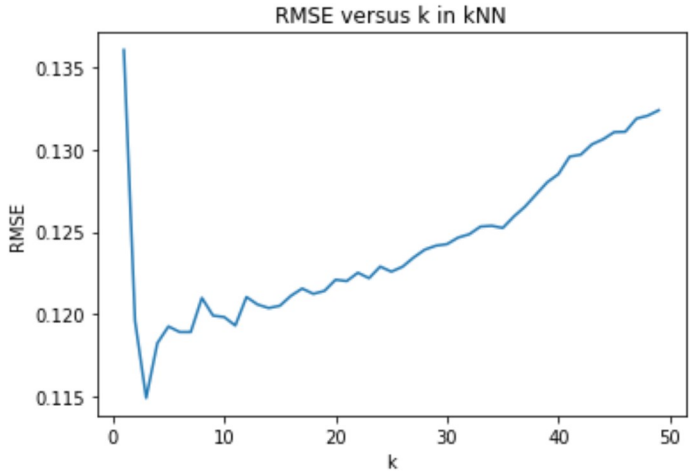
In [12]:
```
# Nearest neighbors to the US. n_neighbors is 8 because the first entry is the US.
scaled_values_train = StandardScaler().fit_transform(values_train)
nbrs = NearestNeighbors(n_neighbors=8).fit(scaled_values_train)
distances, indices = nbrs.kneighbors([pd.DataFrame(data=scaled_values_train).iloc[4
5]])

print('These are the 7 nearest neighbors of the USA in order. SCALED')
print()
for i in indices:
    print(countries_train[i])
```

```
These are the 7 nearest neighbors of the USA in order. SCALED

45      United States
90            Ireland
61     United Kingdom
108           Finland
37            Belgium
69              Malta
110            France
132           Austria
Name: Country, dtype: object
```

n)

In [13]:
```
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsRegressor

k_nearest_neighbors_regression_pipeline = Pipeline(
        [
            # Apply PCA to k Nearest Neighbors Regression
            # ('pca', PCA()),

            # Apply scaling to k Nearest Neighbors Regression
            ('scale', StandardScaler()),

            ('knn', KNeighborsRegressor())
        ]
    )

print("k Nearest Neighbors Regression")
grid = _rmse_grid_search(values_train, hdi_train,
            k_nearest_neighbors_regression_pipeline,
            regression_knn_parameters, 'knn')
```

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



Best value of k: 3

RMSE at this point: 0.11488547357936414

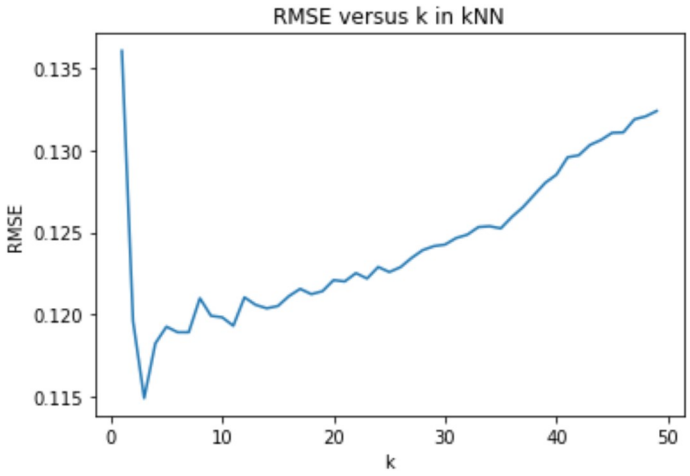
o)

Scaling the features uniformly does not appear to help.

p)

In [14]:
```
values_train, hdi_train, values_test = import_world_values_data()

k_nearest_neighbors_regression_pipeline = Pipeline(
        [
            # Apply PCA to k Nearest Neighbors Regression
            # ('pca', PCA()),

            # Apply scaling to k Nearest Neighbors Regression
            ('scale', StandardScaler()),

            ('knn', KNeighborsRegressor())
        ]
    )

grid = _rmse_grid_search(values_train, hdi_train,
            k_nearest_neighbors_regression_pipeline,
            regression_knn_parameters, 'knn')

print(grid.predict(values_test))
```

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




```
[[0.5109997 ]
 [0.61570753]
 [0.68331113]
 [0.67852229]
 [0.7718753 ]
 [0.60419024]
 [0.65159897]
 [0.67945579]
 [0.73105346]
 [0.8053592 ]
 [0.67938174]
 [0.66520881]
 [0.7793257 ]
 [0.70584762]
 [0.66568196]
 [0.56691101]
 [0.69252235]
 [0.72706267]
 [0.70498736]
 [0.77513888]
 [0.63384268]
 [0.65089538]
 [0.39709827]
 [0.73442026]
 [0.7583716 ]
 [0.71157266]
 [0.44080564]
 [0.90460366]
 [0.62818537]
 [0.65208228]
 [0.6630599 ]
 [0.86539214]
 [0.72870152]
 [0.90989302]
 [0.924362  ]
 [0.8432591 ]
 [0.49853809]
 [0.80810577]]
```

q)

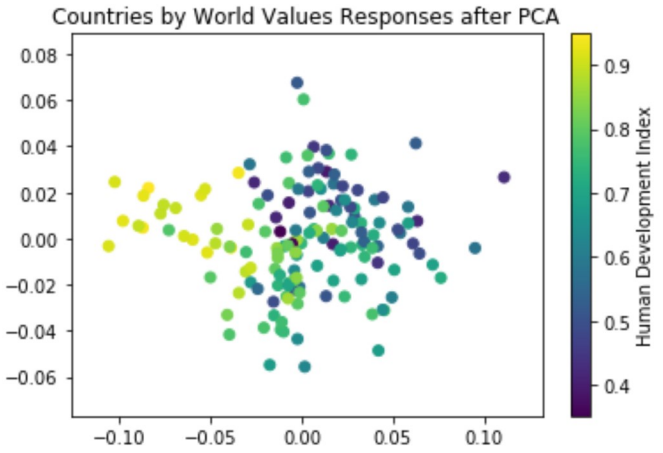
1/K. Because if the data is split perfectly evenly into k classes, your best estimator will have an accuracy of 1/k. In any other case, in which one of the classes has less than a proportion 1/k of the data, we know that there exists another class that has greater than a proportion 1/k of the data, so we know this class would provide a better naive classifier than the smaller proportion. This classifier may not even be the best but it is a lower bound that is higher than 1/k because. This is because we know that if we picked this classifier, we would have an accuracy greater than 1/k because we would get greater than 1/k of the data points classified correctly.

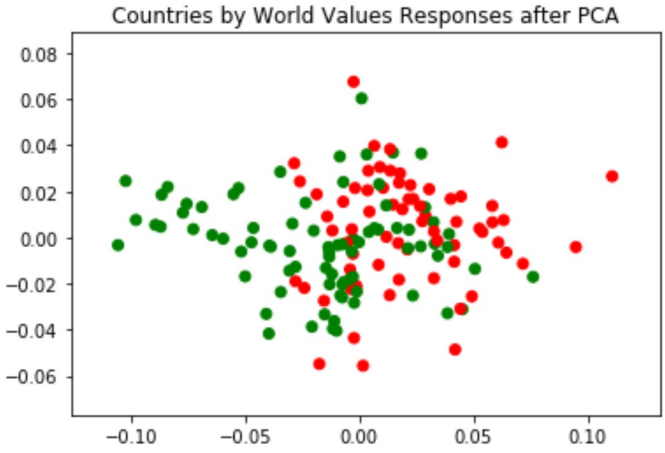
r)

In [15]:
```
# Classification Data #
hdi_class_train = hdi_train['2015'].apply(hdi_classification)

#Part d, r: PCA #
plot_pca(values_train, hdi_train, hdi_class_train)
```

	2015
0	0.774
1	0.420
2	0.736
3	0.754
4	0.420
5	0.924
6	0.726
7	0.592
8	0.692
9	0.448
10	0.527
11	0.899
12	0.418
13	0.418
14	0.579
15	0.921
16	0.683
17	0.781
18	0.748
19	0.775
20	0.402
21	0.579
22	0.706
23	0.638
24	0.949
25	0.925
26	0.513
27	0.739
28	0.742
29	0.767
..	...
118	0.516
119	0.802
120	0.848
121	0.716
122	0.735
123	0.866
124	0.765
125	0.487
126	0.740
127	0.648
128	0.912
129	0.840
130	0.767
131	0.721
132	0.893
133	0.792
134	0.512
135	0.627
136	0.649
137	0.836
138	0.597
139	0.800
140	0.701
141	0.592
142	0.762
143	0.531
144	0.776
145	0.493
146	0.482
147	0.498

[148 rows x 1 columns]





s)

Badly because using PCA there are not clear decision boundaries between the data, So if you would we would need very soft margins.

t)

In [16]:
```
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)
}

svm_classification_pipeline = Pipeline(
        [
            # Apply PCA to SVM Classification
            #('pca', PCA()),

            # Apply scaling to SVM Classification
            #('scale', StandardScaler()),

            ('svm', SVC())
        ]
    )

_accuracy_grid_search(values_train, hdi_class_train,
                        svm_classification_pipeline,
                        classification_svm_parameters)
```

```
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))])
```

Out[16]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=Pipeline(memory=None,
     steps=[('svm', SVC(C=1.0, 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))]),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'svm__C': array([ 1.,  2., ..., 98., 99.]), 'svm__kernel': ['
linear']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

u)

In [17]:
```
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)
}

svm_classification_pipeline = Pipeline(
        [
            # Apply PCA to SVM Classification
            ('pca', PCA()),

            # Apply scaling to SVM Classification
            ('scale', StandardScaler()),

            ('svm', SVC())
        ]
    )

_accuracy_grid_search(values_train, hdi_class_train,
                        svm_classification_pipeline,
                        classification_svm_parameters)

print()
print("Nope, it got worse")
```

```
Accuracy: 0.6959459459459459
Pipeline(memory=None,
     steps=[('pca', PCA(copy=True, iterated_power='auto', n_components=None, ran
dom_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_weig
ht=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))])


Nope, it got worse
```

v)

In [18]:
```
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)
}

_accuracy_grid_search(values_train, hdi_class_train,
                          svm_classification_pipeline,
                          classification_svm_parameters)
```

```
Accuracy: 0.6891891891891891
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))])
```

Out[18]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=Pipeline(memory=None,
     steps=[('svm', SVC(C=1.0, 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))]),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'svm__C': array([ 1.,  2., ..., 98., 99.]), 'svm__kernel': ['
rbf']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

w)

In [19]:
```
k_nearest_neighbors_classification_pipeline = Pipeline(
        [
            # Apply scaling to k Nearest Neighbors Classification
            # ('scale', StandardScaler()),

            ('knn', KNeighborsClassifier())
        ]
    )

_accuracy_grid_search(values_train, hdi_class_train,
                        k_nearest_neighbors_classification_pipeline,
                        classification_knn_parameters)

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

            ('knn', KNeighborsClassifier())
        ]
    )

_accuracy_grid_search(values_train, hdi_class_train,
                        k_nearest_neighbors_classification_pipeline,
                        classification_knn_parameters)

print()
print("YES, SCALING HELPS")
```

```
Accuracy: 0.7635135135135135
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'))])

Accuracy: 0.7702702702702703
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'))])


YES, SCALING HELPS
```

x)

I would predict it to be around 0.72 for our class.

y)

This could potentially work quite well, if we used the measured sensor data and used the data in kNN regression to predict the location.

z)

We explored kNN regressoin. We looked at its shortcomings that occur when we use too many data points. This helped me to understand where it is most useful.