CS 189 Introduction to Machine Learning
Fall 2017 HW11

This homework is due **Monday, November 13 at 10pm.**

## 1 Getting Started

You may typeset your homework in latex or submit neatly handwritten and scanned solutions. Please make sure to start each question on a new page, as grading (with Gradescope) is much easier that way! Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, "HW[n] Write-Up"
2. Submit all code needed to reproduce your results, "HW[n] Code".
3. Submit your test set evaluation results, "HW[n] Test Set".

After you've submitted your homework, be sure to watch out for the self-grade form.

(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

Alone - Alone
Comment: N/A

(b) Please copy the following statement and sign next to it:

*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.*

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

Problem # 2:

a.) $\langle f, g \rangle_{\mathcal{H}} = \langle g, f \rangle_{\mathcal{H}}$

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{m=1}^{M} \sum_{s=1}^{S} \alpha_m \beta_s k(y_m, x_s)$$

$$= \sum_{s=1}^{S} \sum_{m=1}^{M} \beta_s \alpha_m k(x_s, y_m) = \langle g, f \rangle_{\mathcal{H}}$$

$\langle af, g \rangle_{\mathcal{H}} = a \langle f, g \rangle_{\mathcal{H}}$

$$f = \sum_{m=1}^{M} \alpha_m k(x, y_m) \Rightarrow af = \sum_{m=1}^{M} \underbrace{a\alpha_m}_{\alpha'_m} k(x, y_m)$$

$$\Rightarrow \langle af, g \rangle_{\mathcal{H}} = \sum_{m=1}^{M} \sum_{s=1}^{S} \alpha'_m \beta_s k(y_m, x_s)$$

$$= \sum_{m=1}^{M} \sum_{s=1}^{S} a\alpha_m \beta_s k(y_m, x_s)$$

$$= a \sum_{m=1}^{M} \sum_{s=1}^{S} \alpha_m \beta_s k(y_m, x_s)$$

$$= a \langle f, g \rangle_{\mathcal{H}}$$

$\langle f+g, g \rangle_{\mathcal{H}} = \langle f, g \rangle_{\mathcal{H}} + \langle h, g \rangle_{\mathcal{H}}$

$$f + g = \sum_{m=1}^{M} \alpha_m k(x, y_m) + \sum_{n=1}^{N} \gamma_n k(x, z_n)$$

$$= \sum_{i=1}^{M+N} \delta_i k(x, w_i) \quad \text{where} \begin{cases} \delta_i = \alpha_i & \text{if } i \leq M \\ \delta_i = \gamma_{i-M} & \text{if } i > M \end{cases}$$

$$\begin{cases} w_i = y_i & \text{if } i \leq M \\ w_i = z_{i-M} & \text{if } i > M \end{cases}$$

$$\Rightarrow \langle f+g, h \rangle_{\mathcal{H}} = \sum_{i=1}^{M+N} \sum_{s=1}^{S} \delta_i \beta_s k(w_i, x_s)$$

$$\sum_{s=1}^{S} \beta_s \left( \sum_{m=1}^{M} \alpha_m k(y_m, x_s) + \sum_{n=1}^{N} \gamma_n k(z_n, x_s) \right)$$

$$= \sum_{s=1}^{S} \sum_{m=1}^{M} \alpha_m \beta_s k(y_m, x_s) + \sum_{s=1}^{S} \sum_{n=1}^{N} \gamma_n \beta_s (z_n, x_s)$$

$$= \langle f, g \rangle_{\mathcal{H}} + \langle h, g \rangle_{\mathcal{H}}$$

$\langle f, f \rangle_{\mathcal{H}} \geq 0$

$$\langle f, f \rangle = \sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i \alpha_j k(y_i, y_j)$$

$$= \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_m \end{bmatrix} K \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix} = \alpha^T K \alpha$$

$K > 0 \Rightarrow \alpha^T K \alpha > 0 \quad \forall \alpha^T \neq 0^T$

$\alpha^T K \alpha = 0$ iff $\alpha^T = 0^T$

i.e $\langle f, f \rangle = 0$ iff $f = 0$

The norm of the function $f$ is

$$\|f\|_{\mathcal{H}} = \sqrt{\langle f, f \rangle_{\mathcal{H}}} = \sqrt{\alpha^T K \alpha}$$

(b) $\langle k(x, \cdot), k(y, \cdot) \rangle_{\mathcal{H}} = k(x, y)$

Let $f(\cdot) = \sum_{m=1}^{M} \alpha_m k(\cdot, x) = k(\cdot, x) \quad (\alpha_m = 1, m = 1)$

$g(\cdot) = \sum_{m=1}^{S} \beta_s k(\cdot, y) = k(\cdot, y) \quad (\beta_s = 1, s = 1)$

$\Rightarrow \langle f, g \rangle = \langle k(\cdot, x), k(\cdot, y) \rangle = k(x, y)$

$= \langle k(x, \cdot), k(y, \cdot) \rangle$

$$\langle k(\cdot, x_i), f \rangle_H = \langle k(\cdot, x_i), \sum_{m=1}^{M} \alpha_m k(x, y_m) \rangle$$

$$= \sum_{m=1}^{M} \alpha_m \underbrace{\langle k(\cdot, x_i), k(x, y_m) \rangle}_{k(x_i, y_m)}$$

$$= \sum_{m=1}^{M} \alpha_m k(x_i, y_m) = f(x_i)$$

(c)

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

We define $M = \{ \sum_{n=1}^{N} \alpha_n k(x, x_n) \}$ to be the subspace of interest, that is the subspace that if $h \in M$,

$L(y_i, h(x_i)) = 0$ for $i = 1 \ldots n$

We can always find this subspace given $K > 0$. Why?

Because given n value $t_1, t_2 \ldots t_n$ s.t

$$L(y_i, t_i) = 0$$

we can solve the system of equations

$$h(x_i) = t_i$$

$$\sum_{j=1}^{N} \alpha_j k(x_i, x_j) = t_i$$

$$K\alpha = t \qquad (*)$$

since $K$ is invertible $\Rightarrow$ $(*)$ always has solution $\alpha$

$\alpha$ exists $\rightarrow$ $M$ exists

we write: $f = m + g$

The objective function becomes:

$$\min_{f \in \mathcal{H}} F = \min_{f \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} L(y_i, m+g) + \lambda \underbrace{\| m+g \|_{\mathcal{H}}^2}$$

$$\begin{aligned} &\langle m+g, m+g \rangle \\ &= \langle m, m \rangle + \cancel{\langle m, g \rangle} + \cancel{\langle g, m \rangle} + \langle g, g \rangle \\ &= \langle m, m \rangle + \langle g, g \rangle \end{aligned}$$

$$L(y_i, m+g) \geq L(y_i, m) = 0$$

$$\Rightarrow \quad F \geq \frac{1}{N} \sum_{i=1}^{N} L(y_i, m) + \langle m, m \rangle + \langle g, g \rangle$$

$$\geq \frac{1}{N} \sum_{i=1}^{N} L(y_i, m) + \langle m, m \rangle$$

$$\min F = \frac{1}{N} \sum_{i=1}^{N} L(y_i, m) + \langle m, m \rangle$$

iff $f = m$, i.e. $f \in M$

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

(d) $L(y, f(x)) = \max(0, 1 - y f(x))$

$$\min_{f \in H} \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i f(x_i)) + \lambda \underbrace{\|f\|_H^2}_{\alpha^T K \alpha}$$

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

again, we can always find subspace $M = \{\sum_{n=1}^{N} \alpha_i k(x, x_n) : \alpha_i \in R\}$

$\Rightarrow$ kernel SVM:

$$\min_{\alpha \in R^N} \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \alpha^T K \alpha$$

(e) General optimization problem:

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

$$L(y_i, f(x_i)) = (y_i - f(x_i))$$

$$(*) \Rightarrow \min_{f \in H} \frac{1}{N} \underbrace{\sum_{i=1}^{N} (y_i - f(x_i))_2^2}_{\|Y - f(X)\|_2^2} + \lambda \underbrace{\|f\|_H^2}_{\alpha^T K \alpha}$$

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

$$\Rightarrow f(X) = K\alpha \qquad X = [x_1 \dots x_n]^T$$

$(*) \Rightarrow \min_{\alpha \in R^d} \frac{1}{N} \|Y - K\alpha\|_2^2 + \alpha^T K \alpha$

let $F = \frac{1}{N}\|Y - K\alpha\|_2^2 + \lambda \alpha^T K \alpha$

$$\frac{dF}{d\alpha} = \frac{d\,\frac{1}{N}\left(Y^TY - Y^TK\alpha - \alpha^TK^TY + \alpha^TK^TK\alpha\right) + \lambda\alpha^TK\alpha}{d\alpha}$$

$$= \frac{1}{N}\left(0 - 2K^TY + 2K^TK\alpha\right) + 2\lambda K^T\alpha = 0$$

$$K^TK\alpha - K^TY + 2N\lambda K^T\alpha = 0$$

$$K^T\left(K\alpha + \lambda N I_N \alpha\right) = K^TY$$

one solution is $K\alpha + \lambda N I_N \alpha = Y$

$$\left(K + \lambda N I_N\right)\alpha = Y$$

$$\alpha = \left(K + \lambda N I_N\right)^{-1} Y$$

(f) Consider Tikhonov:

$$\min_w \frac{1}{N}\|Y - Xw\|_2^2 + \Gamma\|w\|_2^2$$

we know that the solution is:

$$w = \left(X^TX - N\Gamma\right)^{-1}X^TY$$

we can write:

$$\left(X^TX - N\Gamma\right)w = X^TY$$

$$X^TXw - N\Gamma w = X^TY \qquad (*)$$

$$N\Gamma w = X^TXw - X^TY = X^T(Xw - Y)$$

$$w = \frac{1}{N}\Gamma^{-1}X^T(Xw - Y) = \frac{1}{N}\lambda\Gamma^{-1}X^T\underbrace{\frac{(Xw - Y)}{\lambda}}_{a}$$

$$= \frac{1}{N}\lambda\Gamma^{-1}X^Ta$$

Substitute $w = \frac{1}{N}\lambda\Gamma^{-1}X^T\alpha$ into (*)

$$(*) \Rightarrow \frac{1}{N}X^TX\,\lambda\Gamma^{-1}X^T\alpha - \Gamma\lambda\Gamma^{-1}X^T\alpha = X^TY$$

$$X^T\left(X\lambda\Gamma^{-1}X^T\alpha - N\lambda I\alpha\right) = X^TY$$

one solution is $X\lambda\Gamma^{-1}X^T\alpha - N\lambda I\alpha = Y$

$$\alpha = \left(X\lambda\Gamma^{-1}X^T - N\lambda I\right)^{-1}Y$$

where

$$X = \begin{bmatrix} 1 & x_1 & x_2 & x_1x_2 & x_1^2 & x_2^2 \\ & \vdots & & & & \end{bmatrix}_{N\times 6}$$

Compare to kernel ridge regression

$$\alpha = \left(K + \lambda N I\right)^{-1}Y$$

we need

$$X\lambda\Gamma^{-1}X^T = K$$

we know $k(a,b) = \begin{bmatrix}1 & \sqrt{2}a_1 & \sqrt{2}a_2 & \sqrt{2}a_1a_2 & a_1^2 & a_2^2\end{bmatrix}^T \begin{bmatrix}1\\ \sqrt{2}a_1\\ \sqrt{2}a_1\\ \sqrt{2}a_1a_2\\ a_1^2\\ a_2^2\end{bmatrix}$

(for $\text{len}(a) = \text{len}(b) = 2$)

$\Rightarrow$ $\lambda\Gamma^{-1}$ must account for coefficient

$$\Rightarrow \lambda\Gamma^{-1} = \text{diag}\left(1, \underbrace{2,\ldots,2}_{\frac{N(N+1)}{2}}, \underbrace{1,\ldots 1}_{N}\right)$$

$$\Rightarrow \Gamma = \text{diag}\left(\lambda, \underbrace{\frac{\lambda}{2},\ldots,\frac{\lambda}{2}}_{N(N+1)/2}, \underbrace{\lambda,\ldots,\lambda}_{N}\right)$$

Thus kernel ridge regression is equivalent to polynomial regression for $d=2$ with Tikhonov regularization

$$\Gamma = \operatorname{diag}\Big(\lambda, \underbrace{\tfrac{\lambda}{2}, \dots, \tfrac{\lambda}{2}}_{\frac{N(N+1)}{2}}, \underbrace{\lambda, \dots, \lambda}_{N}\Big)$$

where $N$ is len(samples)

(g) Use kernel: we have to solve $n \times n$ system of eqns
→ runtime $O(n^3)$

No use kernel: solve $O(d^p) \times O(d^p)$ system of eqns → runtime $O(d^{3p})$.

Problem # 4

Extend problem 3 (f). Find Tikhonov regularization to equate OLS polynomial regression to kernel ridge regression with $d = 3$ and $len(sample_i) = 2$.

Solution

$$a = [a_1 \; a_2] \qquad b = [b_1 \; b_2]$$

$$k(a,b) = (1 + a^T b)^3 = (1 + a_1 b_1 + a_2 b_2)^3$$

$$= (1 + 2a_1 b_1 + 2a_2 b_2 + 2a_1 b_1 a_2 b_2 + a_1^2 b_1^2 + a_2^2 b_2^2)(1 + a_1 b_1 + a_2 b_2)$$

$$= 1 + a_1 b_1 + a_2 b_2 + 2a_1 b_1 + 2a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + 2a_2 b_2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2$$

$$+ 2a_1 b_1 a_2 b_2 + 2a_1^2 b_1^2 a_2 b_2 + 2a_1 b_1 a_2^2 b_2^2 + a_1^2 b_1^2 + a_1^3 b_1^3 + a_1^2 b_1^2 a_2 b_2$$

$$+ a_2^2 b_2^2 + a_1 b_1 a_2^2 b_2^2 + a_2^3 b_2^3$$

$$= 1 + 3a_1 b_1 + 3a_2 b_2 + 3a_1^2 a_2^2 + 3a_1^2 a_2^2 + 6a_1 b_1 a_2 b_2$$

$$+ 3a_1^2 b_1^2 a_2 b_2 + 3a_1 b_1 a_2^2 b_2^2 + a_1^3 b_1^3 + a_2^3 b_2^3$$

$$\Rightarrow \Gamma = \begin{bmatrix} \lambda & & & & & & & \\ & \frac{a}{3} & & & & & & \\ & & \frac{\lambda}{3} & & & & & \\ & & & \frac{\lambda}{3} & & & & \\ & & & & \frac{\lambda}{3} & & & \\ & & & & & \frac{\lambda}{6} & & \\ & & & & & & \frac{\lambda}{3} & \\ & & & & & & & \frac{\lambda}{3} \\ & & & & & & & \frac{a}{1} \; \frac{\lambda}{1} \end{bmatrix}$$

In [1]:
```
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_la
bels['2015']), 4))
    print(correlations)
    print()

    # Identify three features
    feature_list = list(training_features.columns)
    feature_correlation = dict( zip(feature_list, correlations) )
    positive_correlation = max( feature_correlation, key=feature_correlation
.get )
    negative_correlation = min( feature_correlation, key=feature_correlation
.get )
    least_correlation = min( feature_correlation, key=lambda x: abs(feature_
correlation.get(x)) )
    print("Most positively correlated:")
    print(positive_correlation + ": " + str(feature_correlation.get(positive
_correlation)))
    plot_hdi_vs_feature(training_features, training_labels, positive_correla
tion,
                        'green', 'HDI versus ' + positive_correlation)
    print()
    print("Most negatively correlated:")
    print(negative_correlation + ": " + str(feature_correlation.get(negative
_correlation)))
    plot_hdi_vs_feature(training_features, training_labels, negative_correla
tion,
                        'magenta', 'HDI versus ' + negative_correlation)
    print()
    print("Least correlated:")
    print(least_correlation + ": " + str(feature_correlation.get(least_corre
lation)))
    plot_hdi_vs_feature(training_features, training_labels, least_correlatio
n,
                        'blue', 'HDI versus ' + least_correlation)
    print()
    print("Observation: For most positively correlated HDI-feature, the poin
ts spread in forward flash shape (/)\n" +
        "For most negatively correlated HDI-feature, the points spread in ba
ckward flash shape (/)\n" +
        "For least correlated HDI-feature, the points spread in C shape")
    print()


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 set

    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["2015"])
    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 plot_pca_2(training_features,
               training_labels,
               training_classes):
    # Run PCA on training_features
    pca = PCA()
    transformed_features = pca.fit_transform(training_features)

    # Plot countries by first two PCA dimensions, color by class
    training_colors = training_classes.apply(lambda x: 'green' if x else 're
d')
    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 (Low-High HDI)'
)
    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 [2]:
```
import numpy as np

regression_ridge_parameters = {
    'ridge__alpha': np.arange(0.001, 1.0, 0.001)
}

regression_lasso_parameters = {
    'lasso__alpha': np.arange(0.00001, 0.01, 0.00001)
}

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']
}

regression_knn_weighted_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_svm_pca_scale_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_svm_rbf_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]:
```
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())
        ]
    )

k_nearest_neighbors_regression_scaled_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())
        ]
    )


svm_classification_pca_scale_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())
        ]
    )

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

            ('knn', KNeighborsClassifier())
        ]
    )
```

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

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

from math import sqrt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

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

from world_values_utils import import_world_values_data
from world_values_utils import hdi_classification
from world_values_utils import calculate_correlations
from world_values_utils import plot_pca
from world_values_utils import plot_pca_2

from world_values_pipelines import ridge_regression_pipeline
from world_values_pipelines import lasso_regression_pipeline
```

```
from world_values_pipelines import k_nearest_neighbors_regression_pipeline
from world_values_pipelines import k_nearest_neighbors_regression_scaled_pipel
ine
from world_values_pipelines import svm_classification_pipeline
from world_values_pipelines import svm_classification_pca_scale_pipeline
from world_values_pipelines import k_nearest_neighbors_classification_pipeline
from world_values_pipelines import k_nearest_neighbors_classification_scale_pi
peline

from world_values_parameters import regression_ridge_parameters
from world_values_parameters import regression_lasso_parameters
from world_values_parameters import regression_knn_parameters
from world_values_parameters import regression_knn_weighted_parameters
from world_values_parameters import classification_svm_parameters
from world_values_parameters import classification_svm_pca_scale_parameters
from world_values_parameters import classification_svm_rbf_parameters
from world_values_parameters import classification_knn_parameters


def main():
    print("========== Question 3.(a) ==========")
    print('Done filling out the "Berkeley F2017 Values Survey"')
    print()

    print("Predicting HDI from World Values Survey")
    print()

    # Import Data #
    print("Importing Training and Testing Data")
    values_train, hdi_train, values_test = import_world_values_data()

    # Center the HDI Values #
    hdi_scaler = StandardScaler(with_std=False)
    hdi_shifted_train = hdi_scaler.fit_transform(hdi_train)

    # Classification Data #
    hdi_class_train = hdi_train['2015'].apply(hdi_classification)

    # Data Information #
    print('Training Data Count:', values_train.shape[0])
    print('Test Data Count:', values_test.shape[0])
    print()

    # Calculate Correlations #
    print("========== Question 3.(b)(c) ==========")
    correlations = calculate_correlations(values_train, hdi_train)

    # PCA #
    print("========== Question 3.(d) ==========")
    plot_pca(values_train, hdi_train, hdi_class_train)
    print()

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

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

```
    print("We try not to scale each feature to have unit variance but having r
ange [0, 1]" +
          "That is, for each feature, we map value using function y = (x-a)/(b-
a)" +
          "where a is the min value of the feature and b is the max value of the
 feature")
    mapping = lambda x, a, b: (x-a)/(b-a)
#     mapped_values_train = scaler.fit_transform(values_train)
#     mapped_values_test = scaler.transform(values_test)
#     knn_map = KNeighborsRegressor(n_neighbors=3, weights="distance")
#     knn_map.fit(mapped_values_train, hdi_train["2015"])
#     knn.predict(scaled_values_test)
    print("It does not helps much")
    print()

    print("========== Question 3.(p) ==========")
    print("The best model so far is kNN with k = 3, weighted neighbors and sca
ling")
    scaler = StandardScaler()
    scaled_values_train = scaler.fit_transform(values_train)
    scaled_values_test = scaler.transform(values_test)
    knn = KNeighborsRegressor(n_neighbors=3, weights="distance")
    knn.fit(scaled_values_train, hdi_train["2015"])
    hdi_test = knn.predict(scaled_values_test)
    with open("submission.txt", "w") as f:
        for i in hdi_test:
            f.write(str(i)[:6] + "\n")
    print("Done and submitted file submission.txt to gradescope.")
    print()

    print("========== Question 3.(q) ==========")
    print("The best naive classifier will assign all points to the most possib
le class.\n" +
          "The answer is 1/k.")
    print()

    # PCA for Classification#
    # print("========== Question 3.(r) ==========")
    # plot_pca_2(values_train, hdi_train, hdi_class_train)
    # print()

    print("========== Question 3.(s) ==========")
    print("The SVM may not work well, since a lot of points mingle with each o
ther.\n" +
          "In other words, they are not linearly separate.")
    print()

    # Classification Grid Searches #
    classification_grid_searches(training_features=values_train,
                                 training_classes=hdi_class_train)

    print("========== Question 3.(x) ==========")
    print("At 110 responses, the feature numbers for Berkeley are: [33,30,15,5
7,51,95,44,55,36,57,22,54,31,47,36,52].")
    berkeley_110 = [33,30,15,57,51,95,44,55,36,57,22,54,31,47,36,52]
    berkeley_110 = [i / sum(berkeley_110) for i in berkeley_110]
    scaled_berkeley_110 = scaler.transform([berkeley_110])
```

```
    print("Predicted HDI: " + str(knn.predict(scaled_berkeley_110)[0]))
    print()
    print("At 162 responses, the feature numbers for Berkeley are: [49,37,19,7
9,71,139,60,83,55,84,31,83,38,73,41,72].")
    berkeley_162 = [49,37,19,79,71,139,60,83,55,84,31,83,38,73,41,72]
    berkeley_162 = [i / sum(berkeley_162) for i in berkeley_162]
    scaled_berkeley_162 = scaler.transform([berkeley_162])
    print("Predicted HDI: " + str(knn.predict(scaled_berkeley_162)[0]))
    print()
    print("At 229 responses, the feature numbers for Berkeley are: [68,46,26,1
16,98,198,83,115,78,118,39,116,58,99,64,89].")
    berkeley_229 = [68,46,26,116,98,198,83,115,78,118,39,116,58,99,64,89]
    berkeley_229 = [i / sum(berkeley_229) for i in berkeley_229]
    scaled_berkeley_229 = scaler.transform([berkeley_229])
    print("Predicted HDI: " + str(knn.predict(scaled_berkeley_229)[0]))
    print()

    print("========== Question 3.(y) ==========")
    print("Regarding the sensor location problem, we can use kNN in the same w
ay:" +
        "Basically, we are given the distances from m sensors, we can treat th
em as a vector of features" +
        "For kNN, we don't learn the model but we learn the boundaries (genera
tive model vs discrimimative model" +
        "that is, given a test point, we determined its 'distance' from k trai
ning points, then we infer its location" +
        "Implementation:" +
        "1. Create kNN model knn = KNeighborsRegressor()" +
        "2. Train model knn.fit(X, y)" +
        "3. Test model and caculate RMSE knn(X_test)" +
        "This is the basic model, we expect to tune parameter k, attempt scali
ng, attempt weighted neighbors, etc.")
    print()

    print("========== Question 3.(z) ==========")
    print("From this problems, I learned that data modelling is so painful and
 requires a lot of patience.\n" +
        "Basically, we have to try many model, and we have to search for wide
 ranges of different parameters" +
        "before come up with an acceptable model." +
        "For the nature of the problem, it looks like if our data spread out w
ith low correlations coefficient,"+
        "the kNN method works better than ridge regression or lasso regressio
n." +
        "Feedback for the problem author: The problem is very interesting and
 useful.")
    print()


def find_neighbors(training_features):
    distance_map = {}
    usa = np.array(training_features.iloc[45])
    for i in range(training_features.shape[0]):
        country = np.array(training_features.iloc[i])
        distance_map[i] = np.mean( (country - usa) ** 2 ) ** 0.5

    usa_neighbors = []
```

```
    for _ in range(8):
        index = min(distance_map, key=distance_map.get)
        distance_map.pop(index)
        usa_neighbors.append(index)
    print("Country indices: " + str(usa_neighbors[1:]))


def _rmse_grid_search(training_features, training_labels, pipeline, parameters
, technique):
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set
        pipeline: regression model specific pipeline
        parameters: regression model specific parameters
        technique: regression model's name

    Output:
        Prints best RMSE and best estimator
        Prints feature weights for Ridge and Lasso Regression
        Plots RMSE vs k for k Nearest Neighbors Regression
    """
    grid = GridSearchCV(estimator=pipeline,
                        param_grid=parameters,
                        scoring='neg_mean_squared_error')
    grid.fit(training_features,
             training_labels)
    print("RMSE:", sqrt(-grid.best_score_))
    print(grid.best_estimator_)

    # Check Ridge or Lasso Regression
    if hasattr(grid.best_estimator_.named_steps[technique], 'coef_'):
        print(grid.best_estimator_.named_steps[technique].coef_)
    else:
        # Plot RMSE vs k for k Nearest Neighbors Regression
        plt.plot(grid.cv_results_['param_knn__n_neighbors'],
                 (-grid.cv_results_['mean_test_score'])**0.5)
        plt.xlabel('k')
        plt.ylabel('RMSE')
        plt.title('RMSE versus k in kNN')
        plt.show()

    print()


def regression_grid_searches(training_features, training_labels):
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set

    Output:
        Prints best RMSE, best estimator, feature weights for Ridge and Lasso
 Regression
        Prints best RMSE, best estimator, and plots RMSE vs k for k Nearest Ne
ighbors Regression
    """
```

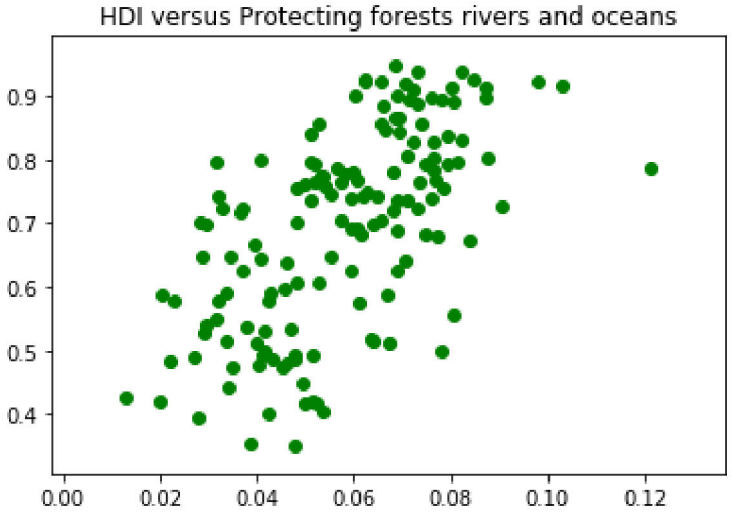
```
    print("========== Question 3.(e) ==========")
    print("Ridge Regression")
    _rmse_grid_search(training_features, training_labels,
                ridge_regression_pipeline, regression_ridge_parameters, 'ridg
e')
    print("I changed the range of hyper-parameter ridge__alpha to obtain the f
iner result. That is\n" +
        "ridge__alpha has np.arange(0.001, 1.0, 0.001) instead of np.arange(0.
01, 1.0, 0.01) given\n" +
        "The best RMSE indicated above.")
    print()

    print("========== Question 3.(f) ==========")
    print("Lasso Regression")
    _rmse_grid_search(training_features, training_labels,
                lasso_regression_pipeline, regression_lasso_parameters, 'lass
o')
    print("I changed the range of hyper-parameter lasso__alpha to obtain the f
iner result. That is\n" +
        "lasso__alpha has np.arange(0.00001, 0.01, 0.00001) instead of np.aran
ge(0.0001, 0.01, 0.0001) given\n" +
        "The best RMSE indicated above.")
    print()

    print("========== Question 3.(g) ==========")
    print("The Lasso Regression does give more 0 weights.\n" +
        "That indicates some features do not really matter in this method.")
    print()

    print("========== Question 3.(h) ==========")
    print("To deal with continuous outputs, we can weight the neigbors instead
 of treating them uniformly.\n" +
        "Say, each neighbor is weighted by its inverse distance, and we use th
e average of k-neareast-neighbor\n" +
        "weights to predict the output.")
    print()

    print("========== Question 3.(i) ==========")
    print("The 7 nearest neighbors of the USA:")
    find_neighbors(training_features)
    print("Countries: Ireland, United Kingdom, Belgium, Finland, Malta, Austri
a, France")
    print()

    print("========== Question 3.(j) ==========")
    print("k Nearest Neighbors Regression")
    _rmse_grid_search(training_features, training_labels,
                k_nearest_neighbors_regression_pipeline,
                regression_knn_parameters, 'knn')
    print("The best value of k is k = 12. The RMSE indicated above")
    print()

    print("========== Question 3.(k) ==========")
    print("When we increase k, the model goes from overfitting to fitting then
 underfitting. That is increasing k leads\n" +
        "to the increase of bias and decrease of variance because the model be
```

```
comes less flexible to accommodate ambiguous\n" +
        "points. Since more points are taken into account when considering a t
est point, the model works more consistent\n" +
        "(with lower variance), but less accurate (with higher bias) if the na
ture of training data is the mingling of data points.")
    print()

    print("========== Question 3.(l) ==========")
    print("k Nearest Neighbors Regression with weighted neighbor distances")
    _rmse_grid_search(training_features, training_labels,
                k_nearest_neighbors_regression_pipeline,
                regression_knn_weighted_parameters, 'knn')
    print("The best value of k is k = 14. The RMSE indicated above")
    print()

    print("========== Question 3.(m) ==========")
    scaler = StandardScaler()
    scaled_training_features = scaler.fit_transform(training_features)
    scaled_training_features = pd.DataFrame(scaled_training_features)
    find_neighbors(scaled_training_features)
    print("Countries: Ireland, United Kingdom, Finland, Belgium, Malta, Franc
e, Austria")
    print()
    print("Compared to (i), the neighbors just change order a little bit.")
    print()

    print("========== Question 3.(n) ==========")
    print("k Nearest Neighbors Regression with weighted neighbor distances and
 scaling the features")
    _rmse_grid_search(training_features, training_labels,
                k_nearest_neighbors_regression_scaled_pipeline,
                regression_knn_weighted_parameters, 'knn')
    print("The best value of k is k = 3. The RMSE indicated above")
    print()

#     print("========== Question 3.(o) ==========")
#     print("The best model so far is kNN with k = 3, weighted neighbors and s
caling")
#     scaler = StandardScaler()
#     scaled_values_train = scaler.fit_transform(values_train)
#     scaled_values_test = scaler.transform(values_test)
#     knn = KNeighborsRegressor(n_neighbors=3, weights="distance")
#     knn.fit(scaled_values_train, hdi_train["2015"])
#     hdi_test = knn.predict(scaled_values_test)
#     with open("submission.txt", "w") as f:
#         for i in hdi_test:
#             f.write(str(i)[:6] + "\n")
#     print("Done and submitted file submission.txt to gradescope.")
#     print()


def _accuracy_grid_search(training_features, training_classes, pipeline, param
eters):
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set
```

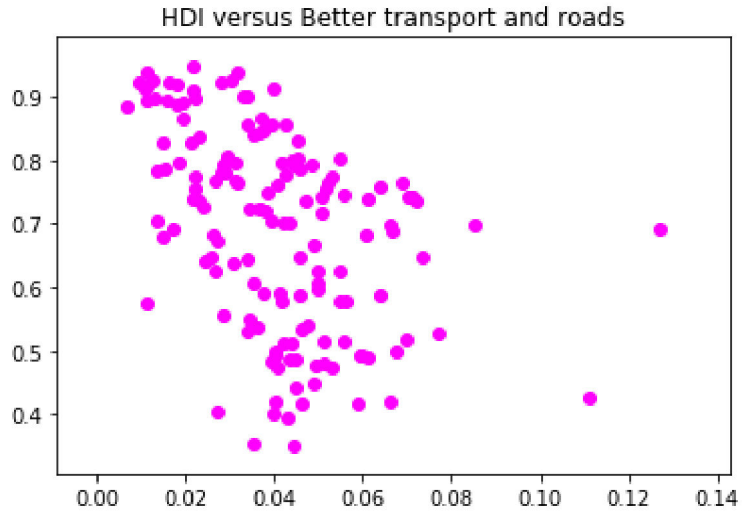
```
        pipeline: classification model specific pipeline
        parameters: classification model specific parameters

    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 Neighbor
s Classification
    """
    print("========== Question 3.(t) ==========")
    print("SVM Classification")
    _accuracy_grid_search(training_features, training_classes,
                        svm_classification_pipeline,
                        classification_svm_parameters)
    print("The accuracy indicated above.")
    print()

    print("========== Question 3.(u) ==========")
    print("SVM Classification modified by adding PCA step and Scaling step")
    _accuracy_grid_search(training_features, training_classes,
                        svm_classification_pca_scale_pipeline,
                        classification_svm_pca_scale_parameters)
    print("The accuracy indicated above. It does improve.")
    print()

    print("========== Question 3.(v) ==========")
    print("SVM Classification with kernel of radial basis function")
    _accuracy_grid_search(training_features, training_classes,
                        svm_classification_pipeline,
                        classification_svm_rbf_parameters)
    print("The accuracy indicated above.")
    print()

    print("========== Question 3.(w) ==========")
    print("k Nearest Neighbors Classification")
    _accuracy_grid_search(training_features, training_classes,
                        k_nearest_neighbors_classification_pipeline,
                        classification_knn_parameters)
    print("The accuracy indicated above.")
    print()
```

```
    print("k Nearest Neighbors Classification with Scaling")
    _accuracy_grid_search(training_features, training_classes,
                          k_nearest_neighbors_classification_scale_pipeline,
                          classification_knn_parameters)
    print("The accuracy indicated above.")
    print("Scling helps a little bit, increasing the accuracy from 0.7635 to
 0.7703.")
    print()


if __name__ == '__main__':
    main()
```

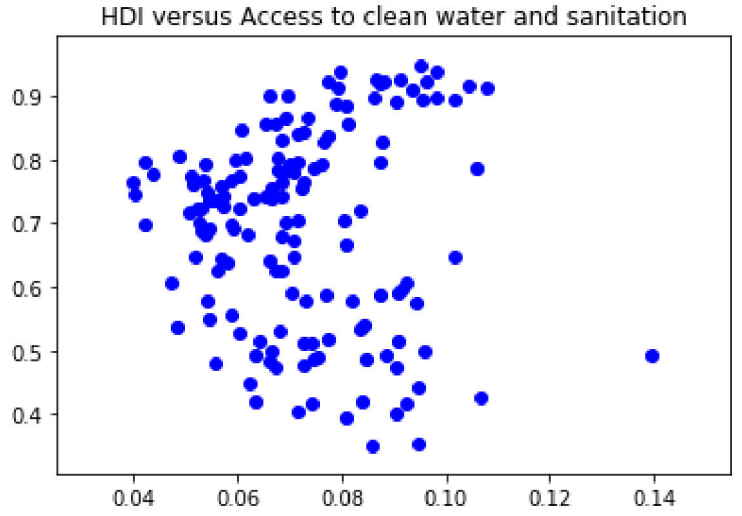
```
========== Question 3.(a) ==========
Done filling out the "Berkeley F2017 Values Survey"

Predicting HDI from World Values Survey

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

========== Question 3.(b)(c) ==========
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, -0.42199999999
999999, -0.30399999999999999, 0.32940000000000003, -0.35160000000000002, -0.2
8539999999999999, 0.19520000000000001, 0.61350000000000005, 0.143300000000000
01, 0.23810000000000001, 0.43290000000000001, 0.27650000000000002, -0.3972999
9999999999]

Most positively correlated:
Protecting forests rivers and oceans: 0.6135
```



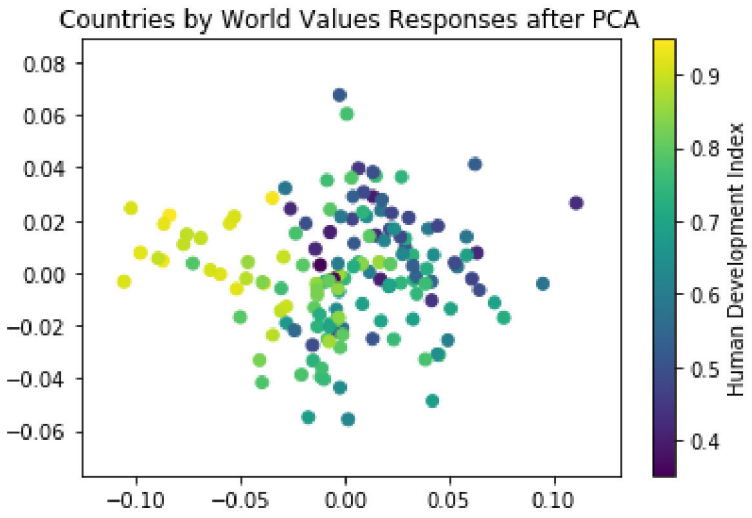

```
Most negatively correlated:
Better transport and roads: -0.4396
```




```
Least correlated:
Access to clean water and sanitation: -0.0182
```

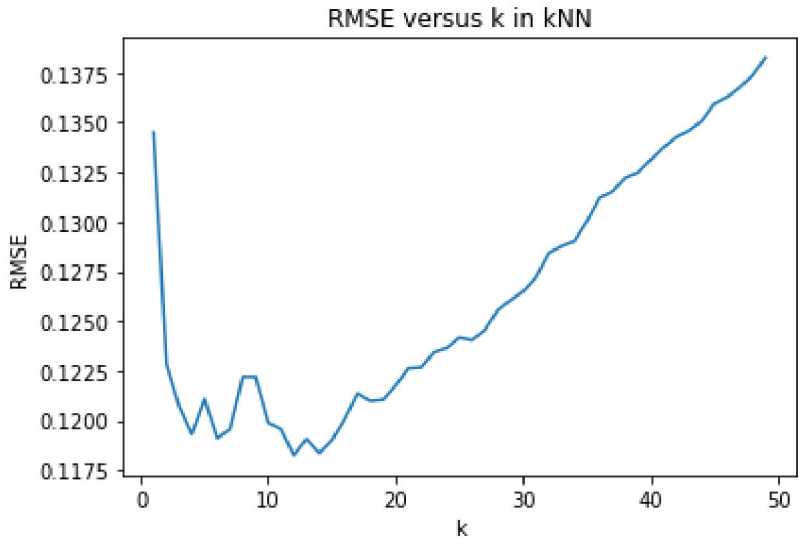



```
Observation: For most positively correlated HDI-feature, the points spread in
forward flash shape (/)
For most negatively correlated HDI-feature, the points spread in backward fla
sh shape (/)
For least correlated HDI-feature, the points spread in C shape

========== Question 3.(d) ==========
```

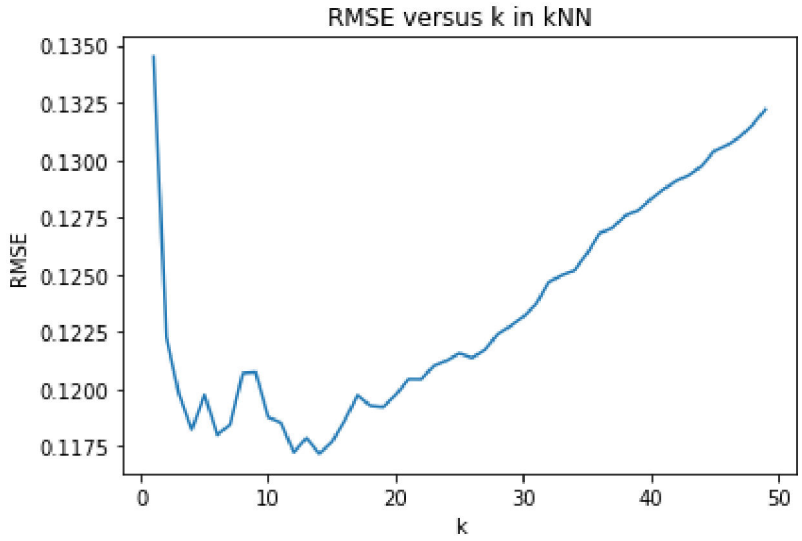

Countries by World Values Responses after PCA
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.10
−0.05
0.00
0.05
0.10
0.9
0.8
0.7
0.6
0.5
0.4
Human Development Index

```
========== Question 3.(e) ==========
Ridge Regression
RMSE: 0.12288664701062982
Pipeline(memory=None,
     steps=[('ridge', Ridge(alpha=0.014999999999999999, copy_X=True, fit_in
tercept=True,
   max_iter=None, normalize=False, random_state=None, solver='auto',
   tol=0.001))])
[[ 0.82512275 -0.73095272 -0.12512918 -1.43113028 -0.69165906 -0.88573308
   0.77408183 -0.99355445 -0.94180091  0.46182676  2.23534355 -0.15521277
   0.52742883  0.75955075  0.43529637 -0.06347831]]

I changed the range of hyper-parameter ridge__alpha to obtain the finer res
ult. That is
ridge__alpha has np.arange(0.001, 1.0, 0.001) instead of np.arange(0.01, 1.
0, 0.01) given
The best RMSE indicated above.

========== Question 3.(f) ==========
Lasso Regression
RMSE: 0.12598055237846487
Pipeline(memory=None,
     steps=[('lasso', Lasso(alpha=0.00018000000000000001, 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))])
[  2.45222329e-01  -7.34515838e-01  -0.00000000e+00  -9.89268505e-01
  -6.75194836e-01  -9.68733261e-02   3.70020452e-01  -4.14971086e-01
  -0.00000000e+00   0.00000000e+00   3.43354506e+00   0.00000000e+00
   1.61387935e-03   8.93093959e-01   3.56620212e-01  -0.00000000e+00]

I changed the range of hyper-parameter lasso__alpha to obtain the finer res
ult. That is
lasso__alpha has np.arange(0.00001, 0.01, 0.00001) instead of np.arange(0.0
001, 0.01, 0.0001) given
The best RMSE indicated above.

========== Question 3.(g) ==========
The Lasso Regression does give more 0 weights.
That indicates some features do not really matter in this method.

========== Question 3.(h) ==========
To deal with continuous outputs, we can weight the neigbors instead of trea
ting them uniformly.
Say, each neighbor is weighted by its inverse distance, and we use the aver
age of k-neareast-neighbor
weights to predict the output.

========== Question 3.(i) ==========
The 7 nearest neighbors of the USA:
Country indices: [90, 61, 37, 108, 69, 132, 110]
Countries: Ireland, United Kingdom, Belgium, Finland, Malta, Austria, Franc
e

========== Question 3.(j) ==========
k Nearest Neighbors Regression
RMSE: 0.1182458946077689
```

```
Pipeline(memory=None,
     steps=[('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, met
ric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=12, p=2,
          weights='uniform'))])
```

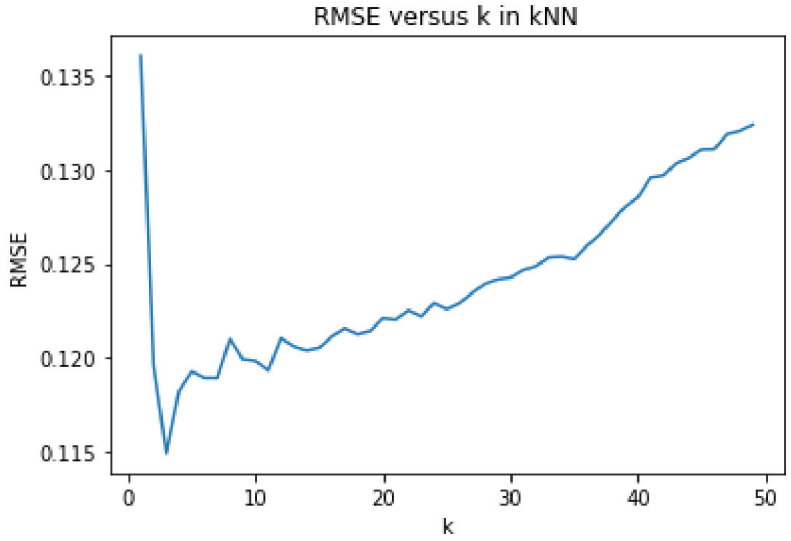



```
The best value of k is k = 12. The RMSE indicated above

========== Question 3.(k) ==========
When we increase k, the model goes from overfitting to fitting then underfitt
ing. That is increasing k leads
to the increase of bias and decrease of variance because the model becomes le
ss flexible to accommodate ambiguous
points. Since more points are taken into account when considering a test poin
t, the model works more consistent
(with lower variance), but less accurate (with higher bias) if the nature of
training data is the mingling of data points.

========== Question 3.(l) ==========
k Nearest Neighbors Regression with weighted neighbor distances
RMSE: 0.1171925270311745
Pipeline(memory=None,
     steps=[('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metri
c='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=14, p=2,
          weights='distance'))])
```




The best value of k is k = 14. The RMSE indicated above

```
========== Question 3.(m) ==========
Country indices: [90, 61, 108, 37, 69, 110, 132]
Countries: Ireland, United Kingdom, Finland, Belgium, Malta, France, Austria

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

========== Question 3.(n) ==========
k Nearest Neighbors Regression with weighted neighbor distances and scaling t
he features
RMSE: 0.11488547357936414
Pipeline(memory=None,
     steps=[('scale', StandardScaler(copy=True, with_mean=True, with_std=Tru
e)), ('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='mink
owski',
          metric_params=None, n_jobs=1, n_neighbors=3, p=2,
          weights='distance'))])
```



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

========== Question 3.(o) ==========
We try not to scale each feature to have unit variance but having range [0
, 1]That is, for each feature, we map value using function y = (x-a)/(b-a)
where a is the min value of the feature and b is the max value of the feat
ure
It does not helps much

========== Question 3.(p) ==========
The best model so far is kNN with k = 3, weighted neighbors and scaling
Done and submitted file submission.txt to gradescope.

========== Question 3.(q) ==========
The best naive classifier will assign all points to the most possible clas
s.
The answer is 1/k.

========== Question 3.(s) ==========
The SVM may not work well, since a lot of points mingle with each other.
In other words, they are not linearly separate.

========== Question 3.(t) ==========
SVM Classification
Accuracy: 0.75
Pipeline(memory=None,
     steps=[('svm', SVC(C=48.0, cache_size=200, class_weight=None, coef0=0
.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

The accuracy indicated above.

========== Question 3.(u) ==========
SVM Classification modified by adding PCA step and Scaling step
Accuracy: 0.810810810811
Pipeline(memory=None,
     steps=[('pca', PCA(copy=True, iterated_power='auto', n_components=7,
random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)), ('scale', StandardScaler(cop
y=True, with_mean=True, with_std=True)), ('svm', SVC(C=0.02, cache_size=20
0, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

The accuracy indicated above. It does improve.

========== Question 3.(v) ==========
SVM Classification with kernel of radial basis function
Accuracy: 0.689189189189
Pipeline(memory=None,
```

```
     steps=[('svm', SVC(C=98.0, cache_size=200, class_weight=None, coef0=0
.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

The accuracy indicated above.

========== Question 3.(w) ==========
k Nearest Neighbors Classification
Accuracy: 0.763513513514
Pipeline(memory=None,
     steps=[('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, m
etric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])

The accuracy indicated above.

k Nearest Neighbors Classification with Scaling
Accuracy: 0.77027027027
Pipeline(memory=None,
     steps=[('scale', StandardScaler(copy=True, with_mean=True, with_std=T
rue)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric
='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])

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

========== Question 3.(x) ==========
At 110 responses, the feature numbers for Berkeley are: [33,30,15,57,51,95
,44,55,36,57,22,54,31,47,36,52].
Predicted HDI: 0.461942596688

At 162 responses, the feature numbers for Berkeley are: [49,37,19,79,71,13
9,60,83,55,84,31,83,38,73,41,72].
Predicted HDI: 0.462173616823

At 229 responses, the feature numbers for Berkeley are: [68,46,26,116,98,1
98,83,115,78,118,39,116,58,99,64,89].
Predicted HDI: 0.461521885868

========== Question 3.(y) ==========
Regarding the sensor location problem, we can use kNN in the same way:Basi
cally, we are given the distances from m sensors, we can treat them as a v
ector of featuresFor kNN, we don't learn the model but we learn the bounda
ries (generative model vs discrimimative modelthat is, given a test point,
we determined its 'distance' from k training points, then we infer its loc
ationImplementation:1. Create kNN model knn = KNeighborsRegressor()2. Trai
n model knn.fit(X, y)3. Test model and caculate RMSE knn(X_test)This is th
e basic model, we expect to tune parameter k, attempt scaling, attempt wei
ghted neighbors, etc.
```

========== Question 3.(z) ==========
From this problems, I learned that data modelling is so painful and requir es a lot of patience.
Basically, we have to try many model, and we have to search for wide range s of different parametersbefore come up with an acceptable model.For the n ature of the problem, it looks like if our data spread out with low correl ations coefficient,the kNN method works better than ridge regression or la sso regression.Feedback for the problem author: The problem is very intere sting and useful.

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

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

from math import sqrt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

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

from world_values_utils import import_world_values_data
from world_values_utils import hdi_classification
from world_values_utils import calculate_correlations
from world_values_utils import plot_pca
from world_values_utils import plot_pca_2

from world_values_pipelines import ridge_regression_pipeline
from world_values_pipelines import lasso_regression_pipeline
from world_values_pipelines import k_nearest_neighbors_regression_pipeline
```

```
from world_values_pipelines import k_nearest_neighbors_regression_scaled_pipeline
from world_values_pipelines import svm_classification_pipeline
from world_values_pipelines import svm_classification_pca_scale_pipeline
from world_values_pipelines import k_nearest_neighbors_classification_pipeline
from world_values_pipelines import k_nearest_neighbors_classification_scale_pipeline

from world_values_parameters import regression_ridge_parameters
from world_values_parameters import regression_lasso_parameters
from world_values_parameters import regression_knn_parameters
from world_values_parameters import regression_knn_weighted_parameters
from world_values_parameters import classification_svm_parameters
from world_values_parameters import classification_svm_pca_scale_parameters
from world_values_parameters import classification_svm_rbf_parameters
from world_values_parameters import classification_knn_parameters



def main():
    print("========== Question 3.(a) ==========")
    print('Done filling out the "Berkeley F2017 Values Survey"')
    print()

    print("Predicting HDI from World Values Survey")
    print()

    # Import Data #
    print("Importing Training and Testing Data")
    values_train, hdi_train, values_test = import_world_values_data()

    # Center the HDI Values #
```

```
hdi_scaler = StandardScaler(with_std=False)
hdi_shifted_train = hdi_scaler.fit_transform(hdi_train)

# Classification Data #
hdi_class_train = hdi_train['2015'].apply(hdi_classification)

# Data Information #
print('Training Data Count:', values_train.shape[0])
print('Test Data Count:', values_test.shape[0])
print()

# Calculate Correlations #
print("========== Question 3.(b)(c) ==========")
correlations = calculate_correlations(values_train, hdi_train)

# PCA #
print("========== Question 3.(d) ==========")
plot_pca(values_train, hdi_train, hdi_class_train)
print()

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

print("========== Question 3.(o) ==========")
print("We try not to scale each feature to have unit variance but having range [0, 1]" +
   "That is, for each feature, we map value using function y = (x-a)/(b-a)" +
   "where a is the min value of the feature and b is the max value of the feature")
mapping = lambda x, a, b: (x-a)/(b-a)
```

```
mapped_values_train = scaler.fit_transform(values_train)
mapped_values_test = scaler.transform(values_test)
knn_map = KNeighborsRegressor(n_neighbors=3, weights="distance")
knn_map.fit(mapped_values_train, hdi_train["2015"])
knn.predict(scaled_values_test)
print("It does not helps much")
print()

print("========== Question 3.(p) ==========")
print("The best model so far is kNN with k = 3, weighted neighbors and scaling")
scaler = StandardScaler()
scaled_values_train = scaler.fit_transform(values_train)
scaled_values_test = scaler.transform(values_test)
knn = KNeighborsRegressor(n_neighbors=3, weights="distance")
knn.fit(scaled_values_train, hdi_train["2015"])
hdi_test = knn.predict(scaled_values_test)
with open("submission.txt", "w") as f:
    for i in hdi_test:
        f.write(str(i)[:6] + "\n")
print("Done and submitted file submission.txt to gradescope.")
print()

print("========== Question 3.(q) ==========")
print("The best naive classifier will assign all points to the most possible class.\n" +
    "The answer is 1/k.")
print()

# PCA for Classification#
# print("========== Question 3.(r) ==========")
```

```
    # plot_pca_2(values_train, hdi_train, hdi_class_train)
    # print()

    print("========== Question 3.(s) ==========")
    print("The SVM may not work well, since a lot of points mingle with each other.\n" +
        "In other words, they are not linearly separate.")
    print()

    # Classification Grid Searches #
    classification_grid_searches(training_features=values_train,
                                 training_classes=hdi_class_train)

    print("========== Question 3.(x) ==========")
    print("At 110 responses, the feature numbers for Berkeley are:
[33,30,15,57,51,95,44,55,36,57,22,54,31,47,36,52].")
    berkeley_110 = [33,30,15,57,51,95,44,55,36,57,22,54,31,47,36,52]
    berkeley_110 = [i / sum(berkeley_110) for i in berkeley_110]
    scaled_berkeley_110 = scaler.transform([berkeley_110])
    print("Predicted HDI: " + str(knn.predict(scaled_berkeley_110)[0]))
    print()
    print("At 162 responses, the feature numbers for Berkeley are:
[49,37,19,79,71,139,60,83,55,84,31,83,38,73,41,72].")
    berkeley_162 = [49,37,19,79,71,139,60,83,55,84,31,83,38,73,41,72]
    berkeley_162 = [i / sum(berkeley_162) for i in berkeley_162]
    scaled_berkeley_162 = scaler.transform([berkeley_162])
    print("Predicted HDI: " + str(knn.predict(scaled_berkeley_162)[0]))
    print()
    print("At 229 responses, the feature numbers for Berkeley are:
[68,46,26,116,98,198,83,115,78,118,39,116,58,99,64,89].")
    berkeley_229 = [68,46,26,116,98,198,83,115,78,118,39,116,58,99,64,89]
```

```
    berkeley_229 = [i / sum(berkeley_229) for i in berkeley_229]

    scaled_berkeley_229 = scaler.transform([berkeley_229])

    print("Predicted HDI: " + str(knn.predict(scaled_berkeley_229)[0]))

    print()


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

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

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

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

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

        "Implementation:" +

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

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

        "3. Test model and caculate RMSE knn(X_test)" +

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

    print()


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

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

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

        "before come up with an acceptable model." +

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

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

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

    print()
```

```
def find_neighbors(training_features):
    distance_map = {}
    usa = np.array(training_features.iloc[45])
    for i in range(training_features.shape[0]):
        country = np.array(training_features.iloc[i])
        distance_map[i] = np.mean( (country - usa) ** 2 ) ** 0.5

    usa_neighbors = []
    for _ in range(8):
        index = min(distance_map, key=distance_map.get)
        distance_map.pop(index)
        usa_neighbors.append(index)
    print("Country indices: " + str(usa_neighbors[1:]))



def _rmse_grid_search(training_features, training_labels, pipeline, parameters, technique):
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set
        pipeline: regression model specific pipeline
        parameters: regression model specific parameters
        technique: regression model's name

    Output:
        Prints best RMSE and best estimator
        Prints feature weights for Ridge and Lasso Regression
```

```
    Plots RMSE vs k for k Nearest Neighbors Regression
    """
    grid = GridSearchCV(estimator=pipeline,
                        param_grid=parameters,
                        scoring='neg_mean_squared_error')
    grid.fit(training_features,
             training_labels)
    print("RMSE:", sqrt(-grid.best_score_))
    print(grid.best_estimator_)

    # Check Ridge or Lasso Regression
    if hasattr(grid.best_estimator_.named_steps[technique], 'coef_'):
        print(grid.best_estimator_.named_steps[technique].coef_)
    else:
        # Plot RMSE vs k for k Nearest Neighbors Regression
        plt.plot(grid.cv_results_['param_knn__n_neighbors'],
                 (-grid.cv_results_['mean_test_score'])**0.5)
        plt.xlabel('k')
        plt.ylabel('RMSE')
        plt.title('RMSE versus k in kNN')
        plt.show()

    print()



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

```
    training_labels: HDI (human development index) on the training set

    Output:
        Prints best RMSE, best estimator, feature weights for Ridge and Lasso Regression
        Prints best RMSE, best estimator, and plots RMSE vs k for k Nearest Neighbors Regression
    """

    print("========== Question 3.(e) ==========")
    print("Ridge Regression")
    _rmse_grid_search(training_features, training_labels,
            ridge_regression_pipeline, regression_ridge_parameters, 'ridge')
    print("I changed the range of hyper-parameter ridge__alpha to obtain the finer result. That is\n" +
        "ridge__alpha has np.arange(0.001, 1.0, 0.001) instead of np.arange(0.01, 1.0, 0.01) given\n" +
        "The best RMSE indicated above.")
    print()

    print("========== Question 3.(f) ==========")
    print("Lasso Regression")
    _rmse_grid_search(training_features, training_labels,
            lasso_regression_pipeline, regression_lasso_parameters, 'lasso')
    print("I changed the range of hyper-parameter lasso__alpha to obtain the finer result. That is\n" +
        "lasso__alpha has np.arange(0.00001, 0.01, 0.00001) instead of np.arange(0.0001, 0.01, 0.0001)
given\n" +
        "The best RMSE indicated above.")
    print()

    print("========== Question 3.(g) ==========")
    print("The Lasso Regression does give more 0 weights.\n" +
        "That indicates some features do not really matter in this method.")
```

```
    print()


    print("========== Question 3.(h) ==========")
    print("To deal with continuous outputs, we can weight the neigbors instead of treating them uniformly.\n" +
          "Say, each neighbor is weighted by its inverse distance, and we use the average of k-neareast-neighbor\n" +
          "weights to predict the output.")
    print()


    print("========== Question 3.(i) ==========")
    print("The 7 nearest neighbors of the USA:")
    find_neighbors(training_features)
    print("Countries: Ireland, United Kingdom, Belgium, Finland, Malta, Austria, France")
    print()


    print("========== Question 3.(j) ==========")
    print("k Nearest Neighbors Regression")
    _rmse_grid_search(training_features, training_labels,
                      k_nearest_neighbors_regression_pipeline,
                      regression_knn_parameters, 'knn')
    print("The best value of k is k = 12. The RMSE indicated above")
    print()


    print("========== Question 3.(k) ==========")
    print("When we increase k, the model goes from overfitting to fitting then underfitting. That is increasing k leads\n" +
          "to the increase of bias and decrease of variance because the model becomes less flexible to accommodate ambiguous\n" +
```

```
      "points. Since more points are taken into account when considering a test point, the model works more consistent\n" +
      "(with lower variance), but less accurate (with higher bias) if the nature of training data is the mingling of data points.")
   print()

   print("========== Question 3.(l) ==========")
   print("k Nearest Neighbors Regression with weighted neighbor distances")
   _rmse_grid_search(training_features, training_labels,
            k_nearest_neighbors_regression_pipeline,
            regression_knn_weighted_parameters, 'knn')
   print("The best value of k is k = 14. The RMSE indicated above")
   print()

   print("========== Question 3.(m) ==========")
   scaler = StandardScaler()
   scaled_training_features = scaler.fit_transform(training_features)
   scaled_training_features = pd.DataFrame(scaled_training_features)
   find_neighbors(scaled_training_features)
   print("Countries: Ireland, United Kingdom, Finland, Belgium, Malta, France, Austria")
   print()
   print("Compared to (i), the neighbors just change order a little bit.")
   print()

   print("========== Question 3.(n) ==========")
   print("k Nearest Neighbors Regression with weighted neighbor distances and scaling the features")
   _rmse_grid_search(training_features, training_labels,
            k_nearest_neighbors_regression_scaled_pipeline,
            regression_knn_weighted_parameters, 'knn')
```

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

    print("========== Question 3.(o) ==========")
    print("The best model so far is kNN with k = 3, weighted neighbors and scaling")
    scaler = StandardScaler()
    scaled_values_train = scaler.fit_transform(values_train)
    scaled_values_test = scaler.transform(values_test)
    knn = KNeighborsRegressor(n_neighbors=3, weights="distance")
    knn.fit(scaled_values_train, hdi_train["2015"])
    hdi_test = knn.predict(scaled_values_test)
    with open("submission.txt", "w") as f:
        for i in hdi_test:
            f.write(str(i)[:6] + "\n")
    print("Done and submitted file submission.txt to gradescope.")
    print()



def _accuracy_grid_search(training_features, training_classes, pipeline, parameters):
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set
        pipeline: classification model specific pipeline
        parameters: classification model specific parameters

    Output:
        Prints best accuracy and best estimator of classification model
    """
```

```
    grid = GridSearchCV(estimator=pipeline,
                        param_grid=parameters,
                        scoring='accuracy')
    grid.fit(training_features, training_classes)
    print("Accuracy:", grid.best_score_)
    print(grid.best_estimator_)
    print()



def classification_grid_searches(training_features, training_classes):
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set

    Output:
        Prints best accuracy and best estimator for SVM and k Nearest Neighbors Classification
    """
    print("========== Question 3.(t) ==========")
    print("SVM Classification")
    _accuracy_grid_search(training_features, training_classes,
                        svm_classification_pipeline,
                        classification_svm_parameters)
    print("The accuracy indicated above.")
    print()

    print("========== Question 3.(u) ==========")
    print("SVM Classification modified by adding PCA step and Scaling step")
    _accuracy_grid_search(training_features, training_classes,
```

```
            svm_classification_pca_scale_pipeline,
            classification_svm_pca_scale_parameters)
print("The accuracy indicated above. It does improve.")
print()

print("========== Question 3.(v) ==========")
print("SVM Classification with kernel of radial basis function")
_accuracy_grid_search(training_features, training_classes,
            svm_classification_pipeline,
            classification_svm_rbf_parameters)
print("The accuracy indicated above.")
print()

print("========== Question 3.(w) ==========")
print("k Nearest Neighbors Classification")
_accuracy_grid_search(training_features, training_classes,
            k_nearest_neighbors_classification_pipeline,
            classification_knn_parameters)
print("The accuracy indicated above.")
print()

print("k Nearest Neighbors Classification with Scaling")
_accuracy_grid_search(training_features, training_classes,
            k_nearest_neighbors_classification_scale_pipeline,
            classification_knn_parameters)
print("The accuracy indicated above.")
print("Scling helps a little bit, increasing the accuracy from 0.7635 to 0.7703.")
print()
```

```
if __name__ == '__main__':
    main()
```